Cloud Computing
PA 2 (Part-B)
Sort on Hadoop/Spark

Name: Sharanheer Choudhari (A20398615)

Goals:

To implement sorting through Hadoop/Spark on multiple nodes and perform different experiments including weak scaling and strong scaling. For this experiment we are trying to sort 8GB, 20GB and 80GB file.

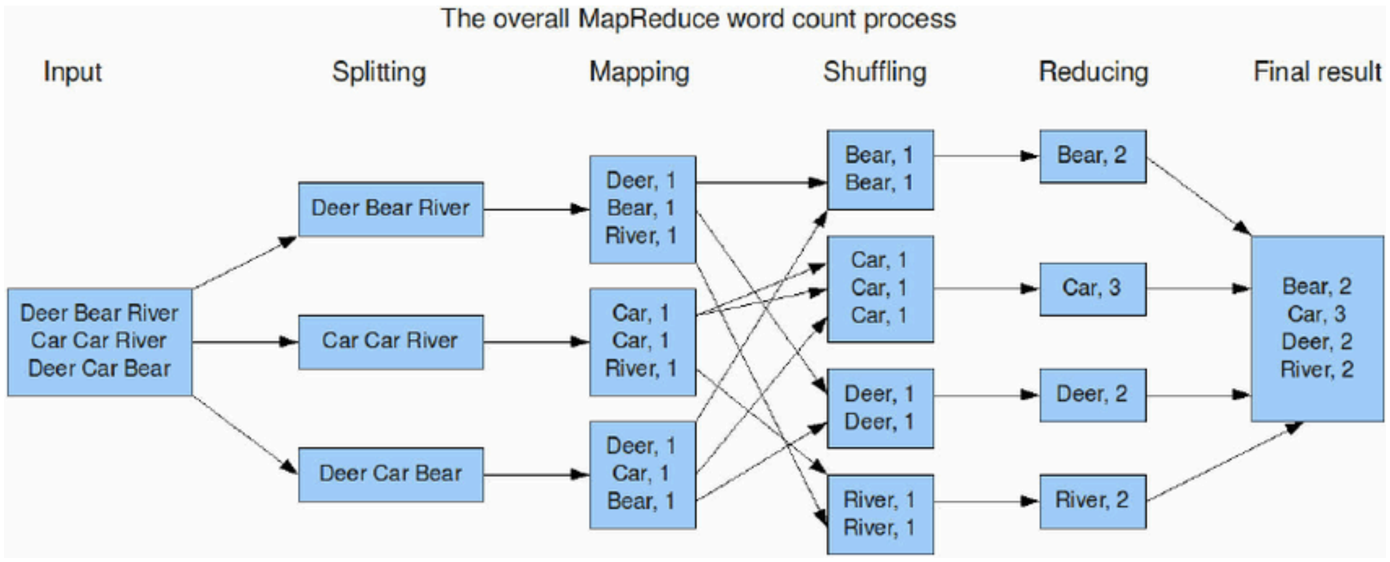
Overview & Methodology:

Hadoop

Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

Mapreduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a map procedure (or method), which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a *reduce* method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).



Spark

Apache Spark is an open-source cluster-computing framework.
Apache Spark has as its architectural foundation the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.

Scaling

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth.[1] For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added.

- *Strong Scaling*

In this case the problem size stays fixed but the number of processing elements are increased. This is used as justification for programs that take a long time to run (something that is cpu-bound). The goal in this case is to find a "sweet spot" that allows the computation to complete in a reasonable amount of time, yet does not waste too many cycles due to parallel overhead. In strong scaling, a program is considered to scale *linearly* if the *speedup* (in terms of work units completed per unit time) is equal to the number of processing elements used (N). In general, it is harder to achieve good strong-scaling at larger process counts since the communication overhead for many/most algorithms increases in proportion to the number of processes used.

- *Weak Scaling*

In this case the problem size (workload) assigned to each processing element stays constant and additional elements are used to solve a larger total problem (one that wouldn't fit in RAM on a single node, for example). Therefore, this type of measurement is justification for programs that take a lot of memory or other system resources (something that is memory-bound). In the case of weak scaling, linear scaling is achieved if the run time stays constant while the workload is increased in direct proportion to the number of processors. Most programs running in this mode should scale well to larger core counts as they typically employ nearest neighbor communication patterns where the communication overhead is relatively constant regardless of the number of processes used.

Experiment 1:

Performance evaluation of sort (weak scaling – small dataset)

Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 8GB)	Spark Sort (4VM 8GB)
Computation Time (sec)	142	30	221	176
Data Read (GB)	2	2	16	8
Data Write (GB)	2	2	16	8
I/O Throughput (MB/sec)	28.84	136.53	144.79	90.91
Speedup	n/a	n/a	2.57	3.227
Efficiency	n/a	n/a	64.25	80.675

Analysis (weak scaling - small dataset)

Hadoop:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 144.79
Speed Up = (Shared Memory Computation Time * 4) / Hadoop Sort Computation Time = 2.57
Efficiency = Speed Up / No. of Cores = 64.25

Spark:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 90.91
Speed Up = (Shared Memory Computation Time * 4) / Spark Sort Computation Time = 3.227
Efficiency = Speed Up / No. of Cores = 80.675

Thus, Spark had better performance compare to Hadoop.

Experiment 2:

Performance evaluation of sort (strong scaling – large dataset)

Experiment	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)	Hadoop Sort (4VM 20GB)	Spark Sort (4VM 20GB)
Computation Time (sec)	731	393	558.2	471
Data Read (GB)	40	40	40	20
Data Write (GB)	40	40	40	20
I/O Throughput (MB/sec)	112.065	208.44	143.31	84.925
Speedup	n/a	n/a	1.309	1.55
Efficiency	n/a	n/a	32.73	38.75

Analysis (strong scaling - large dataset)

Hadoop:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 143.31
Speed Up = Shared Memory Computation Time / Hadoop Sort Computation Time = 1.309
Efficiency = Speed Up / No. of Cores = 32.73

Spark:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 84.925
Speed Up = Shared Memory Computation Time / Spark Sort Computation Time = 1.55
Efficiency = Speed Up / No. of Cores = 38.75

Thus, Spark had better performance compare to Hadoop.

Experiment 3:

Experiment	**Shared Memory** **(1VM 20GB)**	**Linux Sort** **(1VM 20GB)**	**Hadoop Sort** **(4VM 80GB)**	**Spark Sort** **(4VM 80GB)**
Computation Time (sec)	731	393	3360	2820
Data Read (GB)	40	40	160	80
Data Write (GB)	40	40	160	80
I/O Throughput (MB/sec)	112.065	208.44	95.23	56.74
Speedup	n/a	n/a	0.87	1.036
Efficiency	n/a	n/a	21.75	25.9

Performance evaluation of sort (weak scaling – large dataset)

Analysis (weak scaling - large dataset)

Hadoop:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 95.23
Speed Up = (Shared Memory Computation Time * 4) / Hadoop Sort Computation Time = 0.87
Efficiency = Speed Up / No. of Cores = 21.75

Spark:
I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec) = 56.74
Speed Up = (Shared Memory Computation Time * 4) / Spark Sort Computation Time = 1.036
Efficiency = Speed Up / No. of Cores = 25.9

Thus, Spark had better performance compare to Hadoop.

Conclusion:

- Performed sorting on different dataset (8GB, 20GB, 80GB) using Hadoop and Spark framework in java. Also, verified the sorting results using valsort and compared the results.
- Performed weak and strong scaling experiment using different dataset sizes (8GB, 20GB, 80GB) on the proton cluster having 4 nodes with 4 cores each, 8GB of memory, and 80GB of SSD storage.

References:

- http://spark.apache.org/
- http://hadoop.apache.org/
- https://en.wikipedia.org/wiki/Scalability
- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- https://www.youtube.com/watch?v=SqvAaB3vK8U&t=6s
- https://www.youtube.com/watch?v=Y6oit3rCsZo
- https://www.dezyre.com/article/hadoop-architecture-explained-what-it-is-and-why-it-matters/317
- https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/
- https://stackoverflow.com/questions/40814781/printing-values-of-javapairrdddouble-double-in-java
- http://spark.apache.org/docs/1.2.0/api/java/org/apache/spark/api/java/JavaPairRDD.html
- https://spark.apache.org/docs/0.8.1/java-programming-guide.html
- http://timepasstechies.com/spark-sortby-sortbykey-example-java-scala/