Hadoop Distributed Filesystem

Overview The Hadoop distributed filesystem (HDFS) was designed with the following criteria in mind:

- **Very large files** Files are hundreds of MB, GB, or TB in size. Some Hadoop clusters are running over PB of data.
- **Streaming data access** HDFS is built around the idea that the most efficient data processing pattern is a *write-once read-many-times* pattern. A data is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion of the dataset, so *turnaround time* is more important than the *latency* in reading the first record.
- **Moving computation is cheaper than moving data** A computatino 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 dataset is huge.
- **Fault tolerance** Hadoop runs on *unreliable* commodity hardware. So, fault tolerance is of utmost importance.

Hadoop blocks File system blocks are typically a few KB in size, while disk blocks are normally 512 bytes. The HDFS block size is **64 MB** by default. This minimizes the cost of seeks compared to actual data transfer time. Also, it allows a file to be larger than any signle physical disk and *fixed size* of blocks simplifies filesystem management.

The Hadoop is a unit of *replication* and typically three copies of the same block is replicated in different machines.

Namenodes and datanodes An HDFS cluster has two types of node operating in a msater-worker pattern: a *namenode* (the master) and a number of *datanodes* (workers).

The **namenode** manages the filesystem *namespace*. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored *persistently* on the local disk in the form of two files: `FsImage` and `EditLog`. The namenode is responsible for mapping a given file to a set of datanodes which contain the blocks of the file. Actual block locations are updated on system startup through inquiring datanodes.

The **datanode** store and retrieve blocks when they are told to (by clients or the namenode). These nodes report back to the namenode periodically with lists of blocks that they are storing.

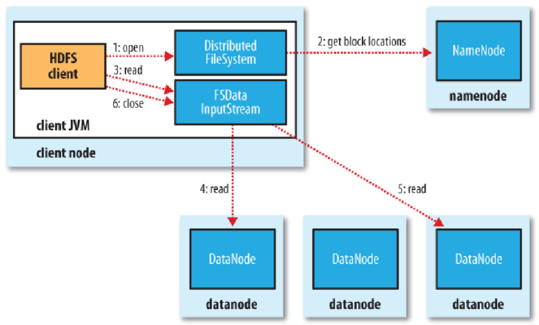
Resilience of namenodes Since the failure in the namenode corrupts the whole file system, it is important to make the namenode resilient. For this, two methods are used. The first way is to *back up* the files that make up the persistent state of the filesystem metadata. Any change in metadata is stored (sometimes to multiple copies) in an atomic and synchronous way.

Also, it's possible to run a *secondary namenode*, which periodically merge the `FsImage` and `EditLog` to prevent the `EditLog` from becoming too large.

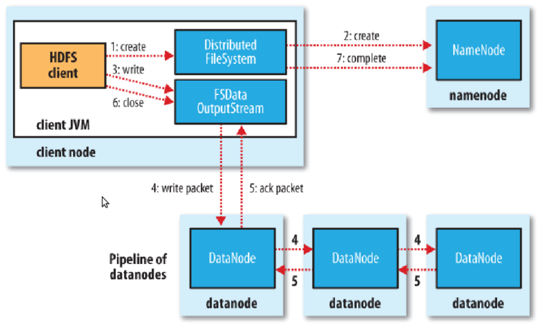
Hadoop filesystem types An HDFS implements the Java abstract class `org.apache.hadoop.fs.FileSystem`. There are several concrete implementations:

- **Local file system**: a local filesystem for the client
- **HDFS**: Hadoop distributed filesystem to be worked with MapReduce
- **HAR**: A filesystem layered on another filesystem for archiving files
- **FTP**: a file system backed by FTP server
- **S3 (native or block-based)**: Amazon S3 filesystem
- **HFTP**, **HSFTP**, **FTP**,**KFS**

Anatomy of HDFS file read



Anatomy of HDFS file write



Persistence of file system metadata The HDFS namespace is stored by the namenode. The namenode uses a *transaction log* called the `EditLog` to persistently record every change that occurs to the filesystem metadata. For example, creating a new file in HDFS causes the namenode to insert a record into the EditLog indicating this.

The namenode keeps an image of the entire filesystem namespace and file `Blockmap` *in memory*. When the namenode starts up, it reads the `FsImage` and `EditLog` from disk, applies all the transactions from the `EditLog` to the in-memory representation of the `FsImage`, and flushes out this new version into a new `FsImage` on disk.

Checkpointing is a process which truncates the old `EditLog` because its transactions have been applied to the persistent `FsImage`.

Communication protocols All HDFS communication is layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the namenode machine. It taskes the **ClientProtocol** with the namenode. The datanodes talk to the namenode using the **DataNodeProtocol**.