An introduction to the Hadoop Distributed File System

Explore HDFS framework and subsystems

J. Jeffrey Hanson

February 01, 2011

The Hadoop Distributed File System (HDFS)—a subproject of the Apache Hadoop project—is a distributed, highly fault-tolerant file system designed to run on low-cost commodity hardware. HDFS provides high-throughput access to application data and is suitable for applications with large data sets. This article explores the primary features of HDFS and provides a high-level view of the HDFS architecture.

HDFS is an Apache Software Foundation project and a subproject of the Apache Hadoop project (see Related topics). Hadoop is ideal for storing large amounts of data, like terabytes and petabytes, and uses HDFS as its storage system. HDFS lets you connect *nodes* (commodity personal computers) contained within clusters over which data files are distributed. You can then access and store the data files as one seamless file system. Access to data files is handled in a *streaming* manner, meaning that applications or commands are executed directly using the MapReduce processing model (again, see Related topics).

HDFS is fault tolerant and provides high-throughput access to large data sets. This article explores the primary features of HDFS and provides a high-level view of the HDFS architecture.

Overview of HDFS

HDFS has many similarities with other distributed file systems, but is different in several respects. One noticeable difference is HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access.

Another unique attribute of HDFS is the viewpoint that it is usually better to locate processing logic near the data rather than moving the data to the application space.

HDFS rigorously restricts data writing to one writer at a time. Bytes are always appended to the end of a stream, and byte streams are guaranteed to be stored in the order written.

HDFS has many goals. Here are some of the most notable:

- Fault tolerance by detecting faults and applying quick, automatic recovery

- Data access via MapReduce streaming
- Simple and robust coherency model
- Processing logic close to the data, rather than the data close to the processing logic
- Portability across heterogeneous commodity hardware and operating systems
- Scalability to reliably store and process large amounts of data
- Economy by distributing data and processing across clusters of commodity personal computers
- Efficiency by distributing data and logic to process it in parallel on nodes where data is located
- Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures

HDFS provides interfaces for applications to move them closer to where the data is located, as described in the following section.

Application interfaces into HDFS

You can access HDFS in many different ways. HDFS provides a native Java™ application programming interface (API) and a native C-language wrapper for the Java API. In addition, you can use a web browser to browse HDFS files.

The applications described in Table 1 are also available to interface with HDFS.

Table 1. Applications that can interface with HDFS

Application	Description
FileSystem (FS) shell	A command-line interface similar to common Linux® and UNIX® shells (bash, csh, etc.) that allows interaction with HDFS data.
DFSAdmin	A command set that you can use to administer an HDFS cluster.
`fsck`	A subcommand of the Hadoop command/application. You can use the `fsck` command to check for inconsistencies with files, such as missing blocks, but you cannot use the `fsck` command to correct these inconsistencies.
Name nodes and data nodes	These have built-in web servers that let administrators check the current status of a cluster.

HDFS has an extraordinary feature set with high expectations thanks to its simple, yet powerful, architecture.

HDFS architecture

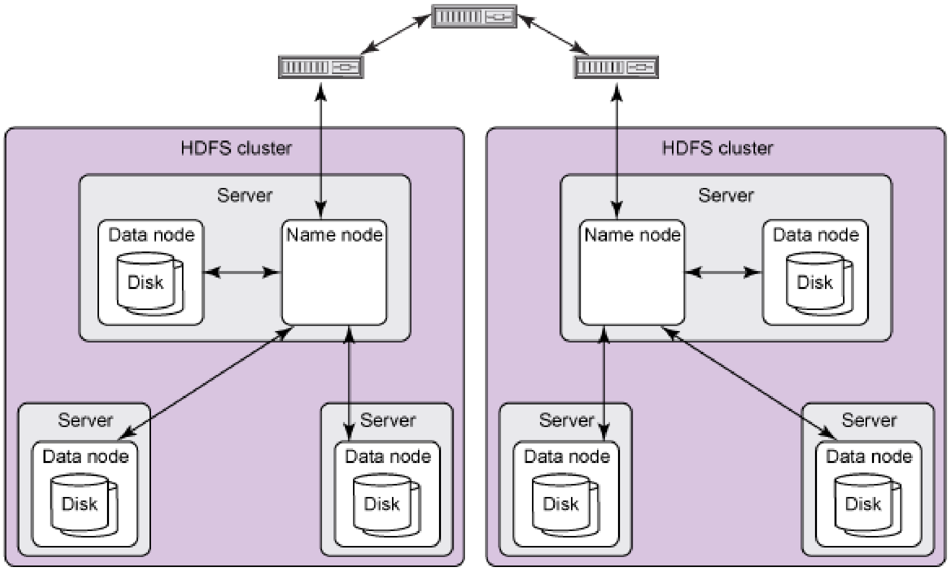
HDFS is comprised of interconnected clusters of nodes where files and directories reside. An HDFS cluster consists of a single node, known as a `NameNode`, that manages the file system namespace and regulates client access to files. In addition, data nodes (`DataNodes`) store data as blocks within files.

Name nodes and data nodes

Within HDFS, a given name node manages file system namespace operations like opening, closing, and renaming files and directories. A name node also maps data blocks to data nodes, which handle read and write requests from HDFS clients. Data nodes also create, delete, and replicate data blocks according to instructions from the governing name node.

Figure 1 illustrates the high-level architecture of HDFS.

Figure 1. The HDFS architecture



As Figure 1 illustrates, each cluster contains one name node. This design facilitates a simplified model for managing each namespace and arbitrating data distribution.

Relationships between name nodes and data nodes

Name nodes and data nodes are software components designed to run in a decoupled manner on commodity machines across heterogeneous operating systems. HDFS is built using the Java programming language; therefore, any machine that supports the Java programming language can run HDFS. A typical installation cluster has a dedicated machine that runs a name node and possibly one data node. Each of the other machines in the cluster runs one data node.

Communications protocols

All HDFS communication protocols build on the TCP/IP protocol. HDFS clients connect to a Transmission Control Protocol (TCP) port opened on the name node, and then communicate with the name node using a proprietary Remote Procedure Call (RPC)-based protocol. Data nodes talk to the name node using a proprietary block-based protocol.

Data nodes continuously loop, asking the name node for instructions. A name node can't connect directly to a data node; it simply returns values from functions invoked by a data node. Each data node maintains an open server socket so that client code or other data nodes can read or write data. The host or port for this server socket is known by the name node, which provides the information to interested clients or other data nodes. See the Communications protocols sidebar for more about communication between data nodes, name nodes, and clients.

The name node maintains and administers changes to the file system namespace.

File system namespace

HDFS supports a traditional hierarchical file organization in which a user or an application can create directories and store files inside them. The file system namespace hierarchy is similar to most other existing file systems; you can create, rename, relocate, and remove files.

HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service (S3) (see Related topics).

Data replication

HDFS replicates file blocks for fault tolerance. An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that. The name node makes all decisions concerning block replication.

Rack awareness

Typically, large HDFS clusters are arranged across multiple installations (racks). Network traffic between different nodes within the same installation is more efficient than network traffic across installations. A name node tries to place replicas of a block on multiple installations for improved fault tolerance. However, HDFS allows administrators to decide on which installation a node belongs. Therefore, each node knows its rack ID, making it *rack aware*.

HDFS uses an intelligent replica placement model for reliability and performance. Optimizing replica placement makes HDFS unique from most other distributed file systems, and is facilitated by a rack-aware replica placement policy that uses network bandwidth efficiently.

Large HDFS environments typically operate across multiple installations of computers. Communication between two data nodes in different installations is typically slower than data nodes within the same installation. Therefore, the name node attempts to optimize communications between data nodes. The name node identifies the location of data nodes by their rack IDs.

Data organization

One of the main goals of HDFS is to support large files. The size of a typical HDFS block is 64MB. Therefore, each HDFS file consists of one or more 64MB blocks. HDFS tries to place each block on separate data nodes.

File creation process

Manipulating files on HDFS is similar to the processes used with other file systems. However, because HDFS is a multi-machine system that appears as a single disk, all code that manipulates files on HDFS uses a subclass of the `org.apache.hadoop.fs.FileSystem` object (see Related topics).

The code shown in Listing 1 illustrates a typical file creation process on HDFS.

Listing 1. Typical file creation process on HDFS

```
byte[] fileData = retrieveFileDataFromSomewhere();
String filePath = retrieveFilePathStringFromSomewhere();
Configuration config = new Configuration();  // assumes to automatically load
                                             // hadoop-default.xml and hadoop-site.xml
org.apache.hadoop.fs.FileSystem hdfs = org.apache.hadoop.fs.FileSystem.get(config);
org.apache.hadoop.fs.Path path = new org.apache.hadoop.fs.Path(filePath);
org.apache.hadoop.fs.FSDataOutputStream outputStream = hdfs.create(path);
outputStream.write(fileData, 0, fileData.length);
```

Staging to commit

When a client creates a file in HDFS, it first caches the data into a temporary local file. It then redirects subsequent writes to the temporary file. When the temporary file accumulates enough data to fill an HDFS block, the client reports this to the name node, which converts the file to a permanent data node. The client then closes the temporary file and flushes any remaining data to the newly created data node. The name node then commits the data node to disk.

Replication pipelining

When a client accumulates a full block of user data, it retrieves a list of data nodes that contains a replica of that block from the name node. The client then flushes the full data block to the first data node specified in the replica list. As the node receives chunks of data, it writes them to disk and transfers copies to the next data node in the list. The next data node does the same. This *pipelining* process is repeated until the replication factor is satisfied.

Data storage reliability

One important objective of HDFS is to store data reliably, even when failures occur within name nodes, data nodes, or network partitions.

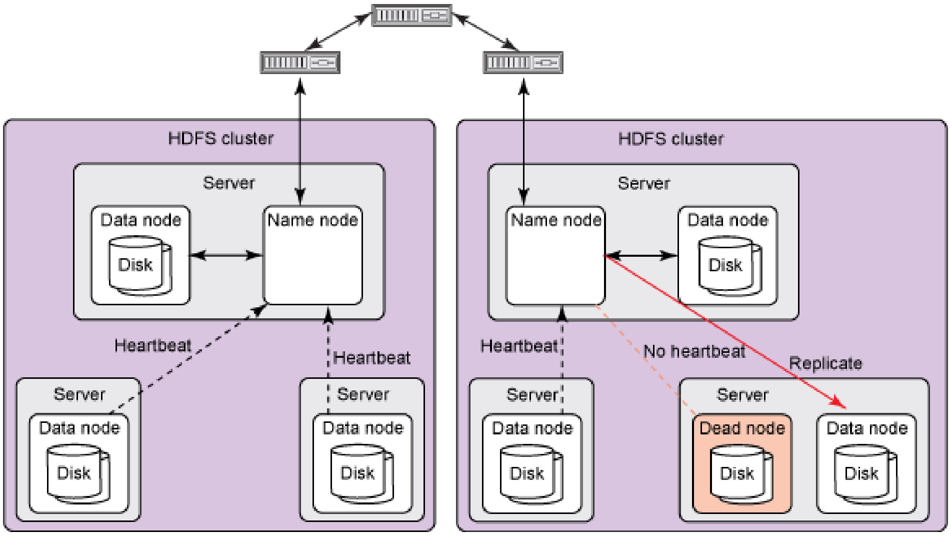
Detection is the first step HDFS takes to overcome failures. HDFS uses heartbeat messages to detect connectivity between name and data nodes.

HDFS heartbeats

Several things can cause loss of connectivity between name and data nodes. Therefore, each data node sends periodic heartbeat messages to its name node, so the latter can detect loss of connectivity if it stops receiving them. The name node marks as dead data nodes not responding to heartbeats and refrains from sending further requests to them. Data stored on a dead node is no longer available to an HDFS client from that node, which is effectively removed from the system. If the death of a node causes the replication factor of data blocks to drop below their minimum value, the name node initiates additional replication to bring the replication factor back to a normalized state.

Figure 2 illustrates the HDFS process of sending heartbeat messages.

Figure 2. The HDFS heartbeat process



Data block rebalancing

HDFS data blocks might not always be placed uniformly across data nodes, meaning that the used space for one or more data nodes can be underutilized. Therefore, HDFS supports rebalancing data blocks using various models. One model might move data blocks from one data node to another automatically if the free space on a data node falls too low. Another model might dynamically create additional replicas and rebalance other data blocks in a cluster if a sudden increase in demand for a given file occurs. HDFS also provides the `hadoop balance` command for manual rebalancing tasks.

One common reason to rebalance is the addition of new data nodes to a cluster. When placing new blocks, name nodes consider various parameters before choosing the data nodes to receive them. Some of the considerations are:

- Block-replica writing policies
- Prevention of data loss due to installation or rack failure
- Reduction of cross-installation network I/O
- Uniform data spread across data nodes in a cluster

The cluster-rebalancing feature of HDFS is just one mechanism it uses to sustain the integrity of its data. Other mechanisms are discussed next.

Data integrity

HDFS goes to great lengths to ensure the integrity of data across clusters. It uses checksum validation on the contents of HDFS files by storing computed checksums in separate, hidden files in the same namespace as the actual data. When a client retrieves file data, it can verify that the data received matches the checksum stored in the associated file.

The HDFS namespace is stored using a transaction log kept by each name node. The file system namespace, along with file block mappings and file system properties, is stored in a file called

FsImage. When a name node is initialized, it reads the FsImage file along with other files, and applies the transactions and state information found in these files.

Synchronous metadata updating

A name node uses a log file known as the EditLog to persistently record every transaction that occurs to HDFS file system metadata. If the EditLog or FsImage files become corrupted, the HDFS instance to which they belong ceases to function. Therefore, a name node supports multiple copies of the FsImage and EditLog files. With multiple copies of these files in place, any change to either file propagates synchronously to all of the copies. When a name node restarts, it uses the latest consistent version of FsImage and EditLog to initialize itself.

HDFS permissions for users, files, and directories

HDFS implements a permissions model for files and directories that has a lot in common with the Portable Operating System Interface (POSIX) model; for example, every file and directory is associated with an owner and a group. The HDFS permissions model supports read (r), write (w), and execute (x). Because there is no concept of file execution within HDFS, the x permission takes on a different meaning. Simply put, the x attribute indicates permission for accessing a child directory of a given parent directory. The owner of a file or directory is the identity of the client process that created it. The group is the group of the parent directory.

Snapshots

HDFS was originally planned to support snapshots that can be used to roll back a corrupted HDFS instance to a previous state. However, HDFS support for snapshots has been tabled for the time being.

Summary

Hadoop is an Apache Software Foundation distributed file system and data management project with goals for storing and managing large amounts of data. Hadoop uses a storage system called HDFS to connect commodity personal computers, known as nodes, contained within clusters over which data blocks are distributed. You can access and store the data blocks as one seamless file system using the MapReduce processing model.

HDFS shares many common features with other distributed file systems while supporting some important differences. One significant difference is HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access.

In order to provide an optimized data-access model, HDFS is designed to locate processing logic near the data rather than locating data near the application space.

Related topics

- The Hadoop wiki provides community input related to Hadoop and HDFS.
- Find free courses on Hadoop fundamentals, stream computing, text analytics, and more at Big Data University.
- Download IBM InfoSphere Streams and build applications that rapidly ingest, analyze, and correlate information as it arrives from thousands of real-time sources.
- Wikipedia's MapReduce page is a great place to begin your research into the MapReduce framework.
- The Hadoop project site contains valuable resources pertaining to the Hadoop architecture and the MapReduce framework.
- Venture to the CloudStore site for downloads and documentation about the integration between CloudStore, Hadoop, and HDFS.