Distributed Systems Seminar

Topic: Evaluate the DASK distributed computing framework in respect to various scientific computing tasks

Task: The list of related software packages along with the description and brief evaluation

I have choosen for 5 software packages for evaulation:

1. Apache Spark - is an open-source unified analytics engine for large-scale data processing. [1]

2. MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. [2]

3. The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. [3]

4. Dask is a flexible open-source Python library for parallel computing. [4]

5. Vaex is a python library that helps to visualize and explore big tabular datasets. It is a high performance library and can solve many of the shortcomings of pandas. [5]

First of all, I searched the literatures in order to get familiarize myself with the choosen software packages and investigate their performance evaulations. The literatures that I have choosen for that task are as following:

- The main difference between the two frameworks is that MapReduce processes data on disk whereas Spark processes and retains data in memory for subsequent steps. [6]
- Performance Evaluation of Apache Spark Vs MPI [7]
- Scaling Pandas: Comparing Dask, Ray, Modin, Vaex, and RAPIDS [8]

After reading and investigating the literatures, I have decided to look at performance evaulation of choosen software packages as following:

- Apache Spark & MapReduce
- Apache Spark & Open MPI
- Dask & Vaex

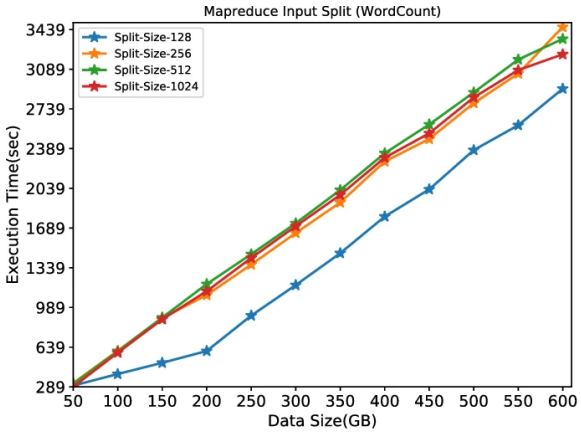
Apache Spark & MapReduce

The main difference between the two frameworks is that MapReduce processes data on disk whereas Spark processes and retains data in memory for subsequent steps. [6]

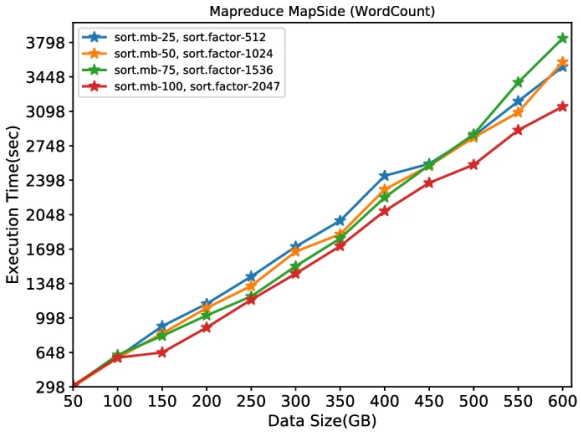
According to the *A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench* article, Spark has better performance as compared to Hadoop by two times with WordCount work load and 14 times with Tera-Sort workloads respectively when default parameters are tuned with new values. Further more, the throughput and speedup results show that Spark is more stable and faster than Hadoop because of Spark data processing ability in memory instead of store in disk for the map and reduced function. The authors have also found that Spark performance degraded when input data was larger.

As a workload, the authors have chosen WordCount [9] and TeraSort [10]. Some of the experiments and results of the articles are follows:

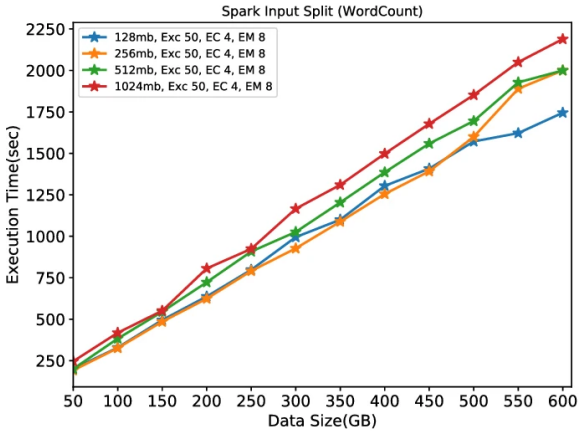
- Execution time



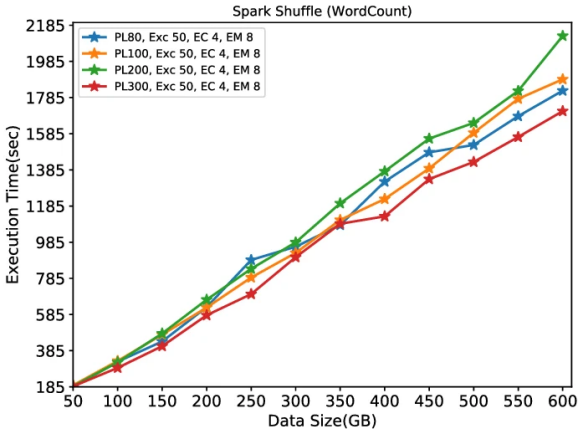
a



b

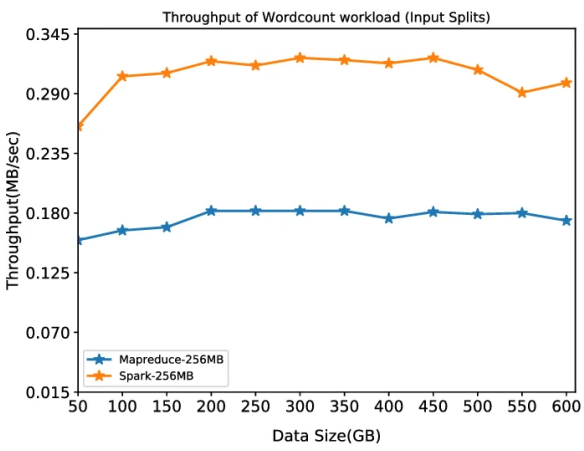


c

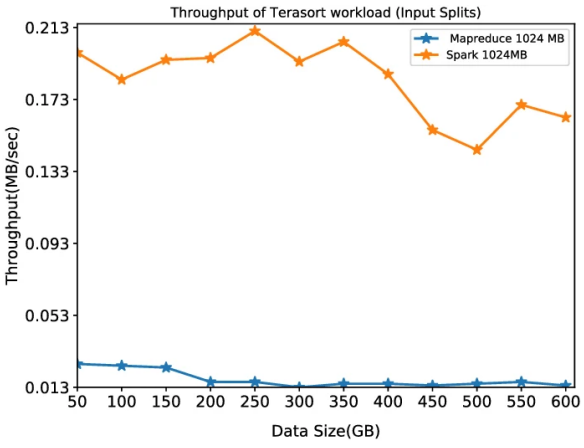


d

- Throughput



a



b

Apache Spark & Open MPI

Accorting to the *Performance Evaluation of Apache Spark Vs MPI* article, better performance on MPI environment than Spark was due to two reasons:

1. In MPI, the freedom of the programmer to choose the memory requirements, in terms of the number of lines of tweets to be executed to avail better performance. Whereas in Spark, the memory allocation and task scheduling are purely under the control of Spark processing Framework and programmer intervention is not possible.

2. C++ is more flexible programming language than Scala.

Some of the experiments and results of the articles are follows:

- Sentiment analysis on apache spark (Scala) (100 GB)

Exe no/ITN no.	1	2	3	4	5	6	7	8	9	10
Exe.Time(min)	3.6	3.5	3.6	3.5	3.6	3.6	3.5	3.6	3.6	3.7
CPU Utilization % (W1)	58.8	52.0	60.0	46.3	62.0	60.0	88.8	61.8	58.8	52.0
CPU Utilization % (W2)	58.3	84.0	62.0	64.8	63.0	56.5	66.8	72.0	58.3	84.0
CPU Utilization % (W3)	49.3	50.8	67.5	84.3	58.0	86.0	85.8	83.8	49.3	50.8
CPU Utilization % (W4)	60.0	66.3	76.0	67.0	68.0	59.0	78.8	53.5	60.0	66.3
CPU Utilization % (W5)	46.5	51.3	74.5	73.3	53.3	69.5	82.5	74.0	46.5	51.3
CPU Utilization % (W6)	63.0	71.3	54.0	68.8	57.8	82.0	67.8	73.8	63.0	71.3
CPU Utilization% (W7)	74.5	62.5	50.0	59.5	76.3	49.0	68.5	54.8	74.5	62.5
CPU Utilization % (W8)	77.5	80.0	58.5	61.8	70.8	50.3	65.3	54.5	77.5	80.0
CPU Utilization % (W9)	57.8	62.0	59.3	50.3	78.8	60.5	62.5	81.5	57.8	62.0
CPU Utilization % (W10)	63.0	66.5	55.3	69.5	72.3	74.0	49.8	76.3	63.0	66.5

- Sentiment analysis on MPI (C/C++)(100 GB)

	Master	S1	S2	S3	S4	S5	S6	S7	S8	S9	S 10
Iteration 1											
CPU Utn %	88.8	89.8	90.3	89.5	89.3	89.0	89.0	89.5	89.5	89.5	89.5
Exec. Time (min)	**1.56**										
Iteration 2											
CPU Utilization %	89.8	90.5	90.5	89.8	90.0	89.8	90.0	90.3	90.3	90.3	90.0
Exec. Time(min)	**1.57**										
Iteration 3											
CPU Utilization %	89.5	89.8	90.8	89.8	89.8	89.5	89.8	89.8	90.0	90.3	90.0
Exec. Time(min)	**1.57**										
Iteration 4											
CPU Utilization %	89.8	90.0	90.5	89.3	89.5	89.8	89.3	89.3	89.5	89.8	90.0
Exec. Time(min)	**1.56**										
Iteration 5											
CPU Utilization %	90.0	90.0	90.3	89.3	89.8	90.5	89.5	89.5	89.8	90.0	90.5
Exec. Time(min)	**1.57**										
Iteration 6											
CPU Utilization %	89.5	90.0	90.8	90.0	90.0	89.8	89.3	90.0	90.3	90.0	90.5
Exec. Time(min)	**1.57**										
Iteration 7											
CPU Utilization %	89.5	89.8	90.5	90.3	90.3	89.8	89.5	89.8	90.3	89.5	90.3
Exec. Time(min)	**1.56**										
Iteration 8											
CPU Utilization %	89.3	90.3	90.5	90.0	89.3	90.3	89.8	89.5	90.5	90.0	89.8
Exec. Time(min)	**1.57**										
Iteration 9											
CPU Utilization %	89.3	90.0	90.3	90.3	89.8	90.0	89.3	90.5	90.0	90.0	90.3
Exec. Time(min)	**1.57**										

- Execution times on 100 GB/500 GB/1 TB datasets in spark processing

Data size	ITN1	ITN2	ITN3	Avg. Exe. Time (Min)
100 GB	3.6	3.5	3.6	3.58
500 GB	17.0	16.0	17.0	16.67
I TB	33.0	32.0	32.0	32.33

- Execution times on 100GB/500GB/1TB datasets in MPI

Data size	ITN1	ITN2	ITN3	Avg. Exe. Time (Min)
100 GB	1.56	1.57	1.57	1.56
500 GB	10.04	10.07	10.06	10.06
I TB	22.18	22.36	22.17	22.23

Dask & Vaex

Dask is better thought of as two projects: a low-level Python scheduler and a higher-level Dataframe module [8]

According to the *Scaling Pandas: Comparing Dask, Ray, Modin, Vaex, and RAPIDS*, Dask is more focused on letting you scale your code to compute clusters, while Vaex makes it easier to work with large datasets on a single machine.

Dask main scaling strategy is *clusters*, while Vaex main scaling strategy is *lazy loading*. Their scaling abilities are changing regarding the data size. For example, the scaling ability is *1 TB+* in Dask, and *100 GB+* in Vaex.

References:

[1] Apache Spark. https://en.wikipedia.org/wiki/Apache_Spark

[2] MapReduce Documents. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[3] Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/

[4] Dask (software). https://en.wikipedia.org/wiki/Dask_(software)

[5] Vaex. https://vaex.io/

[6] Difference Between MapReduce and Spark. http://www.differencebetween.net/technology/difference-between-mapreduce-and-spark

[7] Performance Evaluation of Apache Spark Vs MPI: A Practical Case Study on Twitter Sentiment Analysis. https://thescipub.com/pdf/10.3844/jcssp.2017.781.794

[8] Scaling Pandas: Comparing Dask, Ray, Modin, Vaex, and RAPIDS. https://www.datarevenue.com/en-blog/pandas-vs-dask-vs-vaex-vs-modin-vs-rapids-vs-ray

[9] WordCount. https://cwiki.apache.org/confluence/display/hadoop2/wordcount

[10] Terasort. https://accumulo.apache.org/1.7/examples/terasort