Big Limitations of Hadoop for Big Data Analytics

We will discuss various limitations of Hadoop in this section along with their solution:

1. Issue with Small Files

Hadoop does not suit for small data. [**(HDFS)** **Hadoop distributed file system**](https://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) 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. A small file is significantly smaller than the[**HDFS block**](https://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 this much of files, as HDFS is for working 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 get overload since it stores the namespace of HDFS.

**Solution-**

* Solution to this Drawback of Hadoop to deal with small file issue is simple. Just merge the small files to create bigger files and then copy bigger files to HDFS.
* The introduction of**HAR files** (Hadoop Archives) was for reducing the problem of lots of files putting pressure on the namenode’s memory. By building a layered filesystem on the top of HDFS, HAR files works. Using the Hadoop archive command, HAR files are created, which runs a [**MapReduce**](https://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 the Sequence file into chunks and operate on each chunk independently because the Sequence file is splittable.
* Storing files in [**HBase is a very common design pattern**](https://data-flair.training/blogs/hbase-tutorial-beginners-guide/) 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. Slow Processing Speed

In Hadoop, with a parallel and distributed algorithm, the MapReduce process large data sets. There are tasks that we need to perform: Map and Reduce 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-**

As a Solution to this Limitation of Hadoop 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. We also Flink, as it processes faster than spark because of its streaming architecture and Flink gets instructions 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. The MapReduce framework of Hadoop does not leverage the memory of the [**Hadoop cluster**](https://data-flair.training/blogs/hadoop-cluster/) to the maximum.

**Solution-**

To solve these limitations of Hadoop spark is used that improves the performance, but [**Spark stream processing**](https://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/)is not as 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**](https://data-flair.training/blogs/machine-learning-tutorial/)and graph processing faster.

4. No Real-time Data Processing

Apache Hadoop is for batch processing, which means it takes a huge amount of data in input, process it and produces the result. Although batch processing is very efficient for processing a high volume of data, depending on the size of the data that processes and the computational power of the system, an output can delay 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 processes within a small period of time. Learn more about[**Spark Streaming APIs**](https://data-flair.training/blogs/apache-spark-streaming-transformation-operations/)**.**
* **Apache Flink** provides single run-time for the streaming as well as batch processing, so one common run-time is utilized for data streaming applications and batch processing applications. Flink is a stream processing system that is able to process row after row in real time.

5. No Delta Iteration

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-**

We can use Apache Spark to overcome this type of Limitations of Hadoop, 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, we schedule and execute each iteration separately.

6. Latency

In Hadoop, MapReduce framework is comparatively slower, since it is for supporting 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 elements are broken down into [**key-value pairs**](https://data-flair.training/blogs/key-value-pairs-hadoop-mapreduce/) 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 limitation of Hadoop, Apache Spark is yet another batch system but it is relatively faster since it caches much of the input data on memory by [**RDD(Resilient Distributed Dataset)**](https://data-flair.training/blogs/rdd-in-apache-spark/)and keeps intermediate data in memory itself. Flink’s data streaming achieves low latency and high throughput.FR

7. Not Easy to 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**](https://data-flair.training/blogs/apache-hive-tutorial-introductory-guide/)and pig makes working with MapReduce a little easier for adopters.

**Solution-**

To solve this Drawback of Hadoop, we can use the spark. Spark has interactive mode so that developers and users alike can have intermediate feedback for queries and other activities. Spark is easy to program as it has tons of high-level operators. We can easily use Flink as it also has high-level operators. This way spark can solve many limitations of Hadoop.

8. Security

Hadoop is challenging in managing the complex application. If the user doesn’t know how to enable a platform who is managing the platform, your data can be a 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 **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 a security bonus to overcome these limitations of Hadoop. If we run the spark in HDFS, it can use HDFS ACLs and file-level permissions. Additionally, Spark can run on [**YARN**](https://data-flair.training/blogs/hadoop-yarn-tutorial/) giving it the capability of using Kerberos authentication.

9. No 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 these drawbacks of Hadoop, Spark is used in which we have [**RDD abstraction for the batch.**](https://data-flair.training/blogs/spark-rdd-tutorial/) Flink has Dataset abstraction.

10. Vulnerable by Nature

Hadoop 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. No 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 limitation of Hadoop, as Spark and Flink cache data in memory for further iterations which enhance the overall performance.

12. Lengthy Line of Code

Hadoop has a 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 and java but the implementation is in Scala, so the number of lines of code is lesser than Hadoop. So it will also take less time to execute the program and solve the lengthy line of code limitations of Hadoop.

13. Uncertainty

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