Hadoop

Dr. Mihail

Content derived from:
Ankam, Venkat. Big Data Analytics. Packt Publishing, 2016.

July 9, 2019

Apache Hadoop

What is it?

- Apache Hadoop is a software framework that enables distributed processing on large clusters with thousands of nodes and petabytes of data.
- Apache Hadoop clusters can be built using commodity hardware where failure rates are generally high.
- Hadoop is designed to handle these failures gracefully without user intervention. Also, Hadoop uses the move computation to the data approach, thereby avoiding significant network I/O.
- Users are able to develop parallel applications quickly, focusing on business logic rather than doing the heavy lifting of distributing data, distributing code for parallel processing, and handling failures.

Parts of Hadoop

Apache Hadoop has four major components:

- Hadoop Common (ecosystem core)
- Hadoop Distributed File System (HDFS)
- Yet Another Resource Manager (YARN)
- MapReduce (framework for parallel processing)

Parts of Hadoop

HDFS

- Used to store data
- Data is distributed across nodes in a cluster
- Can handle node failures

Compute

- Frameworks for processing data in parallel: MapReduce, Crunch, Tez, Pig, Spark, etc.

Cluster Resource Management

- Frameworks for managing and distributing cluster resources: YARN, Slider

Why Hadoop

Benefits

- Economy: Low cost per terabyte processing when compared to commercial solutions. This is because of its open source software and commodity hardware.
- Business: The ability to store and process all the data on a massive scale provides higher business value.
- Technical: The ability to store and process any Variety, Volume, Velocity, and Veracity (all four Vs) of Big Data

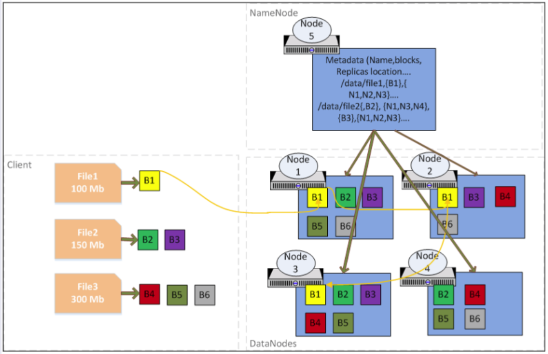
Why Hadoop

Typical characteristics of Hadoop

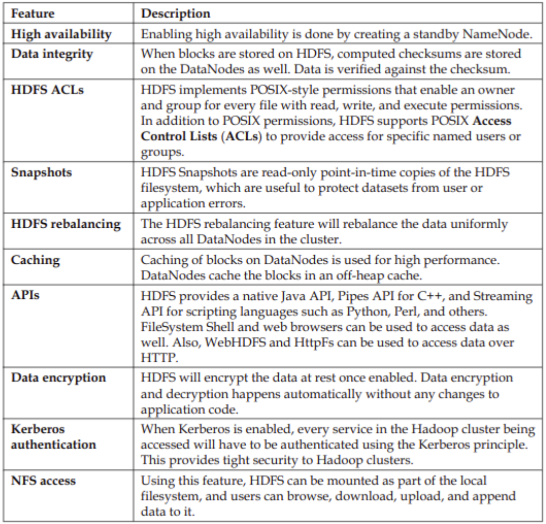
- Commodity: Hadoop can be installed using commodity hardware on-premise or on any cloud provider.
- Robust: It can handle hardware failures at the software layer without user intervention and process failures gracefully without user intervention.
- Scalable: It can commission new nodes to scale out in order to increase the capacity of the cluster.
- Simple: Developers can focus on business logic only, and not on scalability, fault tolerance, and multithreading.
- Data locality: The data size is up to petabytes whereas code size is up to kilobytes. Moving code to the node where data blocks reside provides great reduction in network I/O

HDFS

HDFS files are divided into large blocks that are typically 128 MB in size and distributed across the cluster. Each block is replicated (typically three times) to handle hardware failures and block placement exposed by NameNode so that computation can be moved to data with the MapReduce framework, as illustrated in *Figure 2.1*:

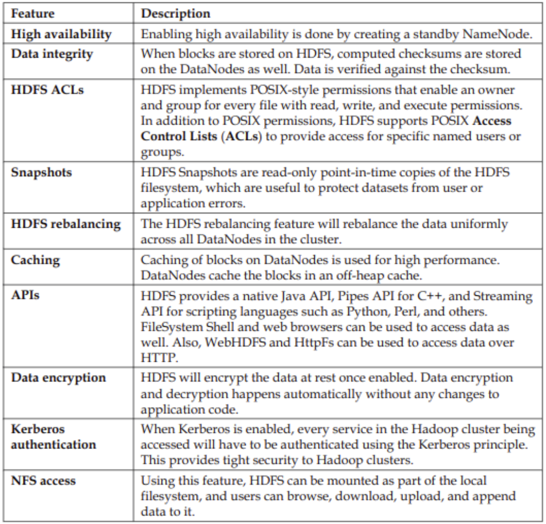

NameNode
Node 5
Metadata (Name,blocks, Replicas location.... /data/file1,{B1},{ N1,N2,N3}.... /data/file2{,B2}, {N1,N3,N4}, {B3},{N1,N2,N3}....
Client
File1 100 Mb
B1
File2 150 Mb
B2
B3
File3 300 Mb
B4
B5
B6
Node 1
Node 2
Node 3
Node 4
DataNodes

HDFS



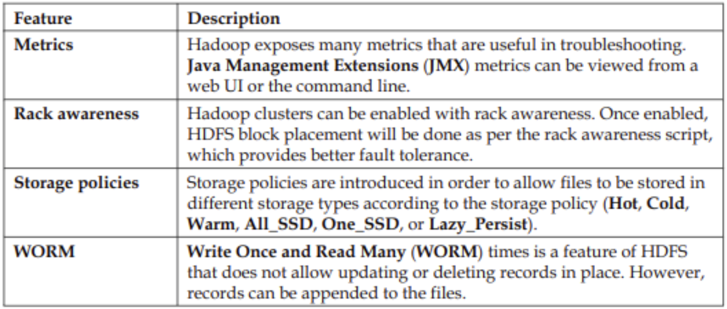
Feature	Description
High availability	Enabling high availability is done by creating a standby NameNode.
Data integrity	When blocks are stored on HDFS, computed checksums are stored on the DataNodes as well. Data is verified against the checksum.
HDFS ACLs	HDFS implements POSIX-style permissions that enable an owner and group for every file with read, write, and execute permissions. In addition to POSIX permissions, HDFS supports POSIX **Access Control Lists** (**ACLs**) to provide access for specific named users or groups.
Snapshots	HDFS Snapshots are read-only point-in-time copies of the HDFS filesystem, which are useful to protect datasets from user or application errors.
HDFS rebalancing	The HDFS rebalancing feature will rebalance the data uniformly across all DataNodes in the cluster.
Caching	Caching of blocks on DataNodes is used for high performance. DataNodes cache the blocks in an off-heap cache.
APIs	HDFS provides a native Java API, Pipes API for C++, and Streaming API for scripting languages such as Python, Perl, and others. FileSystem Shell and web browsers can be used to access data as well. Also, WebHDFS and HttpFs can be used to access data over HTTP.
Data encryption	HDFS will encrypt the data at rest once enabled. Data encryption and decryption happens automatically without any changes to application code.
Kerberos authentication	When Kerberos is enabled, every service in the Hadoop cluster being accessed will have to be authenticated using the Kerberos principle. This provides tight security to Hadoop clusters.
NFS access	Using this feature, HDFS can be mounted as part of the local filesystem, and users can browse, download, upload, and append data to it.

HDFS



Feature	Description
High availability	Enabling high availability is done by creating a standby NameNode.
Data integrity	When blocks are stored on HDFS, computed checksums are stored on the DataNodes as well. Data is verified against the checksum.
HDFS ACLs	HDFS implements POSIX-style permissions that enable an owner and group for every file with read, write, and execute permissions. In addition to POSIX permissions, HDFS supports POSIX **Access Control Lists** (**ACLs**) to provide access for specific named users or groups.
Snapshots	HDFS Snapshots are read-only point-in-time copies of the HDFS filesystem, which are useful to protect datasets from user or application errors.
HDFS rebalancing	The HDFS rebalancing feature will rebalance the data uniformly across all DataNodes in the cluster.
Caching	Caching of blocks on DataNodes is used for high performance. DataNodes cache the blocks in an off-heap cache.
APIs	HDFS provides a native Java API, Pipes API for C++, and Streaming API for scripting languages such as Python, Perl, and others. FileSystem Shell and web browsers can be used to access data as well. Also, WebHDFS and HttpFs can be used to access data over HTTP.
Data encryption	HDFS will encrypt the data at rest once enabled. Data encryption and decryption happens automatically without any changes to application code.
Kerberos authentication	When Kerberos is enabled, every service in the Hadoop cluster being accessed will have to be authenticated using the Kerberos principle. This provides tight security to Hadoop clusters.
NFS access	Using this feature, HDFS can be mounted as part of the local filesystem, and users can browse, download, upload, and append data to it.

HDFS



Feature	Description
Metrics	Hadoop exposes many metrics that are useful in troubleshooting. **Java Management Extensions** (**JMX**) metrics can be viewed from a web UI or the command line.
Rack awareness	Hadoop clusters can be enabled with rack awareness. Once enabled, HDFS block placement will be done as per the rack awareness script, which provides better fault tolerance.
Storage policies	Storage policies are introduced in order to allow files to be stored in different storage types according to the storage policy (**Hot**, **Cold**, **Warm**, **All_SSD**, **One_SSD**, or **Lazy_Persist**).
WORM	**Write Once and Read Many** (**WORM**) times is a feature of HDFS that does not allow updating or deleting records in place. However, records can be appended to the files.

MapReduce

What is it?

Framework to write analytical applications in batch mode on terabytes and petabytes of data.

How does it work?

A MR job typically processes each block of a input file(s) in HDFS with tasks (called maps, or mappers). The MR framework then sorts and shuffles the outputs of the mappers to the reduce tasks in order to produce the output. The framework takes care of:

- Number of tasks needed
- Scheduling of the tasks
- Monitoring
- Re-executing if they fail

MapReduce Features

- Data locality: MR moves the computation to the data. It ships the programs to the nodes where HDFS blocks reside. This reduces the network I/O significantly
- APIs: Native Java, Pipes: C++, Streaming: any shell scripting such as Python
- Distributed cache:A distributed cache is used to cache files such as archives, jars, or any files that are needed by applications at runtime
- Combiner: The combiner feature is used to reduce the network traffic, or, in other words, reduce the amount of data sent from mappers to reducers over the network
- Custom partitioner: This controls which reducer each intermediate key and its associated values go to. A custom partitioner can be used to override the default hash partitioner

MapReduce Features

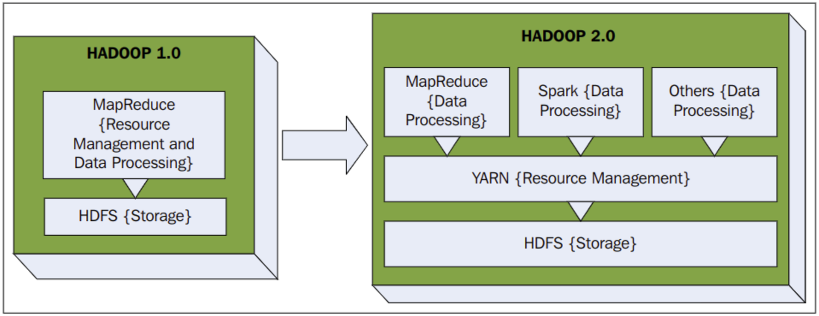
- Sorting:Sorting is done in the sort and shuffle phase, but there are different ways to achieve and control sortingtotal sort, partial sort, and secondary sort
- Joining:Joining two massive datasets with the joining process is easy. If the join is performed by the mapper tasks, it is called a map-side join. If the join is performed by the reducer task, it is called a reduce-side join. Map-side joins are always preferred because it avoids sending a lot of data over the network for reducers to join
- Counters: The MR framework provides built-in counters that give an insight in to how the MR job is performing. It allows the user to define a set of counters in the code, which are then incremented as desired in the mapper or reducer

YARN

What is it?

YARN is the resource management framework that enables an enterprise to process data in multiple ways simultaneously for batch processing, interactive analytics, or real-time analytics on shared datasets. While HDFS provides scalable, fault-tolerant, and cost-efficient storage for Big Data, YARN provides resource management to clusters.

YARN


HADOOP 1.0
MapReduce {Resource Management and Data Processing}
HDFS {Storage}
HADOOP 2.0
MapReduce {Data Processing}
Spark {Data Processing}
Others {Data Processing}
YARN {Resource Management}
HDFS {Storage}

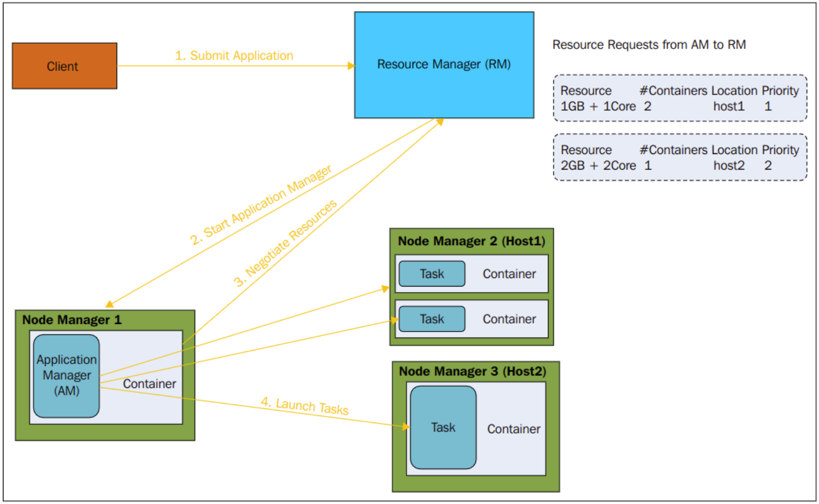
Components of YARN

- **Resource Manager**: keeps track of the resource availability of the entire cluster and provides resources to applications when requested by ApplicationMasteR
- A per-application **AppliationMaster**: negotiates the resources needed by the application to run their tasks. ApplicationMaster also tracks and monitors the progress of the application
- A per-node worker **NodeManager**: is responsible for launching containers provided by ResourceManager, monitoring the resource usage on the slave nodes, and reporting to ResourceManager
- A per-application **container** running on NodeManager: responsible for running the tasks of the application. YARN also has pluggable schedulers (Fair Scheduler and Capacity Scheduler) to control the resource assignments to different applications

YARN Application Lifecycle

- The client submits the MR or Spark job
- The YARN ResourceManager creates an ApplicationMaster on one NodeManager
- The ApplicationMaster negotiates the resources with the ResourceManager
- The ResourceManager provides resources, the NodeManager creates the containers, and the
- ApplicationMaster launches tasks (Map, Reduce, or Spark tasks) in the containers
- Once the tasks are finished, the containers and the ApplicationMaster will be terminated

YARN application lifecycle



Hadoop file storage

Standard or Hadoop container file format

- Standard: structured text (e.g., CSV, TSV, XML and JSON), Unstructured text (e.g., log files and other documents), Unstructured binary data (e.g., Images, Videos and Audio files)
- Hadoop file formats provide splittable compression:
    - File-based structures: Sequence file
    - Serialization format: Thrift, Protocol buffers, Avro
    - Columnar formats: RCFile, ORCFile, Parquet

Sequence file

What?

- Stores data as key-value pairs.
- Supports splitting of files even when data is compressed

Why?

- Small file problem: On an average, each file occupies 600 bytes of space in memory. One million files of 100 KB need 572 MB of main memory on the NameNode. Additionally, the MR job will create one million mappers. Getting Started with Apache Hadoop and Apache Spark
- Solution: Create a sequence file with the key as the filename and value as the content of the file, as shown in the following table. Only 600 bytes of memoryspace is needed in NameNode and an MR job will create 762 mappers with 128 MB block size

Avro

What?

- Row-based data serialization system used for storage and sends data over network efficiently.

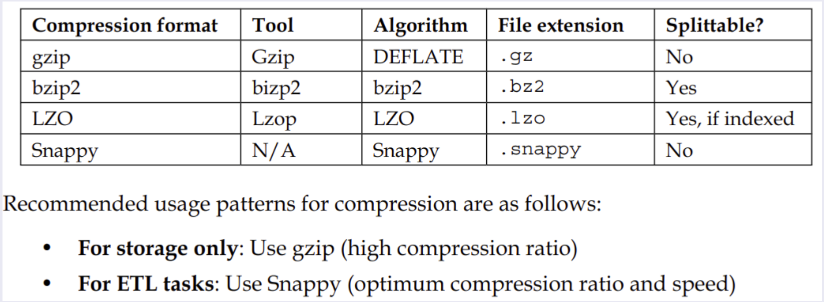
Why?

- Rich data structures
- Compact and fast binary data format
- Simple integration with any language
- Support for evolving schemas
- Great interoperability between Hive, Tez, Impala, Pig and Spark

Compression formats

Why

- Question: if hadoop storage is cheap, why bother with compression?
- Answer: speed up I/O ops, save storage space, speed up transfers over the network
- Compression increases CPU time so trade-offs must be understood well



Compression format	Tool	Algorithm	File extension	Splittable?
gzip	Gzip	DEFLATE	.gz	No
bzip2	bizp2	bzip2	.bz2	Yes
LZO	Lzop	LZO	.lzo	Yes, if indexed
Snappy	N/A	Snappy	.snappy	No

Recommended usage patterns for compression are as follows:

- **For storage only**: Use gzip (high compression ratio)
- **For ETL tasks**: Use Snappy (optimum compression ratio and speed)