**Big Data Hadoop and Spark Developer**

Hadoop, an open source project from [The Apache Software Foundation](http://www.apache.org/), emerged from the needs of companies such as Google, Yahoo, AOL and Facebook. These companies need to support daily access to huge data sets across distributed servers.

But two factors will make Hadoop necessary for -- and available to -- many companies: a growing number of applications utilizing very large data sets, and the availability of clouds containing hundreds or thousands of distributed processors with a virtually unlimited amount of storage.

Hadoop framework including HDFS, YARN, and MapReduce.

Pig, Hive, and Impala to process and analyze large datasets stored in the HDFS, and use Sqoop and Flume for data ingestion.

functional programming in Spark, implement Spark applications, understand parallel processing in Spark, and use Spark RDD optimization techniques

Spark SQL for creating, transforming, and querying data form.

This course will enable you to:

* Understand the different components of Hadoop ecosystem such as Hadoop 2.7, Yarn, MapReduce, Pig, Hive, Impala, HBase, Sqoop, Flume, and Apache Spark

Develops open source software for reliable, scalable and 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, each of which may be prone to failures.

The project includes these modules:

* **Hadoop Common**: The common utilities that support the other Hadoop modules.
* **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

Ambari™: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.

Avro™: A data serialization system.

Cassandra™: A scalable multi-master database with no single points of failure.

Chukwa™: A data collection system for managing large distributed systems.

HBase™: A scalable, distributed database that supports structured data storage for large tables.

Hive™: A data warehouse infrastructure that provides data summarization and ad hoc querying.

Mahout™: A Scalable machine learning and data mining library.

Pig™: A high-level data-flow language and execution framework for parallel computation.

Spark™: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.

Tez™: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by Hive™, Pig™ and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop™ MapReduce as the underlying execution engine.

ZooKeeper™: A high-performance coordination service for distributed applications.

https://wiki.apache.org/hadoop/PoweredBy

* Understand Hadoop Distributed File System (HDFS) and YARN as well as their architecture, and learn how to work with them for storage and resource management
* Understand MapReduce and its characteristics, and assimilate some advanced MapReduce concepts
* Get an overview of Sqoop and Flume and describe how to ingest data using them
* Create database and tables in Hive and Impala, understand HBase, and use Hive and Impala for partitioning
* Understand different types of file formats, Avro Schema, using Arvo with Hive, and Sqoop and Schema evolution
* Understand Flume, Flume architecture, sources, flume sinks, channels, and flume configurations
* Understand HBase, its architecture, data storage, and working with HBase. You will also understand the difference between HBase and RDBMS
* Gain a working knowledge of Pig and its components
* Do functional programming in Spark
* Understand resilient distribution datasets (RDD) in detail
* Implement and build Spark applications
* Gain an in-depth understanding of parallel processing in Spark and Spark RDD optimization techniques
* Understand the common use-cases of Spark and the various interactive algorithms
* Learn Spark SQL, creating, transforming, and querying Data frames

Imp for the exam

aggregation, projection, filtering, sorting and joins) query using spark

hive, sqoop

**Hardware Failure**

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system’s data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

**Streaming Data Access**

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

**Large Data Sets**

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

**Simple Coherency Model**

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

**“Moving Computation is Cheaper than Moving Data”**

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

**Portability Across Heterogeneous Hardware and Software Platforms**

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

Namenode, data node are 2 pieces of Java code to handle HDFS.

Each data node has one or more blocks for which namespaces is managed by the namenode. Whenever client request any ops on blocks then the request is passed by name node to data node and data node serves the requests. Usually there is one data node per cluster. Name node is the arbitrator and repository of all the HDFS metadata.

Any change to the namespace is handled/recorded by Name-node. Whether it is about creating/deleting files or moving files from one directory to another directory. An application can specify number of replicas of a file.

The large files are broken down into blocks and which are replicated across multiple nodes for the purpose for fault tolerance.

The block size and replication factor are configurable per file. File in HDFS are write-once and strictly have one writer at a time

rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization

replica selection is done keeping in view to minimize read latency and global bandwidth consumption.

Persistence of file system metadata – Editlof, Fsimage and checkpoint

Communication protocols

TCP/IP, datanode protocol

The three common types of failures are NameNode failures, DataNode failures and network partitions.

### Data Disk Failure, Heartbeats and Re-Replication

### Cluster Rebalancing

### Data Integrity – maintain it with the help of checksum at block level

### Metadata Disk Failure

The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the NameNode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of the FsImages and EditLogs to get updated synchronously. This synchronous updating of multiple copies of the FsImage and EditLog may degrade the rate of namespace transactions per second that a NameNode can support. However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive. When a NameNode restarts, it selects the latest consistent FsImage and EditLog to use.

The NameNode machine is a single point of failure for an HDFS cluster. If the NameNode machine fails, manual intervention is necessary. Currently, automatic restart and failover of the NameNode software to another machine is not supported.

### Snapshots

Snapshots support storing a copy of data at a particular instant of time. One usage of the snapshot feature may be to roll back a corrupted HDFS instance to a previously known good point in time. HDFS does not currently support snapshots but will in a future release.

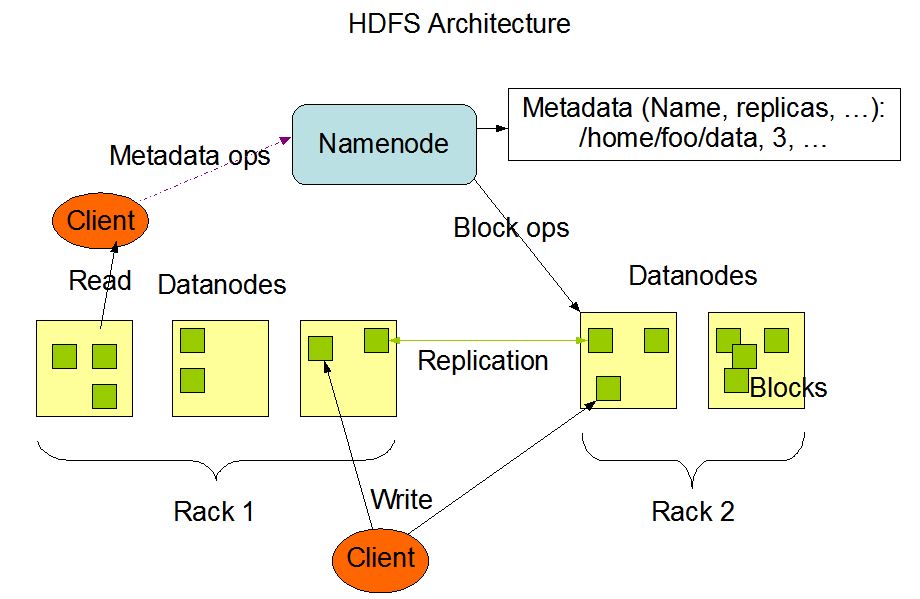
### Data organization

### HDFS files broken into 64MB chunks i.e. blocks and so built for heavy read

### Staging – file buffering till it

### YARN

### Yarn architecture -



Quorum based HA – zookeeper i.e 3 min nodes and these elect 1 of the nodes as master

Doesn’t make any assumptions about the schema of the data

Map reduce model

Mapper – generates key value files

Reducers – does the aggregate or apply function on key value pairs

Note :- You cannot update on Hadoop. Only add/remove . Greenplum allows update the file

Pig is higher level data flow language

Mahout is machine learning stack. Hard to use

Hbase – NoSQL database

Apache Oozie – workflow engine, map reduce is not single but a bunch of steps. Build a whole pipeline of steps and oozie coordinates it.

Sbin contains all the shell scripts. To start Hadoop , we have to run few daemons for eg yarn daemon, name node daemon and datanode daemon. We have to logged in as hduser and same password, mr job history daemon

Job history server manages the jobs running history and it is required by HIVE

Puppet and Ansible are there to deploy Hadoop

Hadoop 1.x has block size

Had a problem where HA and fault tolerance was not present

Now in Hadoop 2.x there is secondary name node as well to enable HA and fault tolerance

Apache Hadoop YARN

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs.

The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based the resource requirements of the applications; it does so based on the abstract notion of a resource *Container* which incorporates elements such as memory, cpu, disk, network etc.

The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the [CapacityScheduler](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html) and the [FairScheduler](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html) would be some examples of plug-ins.

The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.