Comparison of Apache Mahout and Apache Spark

Sanuj Kumar
Department of Computer Science
New Mexico State University
Las Cruces, New Mexico 88001, United States

December 16, 2019

Abstract

Apache Mahout(TM) is a distributed linear algebra framework and mathematically expressive Scala DSL designed to let mathematicians, statisticians, and data scientists quickly implement their own algorithms. Apache Spark is the recommended out-of-the-box distributed back-end, or can be extended to other distributed backends[1]. The motivation behind this project is to compare both the systems in terms of performance and other metrics while training and testing on the same ML models built on and for the datasets.

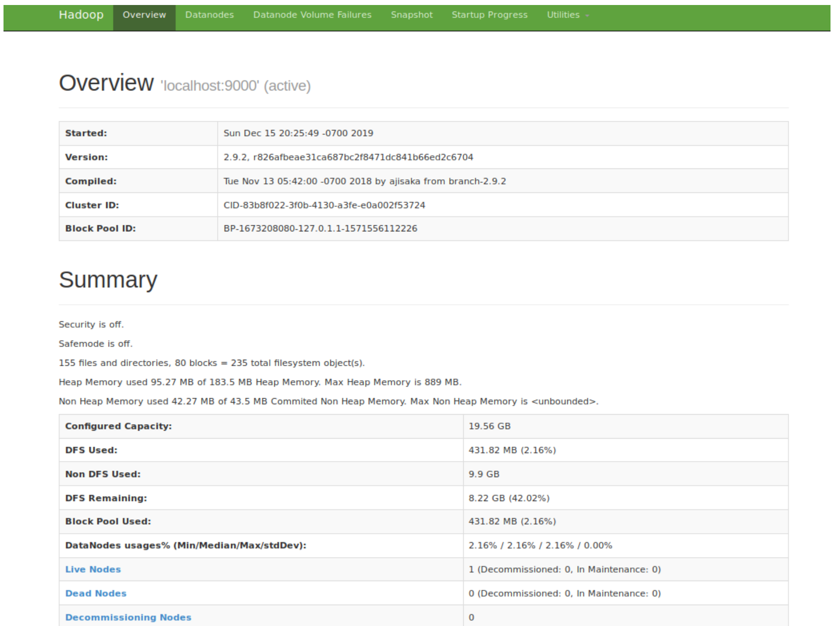
1 Introduction

The Mahout project was started by several people involved in the Apache Lucene (open source search) community with an active interest in machine learning and a desire for robust, well-documented, scalable implementations of common machine-learning algorithms for clustering and categorization. Apache mahout requires java and hadoop as its pre-requisites for installation. In this case the tests were conducted on Apache Spark v2.4.4 built for Hadoop v2.7+ and Apache Mahout v0.13 distribution, running on Hadoop v2.9.2 with JDK8 (java-1.8.0).

Apache Hadoop is an open-source framework designed for distributed storage and processing of very large data sets across clusters of computers. Apache Hadoop consists of components including:

- Hadoop Distributed File System (HDFS), the bottom layer component for storage. HDFS breaks up files into chunks and distributes them across the nodes of the cluster.
- Yarn for job scheduling and cluster resource management.

- MapReduce for parallel processing. Common libraries needed by the other Hadoop subsystems.[4]



The two systems used in the project are discussed below:-

1.1 Apache Mahout

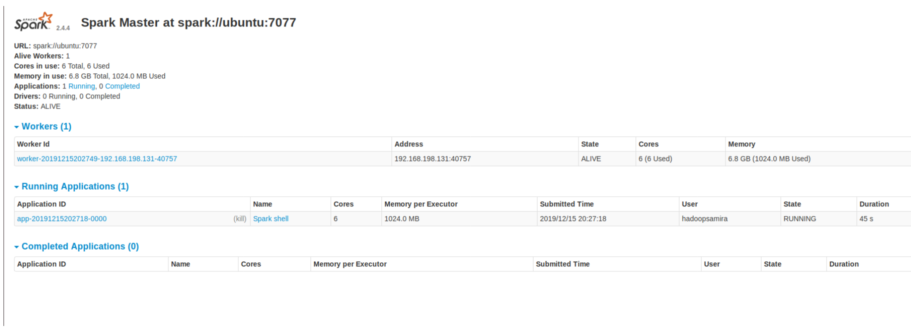
Apache Mahout is a powerful, scalable machine-learning library that runs on top of Hadoop MapReduce. Machine learning is a discipline of artificial intelligence that enables systems to learn based on data alone, continuously improving performance as more data is processed. Machine learning is the basis for many technologies that are part of our everyday lives.[2]

Apache mahout can run on hadoop as well as spark. In this case, Apache mahout was used with Map-Reduce for running the machine learning models : Naives Bayes for classification and KMeans for clustering.

1.2 Apache Spark

Apache Spark is a powerful unified analytic engine for large-scale distributed data processing and machine learning. On top of the Spark core data processing engine are libraries for SQL, machine learning, graph computation, and stream processing. These libraries can be used together in many stages in modern data pipelines and allow for code reuse across batch, interactive, and streaming applications. Spark is useful for ETL processing, analytic and machine learning workloads, and for batch and interactive processing of SQL queries, machines learning inferences, and artificial intelligence applications.[3]

Apache Spark can run on a standalone version and on top of Hadoop. In this case, the classification and clustering algorithms namely : Naives Bayes and KMeans were runned on the standalone version of spark using MLlib (Machine Learning Library for spark).



2 Literature-Review

There is a very little to no existence of a machine learning framework before Mahout, therefore, simple and extensible programming in native language was the only option. Mahout was introduced as a sub-project of Apache Lucene and was later developed as a Machine Learning framework for executing scalable ML algorithms as Map-Reduce task through collabrative filtering. In this day and age, Apache Spark is considered as a competitor of Mahout with its built-in MLlib library used for running machine learning algorithms. MLlib is a unattached collection of high-level algorithms that runs on Spark. This is what Mahout used to be the only Mahout of old was on Hadoop MapReduce. In 2014 Mahout announced it would no longer accept Hadoop Mapreduce code and completely switched new development to Spark (with other engines possibly in the offing, like H2O).

3 Data-sets

The datasets used in this project are extracted from websites and is built natively in Apache Mahout as examples.

- Classification : 20 Newsgroups Dataset - `http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz`. The Size of the dataset is 66MB with more than 18,000 files in txt.
- Clustering : Synthetic Control Chart Dataset - `https://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.data` This data

is small but is at the same highly dimensional which poses a challenge for KMeans-clustering discussed later. It contains 600 records with 60 dimensional parameters.

The dataset used for clustering has to be converted to 'libsvm' format for building the KMeans model(pyspark.ml) and later testing the model.

For the dataset used for classification all the files were first transferred to HDFS and then fed to model in a batch-processing fashion.

4 Queries

There are two query scenarios featuring both supervised and unsupervised machine learning techniques.

4.1 Classification :

Naive Bayes

- APACHE MAHOUT : Mahout currently has two Naive Bayes implementations. The first is standard Multinomial Naive Bayes. The second is an implementation of Transformed Weight-normalized Complement Naive Bayes as introduced by Rennie et al.The former is referred as Bayes and the latter as CBayes. Where Bayes has long been a standard in text classification, CBayes is an extension of Bayes that performs particularly well on datasets with skewed classes and has been shown to be competitive with algorithms of higher complexity such as Support Vector Machines Both Bayes and CBayes are currently trained via MapReduce Jobs. Testing and classification can be done via a MapReduce Job or sequentially. Mahout provides CLI drivers for preprocessing, training and testing. A Spark implementation is currently in the works[5]

```
hadoopsamira@ubuntu:/usr/local/mahout$ examples/bin/classify-20newsgroups.sh
Discovered Hadoop v2.
Setting dfs command to /usr/local/hadoop/bin/hdfs dfs, dfs rm to /usr/local/hadoop/bin/hdfs dfs -rm -r -skipTrash.
Please select a number to choose the corresponding task to run
1. cnaivebayes-MapReduce
2. naivebayes-MapReduce
3. cnaivebayes-Spark
4. naivebayes-Spark
5. sgd
6. clean-- cleans up the work area in /tmp/mahout-work-hadoopsamira
Enter your choice :
```

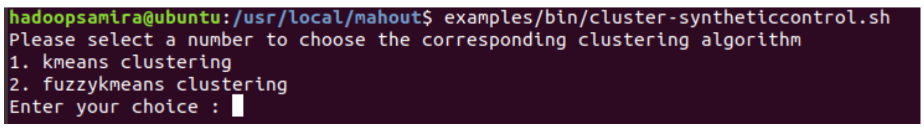
- APACHE SPARK : Naive Bayes classifiers are a family of simple probabilistic, multiclass classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between every pair of features. Naive Bayes can be trained very efficiently. With a single pass over the training data, it computes the conditional probability distribution of each feature given each label. For prediction, it applies Bayes' theorem to compute the conditional probability distribution of each label given an observation. MLlib supports both multinomial naive Bayes and Bernoulli naive Bayes.[6]

	precision	recall	f1-score	support
alt.atheism	0.88	0.93	0.90	342
comp.graphics	0.70	0.85	0.77	445
comp.os.ms-windows.misc	0.84	0.82	0.83	454
comp.sys.ibm.pc.hardware	0.79	0.79	0.79	435
comp.sys.mac.hardware	0.83	0.87	0.85	433
comp.windows.x	0.88	0.83	0.86	454
misc.forsale	0.86	0.87	0.86	432
rec.autos	0.89	0.89	0.89	452
rec.motorcycles	0.88	0.96	0.92	475
rec.sport.baseball	0.94	0.95	0.94	456
rec.sport.hockey	0.98	0.95	0.97	421
sci.crypt	0.98	0.90	0.94	442
sci.electronics	0.89	0.80	0.84	440
sci.med	0.95	0.94	0.95	451
sci.space	0.94	0.91	0.93	450
soc.religion.christian	0.90	0.92	0.91	440
talk.politics.guns	0.90	0.91	0.91	411
talk.politics.mideast	0.97	0.89	0.93	421
talk.politics.misc	0.86	0.90	0.88	351
talk.religion.misc	0.89	0.79	0.84	280
accuracy			0.88	8485
macro avg	0.89	0.88	0.88	8485
weighted avg	0.89	0.88	0.89	8485

4.2 Clustering :

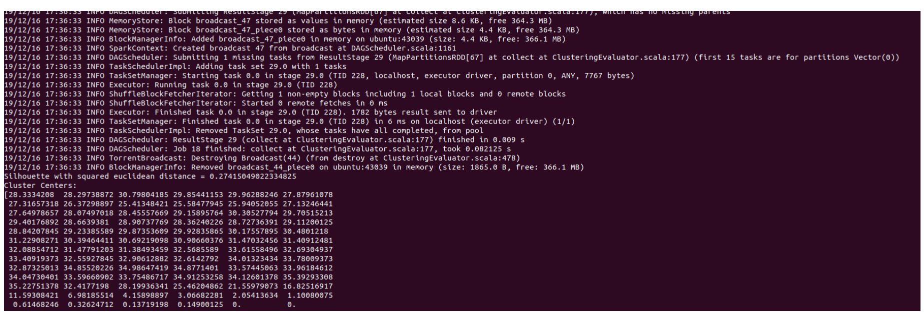
K-Means

- APACHE MAHOUT : K-Means is a simple but well-known algorithm for grouping objects, clustering. All objects need to be represented as a set of numerical features. In addition, the user has to specify the number of groups (referred to as k) she wishes to identify. Each object can be thought of as being represented by some feature vector in an n dimensional space, n being the number of all features used to describe the objects to cluster. The algorithm then randomly chooses k points in that vector space, these point serve as the initial centers of the clusters. Afterwards all objects are each assigned to the center they are closest to. Usually the distance measure is chosen by the user and determined by the learning task. After that, for each cluster a new center is computed by averaging the feature vectors of all objects assigned to it. The process of assigning objects and recomputing centers is repeated until the process converges. The algorithm can be proven to converge after a finite number of iterations. Several tweaks concerning distance measure, initial center choice and computation of new average centers have been explored, as well as the estimation of the number of clusters k. Yet the main principle always remains the same.[7]



```
hadoopsamira@ubuntu:/usr/local/mahout$ examples/bin/cluster-syntheticcontrol.sh
Please select a number to choose the corresponding clustering algorithm
1. kmeans clustering
2. fuzzykmeans clustering
Enter your choice :
```

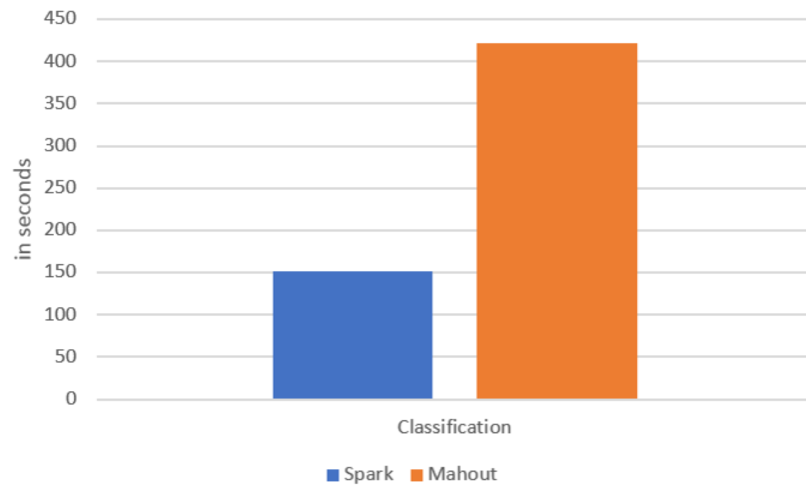
- APACHE SPARK : k-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The MLlib implementation includes a parallelized variant of the k-means++ method called kmeans. KMeans is implemented as an Estimator and generates a KMeansModel as the base model.[6]

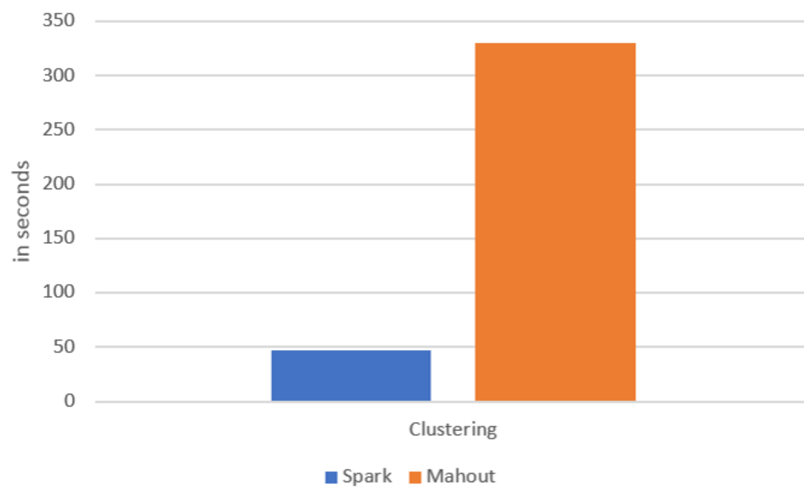


```
19/12/16 17:36:33 INFO MemoryStore: Block broadcast_47 stored as values in memory (estimated size 8.6 KB, free 364.3 MB)
19/12/16 17:36:33 INFO MemoryStore: Block broadcast_47_piece0 stored as bytes in memory (estimated size 4.4 KB, free 364.3 MB)
19/12/16 17:36:33 INFO BlockManagerInfo: Added broadcast_47_piece0 in memory on ubuntu:43039 (size: 4.4 KB, free: 366.1 MB)
19/12/16 17:36:33 INFO SparkContext: Created broadcast 47 from broadcast at DAGScheduler.scala:1161
19/12/16 17:36:33 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 29 (MapPartitionsRDD[67] at collect at ClusteringEvaluator.scala:177) (first 15 tasks are for partitions Vector(0))
19/12/16 17:36:33 INFO TaskSchedulerImpl: Adding task set 29.0 with 1 tasks
19/12/16 17:36:33 INFO TaskSetManager: Starting task 0.0 in stage 29.0 (TID 228, localhost, executor driver, partition 0, ANY, 7767 bytes)
19/12/16 17:36:33 INFO Executor: Running task 0.0 in stage 29.0 (TID 228)
19/12/16 17:36:33 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks including 1 local blocks and 0 remote blocks
19/12/16 17:36:33 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
19/12/16 17:36:33 INFO Executor: Finished task 0.0 in stage 29.0 (TID 228). 1782 bytes result sent to driver
19/12/16 17:36:33 INFO TaskSetManager: Finished task 0.0 in stage 29.0 (TID 228) in 6 ms on localhost (executor driver) (1/1)
19/12/16 17:36:33 INFO TaskSchedulerImpl: Removed TaskSet 29.0, whose tasks have all completed, from pool
19/12/16 17:36:33 INFO DAGScheduler: ResultStage 29 (collect at ClusteringEvaluator.scala:177) finished in 0.009 s
19/12/16 17:36:33 INFO DAGScheduler: Job 18 finished: collect at ClusteringEvaluator.scala:177, took 0.082125 s
19/12/16 17:36:33 INFO TorrentBroadcast: Destroying Broadcast(44) (from destroy at ClusteringEvaluator.scala:478)
19/12/16 17:36:33 INFO BlockManagerInfo: Removed broadcast_44_piece0 on ubuntu:43039 in memory (size: 1865.0 B, free: 366.1 MB)
Silhouette with squared euclidean distance = 0.27415049022334825
Cluster Centers:
[28.3334208  28.29738872 30.79804185 29.85441153 29.96288246 27.87961078
 27.31657318 26.37298897 25.41348421 25.58477945 25.94052055 27.13246441
 27.64978657 28.07497018 28.45557669 29.15895764 30.30527794 29.70515213
 29.40176892 28.6639381  28.90737769 28.36240226 28.72736391 29.11200125
 28.84207845 29.23385589 29.87353609 29.92835865 30.17557895 30.4801218
 31.22908271 30.39464411 30.69219098 30.90660376 31.47032456 31.40912481
 32.08854712 31.47791203 31.38493459 32.5685589  33.61558496 32.69304937
 33.40919373 32.55927845 32.90612882 32.6142792  34.01323434 33.78009373
 32.87325013 34.85520226 34.98647419 34.8771401  33.57445063 33.96184612
 34.04730401 33.59660902 33.75486717 34.91253258 34.12601378 35.39293308
 35.22751378 32.4177198  28.19936341 25.46204862 21.55979073 16.82516917
 11.59308421  6.98185514  4.15898897  3.06682281  2.05413634  1.10080075
  0.61468246  0.32624712  0.13719198  0.14900125  0.          0.
```

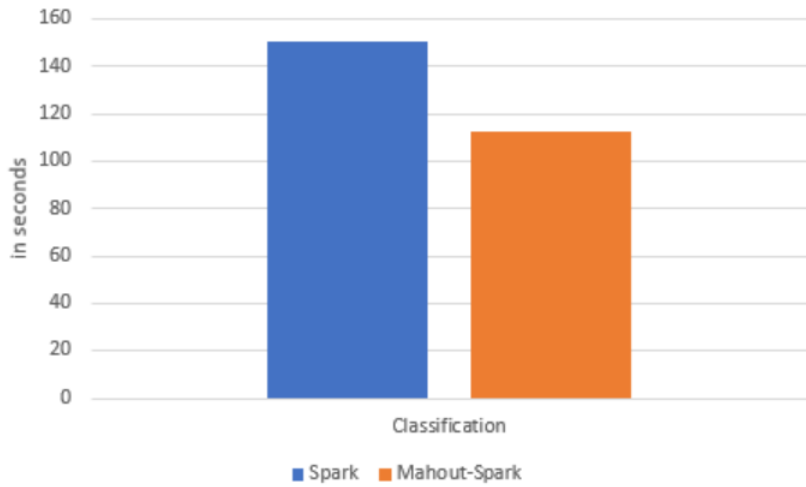
5 Performance Evaluation

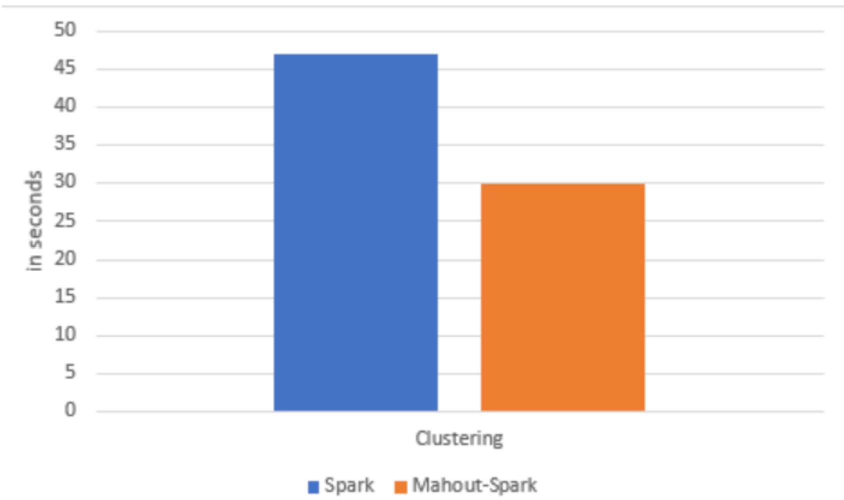
5.1 Mahout-Mapreduce Vs Spark-MLlib





5.2 Mahout-Mapreduce Vs Spark-MLlib





In terms of time taken and resource usage MLlib running on Apache Spark framework proves to be much faster than the Map-Reduce implementation of Apache Mahout. This can be considered as shortcoming in terms of architecture of Map-Reduce being slower than a DAG execution model. Apache Spark has an optimized execution which allows it to perform much better than Mahout-MapReduce. Apache Mahout can also work on top of Spark, so a final comparison was done between Apache Mahout (running on Spark framework) and Spark-MLlib to get the best case scenario.
The results clearly show that when Apache Mahout operates on top of Spark as its base execution engine then it produces the best result in terms of performance.

References

[1] Apache Mahout. Retrieved from https://mahout.apache.org/.

[2] Apache Mahout. Retrieved from https://mapr.com/products/product-overview/apache-mahout/.

[3] Apache Spark. Retrieved from https://mapr.com/products/apache-spark/.

[4] Apache Hadoop. Retrieved from https://mapr.com/products/apache-hadoop/.

[5] Naive Bayes. Retrieved from http://mahout.apache.org/users/classification/naivebayes.html.

[6] Naive Bayes - RDD-based API. (n.d.). Retrieved from http://spark.apache.org/docs/latest/mllib-naive-bayes.html.

[7] KMeans. Retrieved from http://mahout.apache.org/users/clustering/k-means-clustering.html.