

**Experiment No.6 Title: Word count processing on Hadoop**

# Batch:B4 Roll No.:1724008 Experiment No.:6 Title: Word count processing on Hadoop

**Resources needed: Internet, Hadoop**

**Theory:**

**Big Data processing frameworks:**

Since the data is big (Volume, verity, veracity etc.), it requires specialised high performance platform to process it [1][2]. Following frameworks are available for big data processing. Most of the frameworks are available as platform as service on cloud environment such as Cloudera [4].

# Hadoop

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers.

# Spark

Spark is another successor in the Big Data processing domain. Spark and Hadoop are often contrasted as an "either/or" choice, but that isn't really the case. The Hadoop ecosystem can accommodate the Spark processing engine in place of MapReduce, leading to all sorts of different environment make-ups that may include a mix of tools and technologies from both ecosystems. As one specific example of this interplay, Big Data powerhouse Cloudera is now replacing MapReduce with Spark as the default processing engine in all of its Hadoop implementations moving forward. As another example, Spark does not include its own distributed storage layer, and as such it may take advantage of Hadoop's distributed filesystem (HDFS), among other technologies unrelated to Hadoop [3].

Spark differs from Hadoop and the MapReduce paradigm in that it works in-memory, speeding up processing times. Spark also eliminates the imposed linear dataflow of Hadoop's default MapReduce engine, allowing for a more flexible pipeline construction.

# Flink

Apache Flink is a streaming dataflow engine, aiming to provide facilities for distributed computation over streams of data. Treating batch processes as a special case of streaming data, Flink is effectively both a batch and real-time processing framework, but one which clearly puts streaming first.

Flink provides a number of APIs, including a streaming API for Java and Scala, a static data API for Java, Scala, and Python, and an SQL-like query API for embedding in Java and Scala code. It also has its own machine learning and graph processing libraries. Flink has an impressive set of additional features, including:

* + High Performance & Low Latency
  + Support for Event Time and Out-of-Order Events
  + Exactly-once Semantics for Stateful Computations
  + Continuous Streaming Model with Backpressure
  + Fault-tolerance via Lightweight Distributed Snapshots

# Storm

Apache Storm is a distributed real-time computation system, whose applications are designed as directed acyclic graphs. Storm is designed for easily processing unbounded streams, and can be used with any programming language. It has been benchmarked at processing over one million tuples per second per node, is highly scalable, and provides processing job guarantees. Unique for items on this list, Storm is written in Clojure, the Lisp-like functional-first programming language.

Apache Storm can be used for real-time analytics, distributed machine learning, and numerous other cases, especially those of high data velocity. Storm can run on YARN and integrate into Hadoop ecosystems, providing existing implementations a solution for real-time stream processing. Five characteristics which make Storm ideal for real- time processing workloads are (taken from HortonWorks) [3]:

* + Fast - benchmarked as processing one million 100 byte messages per second per node
  + Scalable - with parallel calculations that run across a cluster of machines
  + Fault-tolerant - when workers die, Storm will automatically restart them. If a node dies, the worker will be restarted on another node.
  + Reliable - Storm guarantees that each unit of data (tuple) will be processed at least once or exactly once. Messages are only replayed when there are failures.
  + Easy to operate - standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate.

# Procedure:

**Perform following steps in order to perform word count on Hadoop platform.**

**Execute following steps on Hadoop system,**

**Steps:**

1. **$ mkdir temp**
2. **$ cd temp**
3. **Create file named “data.txt” containing big data and store it in temp folder.**
4. **$ hdfs dfs -mkdir –p /user/hadoop/input**
5. **$ hdfs dfs -put /home/temp/data.txt /user/hadoop/input/**
6. **$ cd $HADOOP\_HOME**
7. **$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar**
8. **$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/ hadoop/input/data.txt /user/hadoop/output1**
9. **$ hdfs dfs -ls /user/hadoop/output1**
10. **$ hdfs dfs -cat /user/hadoop/output1/part-r-00000**

# Results: (Document printout as per the format)

# 

# 

# 

# 

**Outcome:**

**Comprehend fundamentals of Hadoop, Map Reduce and NO SQL.**

**Conclusion: (Conclusion to be based on the objectives and outcomes achieved)**

**Thus we go to know how map-reduce works and also its use in the process like word-count.**

**Grade: AA / AB / BB / BC / CC / CD /DD**



**Signature of faculty in-charge with date References:**

**Books/ Journals/ Websites:**

[1]Paul Zikopoulos, Chris Eaton, “Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data’, McGraw Hill Education [2]<http://www.navint.com/images/Big.Data.pdf>

[3] <http://www.kdnuggets.com/2016/03/top-big-data-processing-frameworks.html>