**Hadoop – Introduction & Features**

Let us start with what is Hadoop and what are Hadoop features that make it so popular.

Hadoop is an open-source software framework for distributed storage and distributed processing of extremely large data sets. Important features of Hadoop are:

* Hadoop is an open source project. It means its code can be modified to business requirements.
* In Hadoop, data is highly available and accessible despite hardware failure due to multiple copies of data. If a machine or any hardware crashes, then data will be accessed from another path.
* Hadoop is highly scalable, as the new hardware can be easily added to the node. Hadoop also provides horizontal scalability which means nodes can be added on the fly without any downtime.
* Hadoop is fault tolerant, as by default 3 replicas of each block is stored across the cluster. So if any node goes down, data on that node can be recovered from the other node easily.
* In Hadoop, data is reliably stored on the cluster despite machine failure due to replication of data on the cluster.
* Hadoop runs on a cluster of commodity hardware which is not very expensive.
* Hadoop is very easy to use, as there is no need of client to deal with distributed computing; the framework takes care of all the things.

## Limitations of Hadoop

Various limitations of Hadoop are discussed below in this section along with their solution-

### **1. Issue with Small Files**

Hadoop is not suited for small data. Hadoop distributed file system lacks the ability to efficiently support the random reading of small files because of its high capacity design.

Small files are the major problem in [HDFS](http://data-flair.training/blogs/apache-hadoop-hdfs-introduction-tutorial/). A small file is significantly smaller than the[HDFS block](http://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/)size (default 128MB). If we are storing these huge numbers of small files, HDFS can’t handle these lots of files, as HDFS was designed to work properly with a small number of large files for storing large data sets rather than a large number of small files. If there are too many small files, then the NameNode will be overloaded since it stores the namespace of HDFS.

**Solution**

* Solution to deal with small file issue is simple merge the small files to create bigger files and then copy bigger files to HDFS.
* **HAR files** (Hadoop Archives) were introduced to reduce the problem of lots files putting pressure on the namenode’s memory. By building a layered filesystem on the top of HDFS, HAR files works. Using Hadoop archive command, HAR files are created, which runs a [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) job to pack the files being archived into a small number of HDFS files. Reading through files in a HAR is not more efficient than reading through files in HDFS. Since each HAR file access requires two index files read as well the data file to read, this makes it slower.
* **Sequence files**work very well in practice to overcome the ‘small file problem’, in which we use the filename as the key and the file contents as the value. By writing a program for files (100 KB), we can put them into a single Sequence file and then we can process them in a streaming fashion operating on the Sequence file. MapReduce can break Sequence file into chunks and operate on each chunk independently because Sequence file is splittable.
* Storing files in **[HBase](http://data-flair.training/blogs/hbase-tutorial-beginners-guide/)**is a very common design pattern to overcome small file problem with HDFS. We are not actually storing millions of small files into HBase, rather adding the binary content of the file to a cell.

### **2. Processing speed**

In Hadoop, with a parallel and distributed algorithm, MapReduce process large data sets. There are tasks that need to be performed: [Map](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/) and [Reduce](http://data-flair.training/blogs/reducer-in-hadoop-mapreduce/) and, MapReduce requires a lot of time to perform these tasks thereby increasing latency. Data is distributed and processed over the cluster in MapReduce which increases the time and reduces processing speed.

**Solution**

Spark has overcome this issue, by in-memory processing of data. In-memory processing is faster as no time is spent in moving the data/processes in and out of the disk. Spark is 100 times faster than MapReduce as it processes everything in memory. Flink is also used, as it processes faster than spark because of its streaming architecture and Flink may be instructed to process only the parts of the data that have actually changed, thus significantly increases the performance of the job.

### **3. Support for batch processing only**

Hadoop supports batch processing only, it does not process streamed data, and hence overall performance is slower. MapReduce framework of Hadoop does not leverage the memory of the Hadoop cluster to the maximum.

**Solution**

Spark improves the performance, but [Spark stream processing](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/) is not as much efficient as Flink as it uses micro-batch processing. Flink improves the overall performance as it provides single run-time for the streaming as well as batch processing. Flink uses native closed loop iteration operators which make machine learning and graph processing faster.

### **4. Data processing**

[Hadoop](http://data-flair.training/blogs/install-cloudera-hadoop-cdh5-ubuntu/) is designed for batch processing, that means it take a huge amount of data in input, process it and produce the result. Although batch processing is very efficient for processing a high volume of data, but depending on the size of the data being processed and computational power of the system, an output can be delayed significantly. Hadoop is not suitable for Real-time data processing.

**Solution**

* Apache Spark supports stream processing. Stream processing involves continuous input and output of data. It emphasizes on the velocity of the data, and data is processed within a small period of time.
* Apache Flink provides single run-time for the streaming as well as batch processing, so one common run-time is utilized for data streaming application and batch processing application. Flink is a stream processing system that is able to process row after row in real time.

### **5. Iterative processing**

Hadoop is not so efficient for iterative processing, as Hadoop does not support cyclic data flow (i.e. a chain of stages in which each output of the previous stage is the input to the next stage).

**Solution**

Apache Spark can be used to overcome this issue, as it accesses data from RAM instead of disk, which dramatically improves the performance of iterative algorithms that access the same dataset repeatedly. Spark iterates its data in batches. For iterative processing in Spark, each iteration has to be scheduled and executed separately.

### **6. Latency**

In [Hadoop](http://data-flair.training/blogs/install-configure-apache-hadoop-2-7-x-on-ubuntu/), MapReduce framework is comparatively slower, since it is designed to support different format, structure and huge volume of data. In MapReduce, Map takes a set of data and converts it into another set of data, where individual element are broken down into key value pair and Reduce takes the output from the map as input and process further and MapReduce requires a lot of time to perform these tasks thereby increasing latency.

**Solution**

Spark is used to reduce this issue, Apache spark is yet another batch system but it is relatively faster since it caches much of the input data on memory by RDD and keeps intermediate data in memory itself. Flink’s data streaming achieves low latency and high throughput.

### **7. Ease of use**

In Hadoop, MapReduce developers need to hand code for each and every operation which makes it very difficult to work. MapReduce has no interactive mode, but adding one such as hive and pig makes working with MapReduce a little easier for adopters.

**Solution**

While Spark can be used for such issue, Spark has interactive mode so that developers and users alike can have intermediate feedback for queries and other action. Spark is easy to program as it has tons of high-level operators. Flink can also be easily used as it also has high-level operators.

### **8. Security**

Hadoop can be challenging in managing the complex application.  If the user doesn’t know how to enable platform who is managing the platform, your data could be at huge risk. At storage and network levels, Hadoop is missing encryption, which is a major point of concern. Hadoop supports Kerberos authentication, which is hard to manage.

[HDFS](http://data-flair.training/blogs/top-hadoop-hdfs-commands-tutorial/) supports access control lists (ACLs) and a traditional file permissions model. However, third party vendors have enabled an organization to leverage Active Directory Kerberos and LDAP for authentication.

**Solution**

Spark provides security bonus. If we run spark in [HDFS](http://data-flair.training/blogs/most-used-hdfs-commands-tutorial-examples/), it can use HDFS ACLs and file-level permissions. Additionally, Spark can run on [YARN](http://data-flair.training/blogs/hadoop-yarn-tutorial/) giving it the capability of using Kerberos authentication.

### **9. Abstraction**

Hadoop does not have any type of abstraction so MapReduce developers need to hand code for each and every operation which makes it very difficult to work.

**Solution**

To overcome this, Spark is used in which for batch we have RDD abstraction. Flink has Dataset abstraction.

### **10. Vulnerable by nature**

[Hadoop](http://data-flair.training/blogs/install-hadoop-2-x-ubuntu-hadoop-multi-node-cluster/) is entirely written in java, a language most widely used, hence java been most heavily exploited by cyber criminals and as a result, implicated in numerous security breaches.

### **11. Caching**

Hadoop is not efficient for caching. In Hadoop, MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.

**Solution**

Spark and Flink can overcome this, as Spark and Flink cache data in memory for further iterations which enhance the overall performance.

### **12. Line of code**

Hadoop has 1, 20,000 line of code, the number of lines produces the number of bugs and it will take more time to execute the program.

**Solution**

Although Spark and Flink are written in[scala](http://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/)and java but they are implemented in scala, so the number of line of code is lesser than Hadoop. So it will also take less time to execute the program.

### **13. Uncertainty**

Hadoop only ensures that data job is complete, but it’s unable to guarantee when the job will be complete.