**Hadoop**

1. **What is Apache Hadoop?**

[**Hadoop**](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) emerged as a solution to the “[**Big Data**](http://data-flair.training/blogs/why-learn-big-data-use-cases/)” problems. It is a part of the Apache project sponsored by the *Apache Software Foundation (ASF)*. It is an open source software framework for distributed storage and distributed processing of large data sets. Open source means it is freely available and even we can change its source code as per our requirements. Apache Hadoop makes it possible to run applications on the system with thousands of commodity hardware nodes. It’s distributed file system has the provision of rapid data transfer rates among nodes. It also allows the system to continue operating in case of node failure. Apache Hadoop provides:

**Storage layer**– [**HDFS**](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/)

**Batch processing engine**– [**MapReduce**](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/)

**Resource Management Layer**– [**YARN**](http://data-flair.training/blogs/hadoop-yarn-tutorial/)

1. **Why do we need Hadoop?**

The picture of Hadoop came into existence to deal with Big Data challenges. The challenges with Big Data are-

* **Storage –** Since data is very large, so storing such huge amount of data is very difficult.
* **Security –** Since the data is huge in size, keeping it secure is another challenge.
* **Analytics –** In Big Data, most of the time we are unaware of the kind of data we are dealing with. So analyzing that data is even more difficult.
* **Data Quality –** In the case of Big Data, data is very messy, inconsistent and incomplete.
* **Discovery –** Using a powerful algorithm to find patterns and insights are very difficult.
* Hadoop is an open-source software framework that supports the storage and processing of large data sets. Apache Hadoop is the best solution for storing and processing Big data because:  
  Apache Hadoop stores huge files as they are (raw) without specifying any schema.
* **High scalability –** We can add any number of nodes, hence enhancing performance dramatically.
* **Reliable –** It stores data reliably on the cluster despite machine failure.
* **High availability –** In Hadoop data is highly available despite hardware failure. If a machine or hardware crashes, then we can access data from another path.
* **Economic –**Hadoop runs on a cluster of commodity hardware which is not very expensive

1. **What are the core components of Hadoop?**

**Hadoop** is an open-source software framework for distributed storage and processing of large datasets. Apache Hadoop core components are HDFS, MapReduce, and YARN.

* **HDFS-** Hadoop Distributed File System (HDFS) is the primary storage system of Hadoop. HDFS store very large files running on a cluster of commodity hardware. It works on the principle of storage of less number of large files rather than the huge number of small files. HDFS stores data reliably even in the case of hardware failure. It provides high throughput access to an application by accessing in parallel.
* **MapReduce-** [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) is the data processing layer of Hadoop. It writes an application that processes large structured and unstructured data stored in HDFS. MapReduce processes a huge amount of data in parallel. It does this by dividing the job (submitted job) into a set of independent tasks (sub-job). In Hadoop, MapReduce works by breaking the processing into phases: **Map** and **Reduce.** The Map is the first phase of processing, where we specify all the complex logic code. Reduce is the second phase of processing. Here we specify light-weight processing like aggregation/summation.
* **YARN-** YARN is the processing framework in Hadoop. It provides Resource management and allows multiple data processing engines. For example real-time streaming, data science, and batch processing.

### ****Tell me about the various Hadoop daemons and their roles in a Hadoop cluster.****

Generally approach this question by first explaining the HDFS daemons i.e. NameNode, DataNode and Secondary NameNode, and then moving on to the YARN daemons i.e. ResorceManager and NodeManager, and lastly explaining the JobHistoryServer.

* **NameNode:**It is the master node which is responsible for storing the metadata of all the files and directories. It has information about blocks, that make a file, and where those blocks are located in the cluster.
* **Datanode:**It is the slave node that contains the actual data.
* **Secondary NameNode:**It periodically merges the changes (edit log) with the FsImage (Filesystem Image), present in the NameNode. It stores the modified FsImage into persistent storage, which can be used in case of failure of NameNode.
* **ResourceManager:**It is the central authority that manages resources and schedule applications running on top of YARN.
* **NodeManager:**It runs on slave machines, and is responsible for launching the application’s containers (where applications execute their part), monitoring their resource usage (CPU, memory, disk, network) and reporting these to the ResourceManager.
* **JobHistoryServer:** It maintains information about MapReduce jobs after the Application Master terminates.

### ****Compare HDFS with Network Attached Storage (NAS).****

In this question, first explain NAS and HDFS, and then compare their features as follows:

Network-attached storage (NAS) is a file-level computer data storage server connected to a computer network providing data access to a heterogeneous group of clients. NAS can either be a hardware or software which provides services for storing and accessing files. Whereas Hadoop Distributed File System (HDFS) is a distributed filesystem to store data using commodity hardware.

In HDFS Data Blocks are distributed across all the machines in a cluster. Whereas in NAS data is stored on a dedicated hardware.

HDFS is designed to work with MapReduce paradigm, where computation is moved to the data. NAS is not suitable for MapReduce since data is stored separately from the computations.

HDFS uses commodity hardware which is cost effective, whereas a NAS is a high-end storage devices which includes high cost.

### ****How does NameNode tackle DataNode failures?****

NameNode periodically receives a Heartbeat (signal) from each of the DataNode in the cluster, which implies DataNode is functioning properly.

A block report contains a list of all the blocks on a DataNode. If a DataNode fails to send a heartbeat message, after a specific period of time it is marked dead.

The NameNode replicates the blocks of dead node to another DataNode using the replicas created earlier.

### ****What will you do when NameNode is down?****

The NameNode recovery process involves the following steps to make the Hadoop cluster up and running:

* Use the file system metadata replica (FsImage) to start a new NameNode.
* Then, configure the DataNodes and clients so that they can acknowledge this new NameNode, that is started.
* Now the new NameNode will start serving the client after it has completed loading the last checkpoint FsImage (for metadata information) and received enough block reports from the DataNodes.

1. **What are the Features of Hadoop?**

The various Features of Hadoop are:

* **Open Source –** Apache Hadoop is an open source software framework. Open source means it is freely available and even we can change its source code as per our requirements.
* **Distributed processing –** As *HDFS* stores data in a distributed manner across the cluster. MapReduce process the data in parallel on the cluster of nodes.
* **Fault Tolerance –** Apache Hadoop is highly Fault-Tolerant. By default, each block creates 3 replicas across the cluster and we can change it as per needment. So if any node goes down, we can recover data on that node from the other node. Framework recovers failures of nodes or tasks automatically.
* **Reliability –** It stores data reliably on the cluster despite machine failure.
* **High Availability –** Data is highly available and accessible despite hardware failure. In Hadoop, when a machine or hardware crashes, then we can access data from another path.
* **Scalability –** Hadoop is highly scalable, as one can add the new hardware to the nodes.
* **Economic-** Hadoop runs on a cluster of commodity hardware which is not very expensive. We do not need any specialized machine for it.
* **Easy to use –** No need of client to deal with distributed computing, the framework take care of all the things. So it is easy to use.

1. **Compare Hadoop and RDBMS?**

Apache Hadoop is the future of the database because it stores and processes a large amount of data. Which will not be possible with the traditional database. There is some difference between **Hadoop** and **RDBMS** which are as follows:

* **Architecture –**Traditional RDBMS have**ACID**properties. Whereas Hadoop is distributed computing framework having two main components: Distributed**file system (HDFS)** and **MapReduce.**
* **Data acceptance –**RDBMS accepts only structured data. While Hadoop can accept both structured as well as unstructured data. It is a great feature of hadoop, as we can store everything in our database and there will be no data loss.
* **Scalability –**RDBMS is a traditional database which provides vertical scalability. So if the data increases for storing then we have to increase particular system configuration. While Hadoop provides horizontal scalability. So we just have to add one or more node to the cluster if there is any requirement for an increase in data.
* **OLTP (Real-time data processing) and OLAP –**Traditional RDMS support OLTP (Real-time data processing). OLTP is not supported in Apache Hadoop. Apache Hadoop supports large scale Batch Processing workloads (OLAP).

1. **What are the modes in which Hadoop run?**

Apache Hadoop runs in three modes:

* **Local (Standalone) Mode –**Hadoop by default run in a single-node, non-distributed mode, as a single Java process. Local mode uses the local file system for input and output operation. It is also used for debugging purpose, and it does not support the use of HDFS. Further, in this mode, there is no custom configuration required for configuration files.
* **Pseudo-Distributed Mode –**Just like the Standalone mode, Hadoop also runs on a single-node in a Pseudo-distributed mode. The difference is that each daemon runs in a separate Java process in this Mode. In Pseudo-distributed mode, we need configuration for all the four files mentioned above. In this case, all daemons are running on one node and thus, both Master and Slave node are the same.
* **Fully-Distributed Mode –**In this mode, all daemons execute in separate nodes forming a multi-node cluster. Thus, it allows separate nodes for Master and Slave.ensed software, therefore we have to pay for the software. Whereas Hadoop is open source framework, so we don’t need to pay for software.

1. **What are configuration files in Hadoop?**

* **Core-site.xml –**It contain configuration setting for Hadoop core such as I/O settings that are common to HDFS & MapReduce. It use Hostname and port .The most commonly used port is 9000.
* **hdfs-site.xml –** This file contains the configuration setting for HDFS daemons. hdfs-site.xml also specify default block replication and permission checking on HDFS.
* **mapred-site.xml –**In this file, we specify a framework name for MapReduce. we can specify by setting the mapreduce.framework.name.
* **yarn-site.xml –**This file provide configuration setting for [NodeManager](http://data-flair.training/blogs/hadoop-yarn-node-manager-tutorial-guide/) and [ResourceManager](http://data-flair.training/blogs/hadoop-yarn-resource-manager-guide-tutorial/).

1. **What are the limitations of Hadoop?**

Various limitations of Hadoop are:

* **Issue with small files –** Hadoop is not suited for small files. Small files are the major problems in HDFS. A small file is significantly smaller than the HDFS block size (default 128MB). If you are storing these large number of small files, HDFS can’t handle these lots of files. As HDFS works with a small number of large files for storing data sets rather than larger number of small files. If one use the huge number of small files, then this will overload the namenode. Since namenode stores the namespace of HDFS.  
  HAR files, Sequence files, and Hbase overcome small files issues.
* **Processing Speed –** With parallel and distributed algorithm, MapReduce process large data sets. MapReduce performs the task: ***Map***and ***Reduce***. MapReduce requires a lot of time to perform these tasks thereby increasing latency. As data is distributed and processed over the cluster in MapReduce. So, it will increase the time and reduces processing speed.
* **Support only Batch Processing –** Hadoop supports only batch processing. It does not process streamed data and hence, overall performance is slower. MapReduce framework does not leverage the memory of the cluster to the maximum.
* **Iterative Processing –** Hadoop is not efficient for iterative processing. As hadoop does not support cyclic data flow. That is the chain of stages in which the input to the next stage is the output from the previous stage.
* **Vulnerable by nature –** Hadoop is entirely written in Java, a language most widely used. Hence java been most heavily exploited by cyber-criminal. Therefore it implicates in numerous security breaches.
* **Security-** Hadoop can be challenging in managing the complex application. Hadoop is missing encryption at storage and network levels, which is a major point of concern. Hadoop supports Kerberos authentication, which is hard to manage.

1. **What are the most commonly defined input formats in Hadoop?**

The most common Input Formats defined in Hadoop are:

* Text Input Format- This is the default input format defined in Hadoop.
* Key Value Input Format- This input format is used for plain text files wherein the files are broken down into lines.
* Sequence File Input Format- This input format is used for reading files in sequence.

1. **Compare Hadoop 2 and Hadoop 3?**

In Hadoop 2, minimum supported version of Java is **Java 7**, while in Hadoop 3 is **Java 8**.

Hadoop 2, handle fault tolerance by replication (which is wastage of space). While Hadoop 3 handle it by Erasure coding.

For data balancing Hadoop 2 uses HDFS balancer. While Hadoop 3 uses Intra-datanode balancer.

In Hadoop 2 some default ports are Linux ephemeral port range. So at the time of startup, they will fail to bind. But in Hadoop 3 these ports have been moved out of the ephemeral range.In hadoop 2, HDFS has 200% overhead in storage space. While Hadoop 3 has 50% overhead in storage space.

Hadoop 2 has features to overcome**SPOF** (single point of failure). So whenever NameNode fails, it recovers automatically. Hadoop 3 recovers SPOF automatically no need of manual intervention to overcome it.

1. **Explain Data Locality in Hadoop?**

Hadoop major drawback was cross-switch network traffic due to the huge volume of data. To overcome this drawback, ***Data locality*** came into the picture. It refers to the ability to move the computation close to where the actual data resides on the node, instead of moving large data to computation. Data locality increases the overall throughput of the system.  
In Hadoop, HDFS stores datasets. Datasets are divided into blocks and stored across the datanodes in[Hadoop cluster.](http://data-flair.training/blogs/installation-hadoop-3-x-ubuntu-pseudo-distributed-mode/) When a user runs the MapReduce job then NameNode sends this MapReduce code to the datanodes on which data is available related to MapReduce job.  
**Data locality has three categories:**

* **Data local –** In this category data is on the same node as the mapper working on the data. In such case, the proximity of the data is closer to the computation. This is the most preferred scenario.
* **Intra –** Rack- In this scenarios mapper run on the different node but on the same rack. As it is not always possible to execute the mapper on the same datanode due to constraints.
* **Inter-Rack –** In this scenarios mapper run on the different rack. As it is not possible to execute mapper on a different node in the same rack due to resource constraints.

1. **What is Safemode in Hadoop?**

Safemode in Apache Hadoop is a maintenance state of NameNode. During which NameNode doesn’t allow any modifications to the file system. During Safemode, HDFS cluster is in read-only and doesn’t replicate or delete blocks. At the startup of NameNode:

* It loads the file system namespace from the last saved FsImage into its main memory and the edits log file.
* Merges edits log file on FsImage and results in new file system namespace.
* Then it receives block reports containing information about block location from all datanodes.

In SafeMode NameNode perform a collection of block reports from datanodes. NameNode enters safemode automatically during its start up. NameNode leaves Safemode after the DataNodes have reported that most blocks are available. Use the command:

*hadoop dfsadmin –safemode get: To know the status of Safemode*  
*bin/hadoop dfsadmin –safemode enter: To enter Safemode*  
*hadoop dfsadmin -safemode leave: To come out of Safemode*  
NameNode front page shows whether safemode is on or off.

1. **What is a “Distributed Cache” in Apache Hadoop?**

In Hadoop, data chunks process independently in parallel among DataNodes, using a program written by the user. If we want to access some files from all the DataNodes, then we will put that file to distributed cache.  
MapReduce framework provides **Distributed Cache** to caches files needed by the applications. It can cache read-only text files, archives, jar files etc.  
Once we have cached a file for our job. Then, Hadoop will make it available on each datanodes where map/reduce tasks are running. Then, we can access files from all the datanodes in our map and reduce job.  
An application which needs to use distributed cache should make sure that the files are available on URLs. URLs can be either **hdfs:// or http://.** Now, if the file is present on the mentioned URLs. The user mentions it to be cache file to distributed cache. This framework will copy the cache file on all the nodes before starting of tasks on those nodes. By default size of distributed cache is 10 GB. We can adjust the size of distributed cache using local.cache.size.

1. **1How is security achieved in Hadoop?**

Apache Hadoop achieves security by using **Kerberos**.  
At a high level, there are three steps that a client must take to access a service when using Kerberos. Thus, each of which involves a message exchange with a server.

* **Authentication –**The client authenticates itself to the authentication server. Then, receives a timestamped *Ticket-Granting Ticket (TGT)*.
* **Authorization –**The client uses the TGT to request a service ticket from the Ticket Granting Server.
* **Service Request –**The client uses the service ticket to authenticate itself to the server.

1. **What is throughput in Hadoop?**

The amount of work done in a unit time is Throughput. Because of bellow reasons HDFS provides good throughput:

The HDFS is Write Once and Read Many Model. It simplifies the data coherency issues as the data written once, one can not modify it. Thus, provides high throughput data access.  
Hadoop works on **Data Locality** principle. This principle state that moves computation to data instead of data to computation. This reduces network congestion and therefore, enhances the overall system throughput.

1. **How to restart NameNode or all the daemons in Hadoop?**

By following methods we can restart the NameNode:

You can stop the NameNode individually using ***/*sbin*/hadoop-daemon.sh*** stop namenode command. Then start the NameNode using ***/sbin/hadoop-daemon.sh***start namenode.  
Use ***/sbin/stop-all.sh*** and the use ***/sbin/start-all.sh***, command which will stop all the demons first. Then start all the daemons.  
The sbin directory inside the Hadoop directory stores these script files.

1. **What does jps command do in Hadoop?**

The jbs command helps us to check if the Hadoop daemons are running or not. Thus, it shows all the Hadoop daemons that are running on the machine. Daemons are *Namenode, Datanode, ResourceManager, NodeManager etc*.

1. **What are the main hdfs-site.xml properties?**

**hdfs-site.xml –** This file contains the configuration setting for HDFS daemons. hdfs-site.xml also specify default block replication and permission checking on HDFS.  
The three main hdfs-site.xml properties are:

* ***dfs.name.dir*** gives you the location where NameNode stores the metadata (FsImage and edit logs). And also specify where DFS should locate – on the disk or in the remote directory.
* ***dfs.data.dir*** gives the location of DataNodes where it stores the data.
* ***fs.checkpoint.dir*** is the directory on the file system. On which secondary NameNode stores the temporary images of edit logs. Then this EditLogs and FsImage will merge for backup.

1. **What is fsck?**

**fsck** is the **File System Check**. Hadoop HDFS use the fsck (filesystem check) command to check for various inconsistencies. It also reports the problems with the files in HDFS. For example, missing blocks for a file or under-replicated blocks. It is different from the traditional fsck utility for the native file system. Therefore it does not correct the errors it detects.

Normally NameNode automatically corrects most of the recoverable failures. Filesystem check also ignores open files. But it provides an option to select all files during reporting. The HDFS fsck command is not a Hadoop shell command. It can also run as *bin/hdfs fsck*. Filesystem check can run on the whole file system or on a subset of files.

**Usage:**

hdfs fsck <path>  
[-list-corruptfileblocks |  
[-move | -delete | -openforwrite]  
[-files [-blocks [-locations | -racks]]]  
[-includeSnapshots]  
Path- Start checking from this path  
-delete- Delete corrupted files.  
-files- Print out the checked files.  
-files –blocks- Print out the block report.  
-files –blocks –locations- Print out locations for every block.  
-files –blocks –rack- Print out network topology for data-node locations  
-includeSnapshots- Include snapshot data if the given path indicates or include snapshottable directory.  
-list -corruptfileblocks- Print the list of missing files and blocks they belong to.

1. **How to debug Hadoop code?**

First, check the list of MapReduce jobs currently running. Then, check whether orphaned jobs is running or not; if yes, you need to determine the location of RM logs.

First of all, Run: “**ps –ef| grep –I ResourceManager**” and then, look for log directory in the displayed result. Find out the job-id from the displayed list. Then check whether error message associated with that job or not.  
Now, on the basis of RM logs, identify the worker node which involves in the execution of the task.

Now, login to that node and run- “**ps –ef| grep –I NodeManager**”  
Examine the NodeManager log.  
The majority of errors come from user level logs for each amp-reduce job.

1. **Explain Hadoop streaming?**

Hadoop distribution provides generic *application programming interface (API)*. This allows writing Map and Reduce jobs in any desired programming language. The utility allows creating/running jobs with any executable as Mapper/Reducer.  
For example:

hadoop jar hadoop-streaming-3.0.jar \  
-input myInputDirs \  
-output myOutputDir \  
-mapper /bin/cat \  
-reducer /usr/bin/wc

In the example, both the Mapper and reducer are executables. That read the input from stdin (line by line) and emit the output to stdout. The utility allows creating/submitting Map/Reduce job, to an appropriate cluster. It also monitors the progress of the job until it completes. Hadoop Streaming uses both streaming command options as well as generic command options. Be sure to place the generic options before the streaming. Otherwise, the command will fail.

The general line syntax shown below:

1. **What does hadoop-metrics.properties file do?**

Statistical information exposed by the Hadoop daemons is Metrics. Hadoop framework uses it for monitoring, performance tuning and debug.

By default, there are many metrics available. Thus, they are very useful for troubleshooting.

Hadoop framework use **hadoop-metrics.properties** for ‘**Performance Reporting**’ purpose. It also controls the reporting for Hadoop. The API provides an abstraction so we can implement on top of a variety of metrics client libraries. The choice of client library is a configuration option. And different modules within the same application can use different metrics implementation libraries.  
This file is present inside*/etc/hadoop.*

1. **What are the different commands used to startup and shutdown Hadoop daemons?**

• To start all the hadoop daemons use: ./sbin/start-all.sh.  
Then, to stop all the Hadoop daemons use:./sbin/stop-all.sh  
• You can also start all the dfs daemons together using ./sbin/start-dfs.sh. Yarn daemons together using ./sbin/start-yarn.sh. MR Job history server using /sbin/mr-jobhistory-daemon.sh start history server. Then, to stop these daemons we can use  
./sbin/stop-dfs.sh  
./sbin/stop-yarn.sh  
/sbin/mr-jobhistory-daemon.sh stop historyserver.  
• Finally, the last way is to start all the daemons individually. Then, stop them individually:  
./sbin/hadoop-daemon.sh start namenode  
./sbin/hadoop-daemon.sh start datanode  
./sbin/yarn-daemon.sh start resourcemanager  
./sbin/yarn-daemon.sh start nodemanager  
./sbin/mr-jobhistory-daemon.sh start historyserver

**Big Limitations of Hadoop for Big Data Analytics**

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

### 3.1. Issue with Small Files

**Hadoop** is not suited for small data. [**(HDFS)** **Hadoop distributed file system**](http://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**](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.

### 3.2. Slow 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**](http://data-flair.training/blogs/apache-spark-in-memory-computing/) 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.

### 3.3. 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**](http://data-flair.training/blogs/install-hadoop-2-x-ubuntu-hadoop-multi-node-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**](http://data-flair.training/blogs/machine-learning-tutorial/)and graph processing faster.

### 3.4. No 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. Learn more about [Spark Streaming APIs](http://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 application and batch processing application. Flink is a stream processing system that is able to process row after row in real time.

### 3.5. 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-**

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.

### 3.6. Latency

In Hadoop, 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**](http://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 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(Resilient Distributed Dataset)**](http://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.

### 3.7. 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**](http://data-flair.training/blogs/apache-hive-tutorial-introductory-guide/)and[**pig**](http://data-flair.training/blogs/apache-pig-tutorial-introduction-guide/)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.

### 3.8. 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 **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, 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.

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

### 3.10. 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.

### 3.11. 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, as Spark and Flink cache data in memory for further iterations which enhance the overall performance.

### 3.12. Lengthy 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.

Questions:

1. What is Hadoop framework?
2. Concept on which Hadoop works. Explain?
3. Master Slave Architecture in Hadoop.
4. What is the restriction to the key value class?
5. What mapper does?
6. What is inputSplit and InputFormat?
7. What are the methods in mapper Interface?
8. Most common Input Output formats in Hadoop
9. Record Reader in hadoop