By Anne Pernille Wulff Wold and Mari Elida Tuhus

HDFS

Hadoop Distributed File System

HDFS is a file system designed by Apache Hadoop to store large files reliably on affordable commodity hardware. It supports the write-once-read-many-times pattern, which is believed to be the most efficient data processing pattern. A dataset is typically copied from source, and then various analyses are performed on that dataset over time.
Hadoop file system is a highly scalable file system and can easily be scaled by adding more community servers and disks.
The file system supports parallel reading and processing of data, and the system is optimized for streaming read/writes of large files. The system is fault tolerant and easy managementable, because of the system built in redundancy, which means that the system tolerates disk and node failures because of the replication of data.

Architecture:

HDFS' architecture is based on the principle of master-slave, which means that the *cluster* has one instance that controls processes and propagates instructions to worker nodes which execute them. Namenode is the master, datanodes are the worker nodes and a client communicates with master node and reads and writes data in the data nodes. Data is splitted into blocks which are stored within each datanode.

Namenode:

As mentioned the HDFS cluster consists of a single NameNode. The Namenode is the administrator of the HDFS cluster and its main tasks are the distribution of files, i.e. the load balancing, and keeping track of the localization of these files in the cluster.

Namespace:

The Namespace keeps all the metadata, the information about the files in the cluster (blocks that contain content, replication factor, ownership, client access etc) and where they are stored. The namespace image is the state of the namespace at one point, and the namespace image is stored in RAM and persistently on disk. Between two images, an edit log records all the changes made to the cluster. During restart, the image and edit log are merged and stored as a new namespace image and an empty edit log.

Client:

The client in the Hadoop file system refers to the interface used to communicate with the file system. There are different types of clients available to perform different tasks.

Block:

A file is split into one or more blocks. These blocks are typically really large, and usually has the default size of 64 MB or 128 MB.

Datanode:

In the HDFS architecture the datanodes store the actual data, and the node is also known as the slave node. The data nodes are responsible for serving all the read and write request for the clients.

Datanodes send information to the NameNode about the files and blocks stored in that node and respond to the NameNode for all filesystem operations.

Rack:
A rack is a collection of 30 or 40 nodes that are physically stored close together and are all connected to the same network switch. The network bandwidth between any two nodes in the same rack is greater than the bandwidth between two nodes on different racks.

Switcher:
The communication between the different nodes in the system is directed via switch.

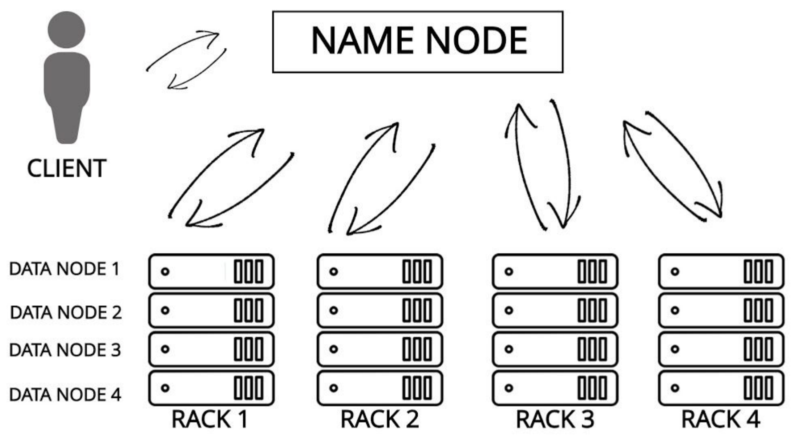

Figure 1: The figure illustrates how the clients communicates with the name node, and how the client file is stored in different data nodes that are located in different racks.

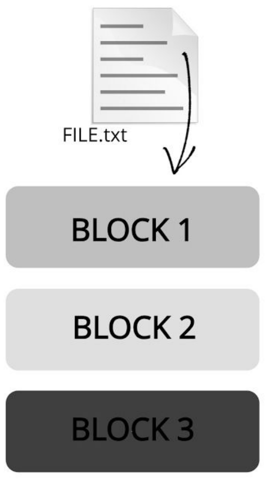

Figure 2: The figure illustrates how a client's file is stored into different blocks according to the size of the file. The blocks are later replicated.

By Anne Pernille Wulff Wold and Mari Elida Tuhus

Replication of data and pipelining:

As mentioned, the data written by a client into an HDFS file is split into one or more blocks. These blocks are later replicated to provide fault tolerance. The default replication factor is 3, but this factor is again configurable. The placement of the replicas is critical for ensuring high reliability and performance of HDFS. When a client is writing data to an HDFS file, the data is first written to an local file. (Suppose the HDFS file has a replication factor of three.)

When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the Namenode. This list contains the Datanodes that will host the replica of that block. The client then flushes the data block to the first Datanode, and the first datanode starts receiving the data in small portions. The datanode writes each portion of data to its local repository and transfers the portion to the second DataNode in the list. The second DataNode, starts receiving data from the first Datanode, saving the data to its local repository and transfer the same portion of data to the third DataNode in the list. This way for transferring and receiving data is called pipelining, and ensures that Data Nodes can receive and transfer data at the same time.

When replicating and storing data on the different DataNodes, the NameNode also ensures that all the replicas are not stored on the same rack. It follows an in-built "Rack Awareness Algorithm" to reduce latency as well as provide fault tolerance. Considering the replication factor to be 3, the Rack Awareness Algorithms says that the first replica on the block will be stored on a local rack, and the next two replicas will be stored on a different rack, but on a different DataNode within the same rack. Hence, no more than one replica is placed on one data node. And no more than two replicas are placed on the same rack. The namenode assigns the 2nd and 3rd replica of a block to nodes in a different rack from the first replica. This provides data protection even against rack failure.

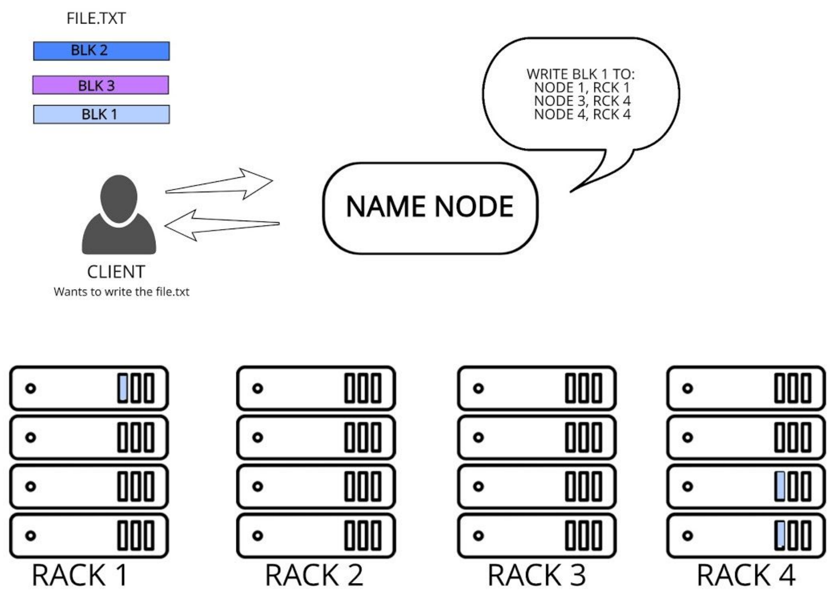
FILE.TXT
BLK 2
BLK 3
BLK 1
WRITE BLK 1 TO: NODE 1, RCK 1 NODE 3, RCK 4 NODE 4, RCK 4
NAME NODE
CLIENT
Wants to write the file.txt
RACK 1
RACK 2
RACK 3
RACK 4

Figure 3: The figure illustrates how the clients file is stored into blocks. The name node sends out instructions on where to store the block. As seen in the figure the light blue block is later stored into different data nodes. The transferring and receiving of data between the data nodes is called pipelining.

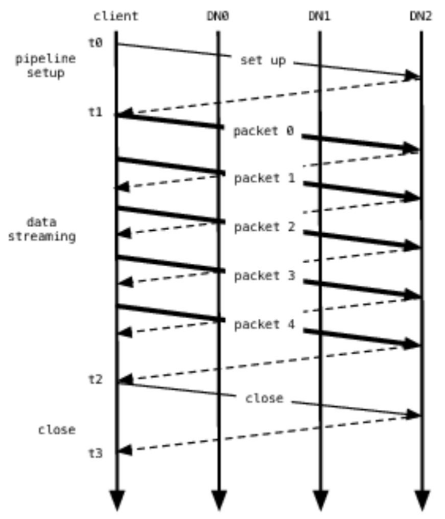

Figure 4: The figure illustrates pipelining. The block is flushed to the first datanode the namenode assigns, and the first datanode starts receiving the data in small portions. The first datanode then transfers the data to the second, which does the same to the third data node. Pipelining allows to transfer and receive data at the same time.

The communication between the different nodes on different racks is directed via switch, as mentioned. In general, you will have greater network bandwidth between machines, data nodes, in the same rack, than machines residing in different racks. So, the Rack Awareness Algorithm helps reduce the write traffic between different racks, and thus provide a better performance.

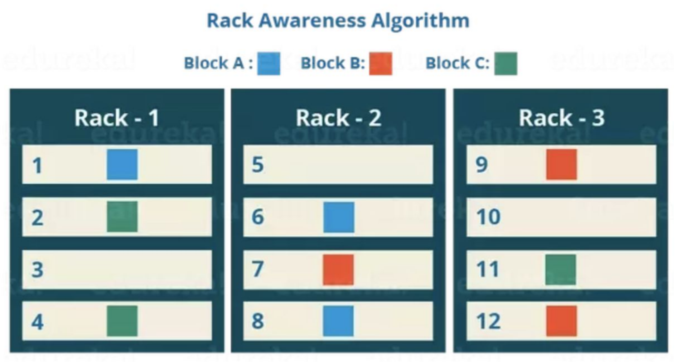

Figure 5: Figure 5 illustrates how the Rack Awareness Algorithm works. The algorithm stores the data first on a local rack, and the two next replicas will be stored in different data nodes, but in the same rack.

Fault tolerance:
The primary objective of HDFS is to store data reliably even in the presence of failures. Since the data is stored on commodity hardware, there will be failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Node: A data node sends a frequent heartbeat to the Name Node to let the Name Node it is alive and working. The Name node interprets absence of heartbeats over time as a DataNode death. This means that no IO operations will be allocated to that node, and the content of the Data Node is scheduled to be copied from one of the other replicas to another Data Node.
In general, a single Data Node fault is not critical in a hdfs cluster due to block replication. Whenever a Data Node fails, the blocks stored on this node is already on two other nodes, and the fault will quickly discovered and replication of the content scheduled.

Network Partition:A network partition can cause a subset of nodes to lose connection with the Name Node and prohibit their *heartbeat* signal from arriving at the Name Node.

Name Node: The Name Node is a single point failure, which means that there needs to be manual intervention if it fails. Because the Name Node is vital to the cluster, there are many ways to keep the consistency of the cluster.

As we already mentioned, there is a persistently saved checkpoint in addition to the namespace image and edit log in RAM at all times.

The Name Node can also run a Secondary Namenode, Checkpoint or Backup node at the same time as it serves clients. Their job is to periodically merge current namespace image and edit log in RAM and return to Name Node so that it can be stored on disk to update the checkpoint more frequently than on restart. The backup Node registers all changes in the edit log and sometimes referred to as a 'Read only Namenode'. The Checkpoint/Backup nodes are run in a different host than Namenode because they have same memory requirements.

In Hadoop 2.0, HDFS High Availability was launched. The concept of HDFS HS is to launch two redundant NameNodes, an active and a stand-by node. The active node performs the "normal" name node tasks while the standby node keeps its state synchronized so that it is ready to take over for the active Namenode in case of failure. Synchronization is obtained through JournalNodes. The standby node continuously surveys the JournalNodes and copies changes applied by the active node. In order to maintain an updated image of block locations, the Data Nodes send block location information and heartbeats to both the Name Nodes.

Other
Balancer: As mentioned previously, there are several competing considerations when blocks are placed in the cluster, and this can result in uneven load placement throughout the cluster. This also happens when a new node is added to the cluster. The Balancer assures a uniform distribution across the nodes in the cluster. It takes a threshold value and tries to keep the difference between a single nodes utilization and the whole cluster's below the threshold value. The balancer does not change the number of replicas or racks when it moves blocks.

Block scanner
Each DataNode periodically scans the block replicas to verify that previously stored checksums (previous log) are identical. The NameNode is notified when the Block Scanner finds a corrupt block. Then, the NameNode starts replicating good replicas and finally, when the replica count of good replicas is at the replication factor, the corrupt block is scheduled for deletion.

Decommissioning
The cluster administrator keeps a list of possible cluster nodes by listing host addresses (and a black-list for those who are not). If a member is changed from included to excluded, it is marked for decommissioning. The block can still serve read request while the Namenode starts replicating the content of the node. When all the content is replicated on other nodes, the decommissioned node can be safely removed.

When to use Hadoop:

1. Large files. The file system easily handles large files, and actually prefers larger files to smaller due to shorter access time.
2. Low cost per byte. HDFS is run on commodity servers, thus the cost per byte is far lower than most comparable competitors
3. High bandwidth. HDFS meets the Big Data requirement of data delivery at a huge data rate
4. Data reliability. The system is built bottom-up to avoid crashes. It has proved its reliability in numerous use cases in clusters of various sizes
5. Scalability. HDFS clusters consist of thousands of nodes, and it is the Name Node memory that limits the scalability. By increasing file sizes, the cluster can be further scaled

When to not use Hadoop:

1. Real time analytics: If quick processing of data is required the use of Hadoop will be disadvantageous. It is because Hadoop is designed for batch processing and hence the response time for complete analysis will be high.
2. Multiple smaller datasets and processing smaller data sets
   Hadoop platform is not recommended for small-structured data sets or processing smaller sets.
3. Hadoop WORM (Write Once Read Many Times)
   If the files are updated frequently, Hadoop is not the right choice, as it is a append-only system.
   The system assumes that a file in HDFS once written will not be modified, though it can be accessed “n” number of times.