DFS requirements

The Hadoop Distributed File System requirements.

The Hadoop Distributed File system (DFS) is a fault tolerant scalable distributed storage component of the Hadoop distributed high performance computing platform. The purpose of this document is to summarize the requirements Hadoop DFS should be targeted for, and to outline further development steps towards achieving this requirements.

The requirements are divided below into groups: *reliability*, *scalability*, *functionality*, *performance*, etc. listed in the order of their importance. The requirements are intended to reflect the scale and the nature of the distributed tasks that should run under the Hadoop platform. The prioritized list of projects presented in the last section is a cumulative digest of ideas proposed by Hadoop developers that we believe will lead to achieving formulated goals. As the next step we would like to socialize and finalize the list. An important step in this direction would be building a re-factoring procedure which would let us upgrade different components of DFS step by step keeping the system functional and backward compatible all the way through the process.

DFS Scale requirements:

- Number of nodes – 10 thousand.
- Total data size – 10 PB.
  - Assuming 10,000 nodes capable of storing 1TB each. This is an order of magnitude estimate. With 750GB disks becoming commodity we could reasonably expect to have to support 750GB*4/node = 3TB/node = 30PB total.
- Number of files – 100 million.
  - If DFS data size is 10^{16} and the block size is 10^{8} then under the assumption that each file has exactly one block we need to support 10^{8} files.
  - On our current installation there is 32TB of data, using 55,000 files and folders. Scaling 32TB to 10PB under the assumption the average file size remains the same gives us an estimate of 18,000,000 files.
- Number of concurrent clients – 100 thousand.
  - If on a 10,000 cluster each node has one task tracker running 4 tasks each according to current m/r defaults then we need to support 40,000 simultaneous clients.
- Acceptable level of data loss – 1 hour.
  - Any data created or updated in DFS 1 hour ago or before is guaranteed to be recoverable in case of system failures.
- Acceptable downtime level – 2 hours.
  - DFS failure requires manual system recovery. The system is guaranteed to be available again not later than 2 hours after the recovery start.

Feature requirements:

- File Operations: open, create, close, read, append, truncate, delete, rename, undelete, get block locations, set replication, seek, tell
- Exclusive and shared access for serial/random reads and appends.
- Appenders should support flush() operation, which would flush buffer directly to DFS.
- File system Operations: readdir, directory rename, statistics (nfiles, nbytes, nblocks)
- Robust check-pointing and journaling
- Metadata versioning for backward compatibility
- Programming language agnostic client protocol
- Topology awareness (rack awareness minimally) for smarter block placement
- Automatic corruption detection and correction
- Atomic append
- Multiple appenders
- Owners, permissions, quotas

List of projects:

1. **Re-factoring.** Develop abstractions for DFS components with each component represented by an interface, specifying its functionality and interaction with other components. With good abstractions, it should be easy to add new features without compromising reliability. The abstractions should be evaluated with required future features in mind.
   For example, data nodes might have a block transfer object, a block receive object, etc., with carefully defined behavior, coordinated by a top-level control structure, instead of the morass of methods in the data node at present.
2. (Reliability) **Robust name node checkpointing** and namespace edits logging.
   Currently the system is not restorable in case of name node hardware failure.
   DFS should store “image” and “edits” files on a local name node disk and replicate them on backup nodes using a simple streaming protocol.
   HADOOP-90 Done.
3. (Reliability) Define the **startup process**, what is done by each component, in which order. Introduce a concept of **“safe mode”**, which would not make any block replication/removal decisions or change the state of the namespace in any way. Name node stays in safe mode until a configurable number of nodes have been started and reported to the name node a configurable percentage of data blocks.
   HADOOP-306 Done.
4. (Reliability) The name node **checkpoint should store a list of data nodes** serving distinct data storages that ever reported to the name node.
   Namely, the following is stored for each data node in the cluster:
   <host:port; storageID; time of last heartbeat; user id>.
   Missing nodes should be reported in the DFS UI, and during the startup. See also 3.a.
   HADOOP-456, Done.
5. (Reliability) Nodes with **read only disks** should report the problem to the name node and shut themselves down if all their local disks are unavailable.
   HADOOP-163 Done.
6. (Specification) Define **recovery/failover and software upgrade procedures**.

   a. The recovery of the cluster is manual; a document describing steps for the cluster safe recovery after a name node failure is desired.
   b. Based on the recovery procedures estimate the downtime of the cluster when the name node fails.
   c. A document is needed describing general procedures required to transition DFS from one software version to another.
   HADOOP-702], [http://issues.apache.org/jira/browse/HADOOP-702 Done.
7. (Reliability) The name node should boost the **priority of re-replicating blocks** that are far from their replication target. If necessary it should delay requests for new blocks, opening files etc., in favor of re-replicating blocks that are close to being lost forever.
   HADOOP-659 Done.
8. (Functionality) Currently DFS supports exclusive on create only **file appends**. We need more general appends that would allow re-opening files for appending. Our plan is to implement it in two steps:
   a. Exclusive appends.
   HADOOP-1700, HDFS-265 Done.
   b. Concurrent appends.
9. (Functionality) Support for **"truncate"** operation.
   This is a new functionality that is not currently supported by DFS.
10. (Functionality) **Configuration**:
   a. Accepting/rejecting rules for hosts and users based on regular expressions. The string that is matched against the regular expression should include the host, user, and cluster names.
   HADOOP-442 Done.
11. (Functionality) **DFS browsing UI.**
   Currently DFS has a rather primitive UI.
   The UI should
   a. Let browse the file system going down to each file, each file block, and further down to the block replicas.
   HADOOP-347 , HADOOP-392 Done.
   b. Report status of each directory, file, block, and block replica.
   HADOOP-347 Done.
   c. Show list of data nodes, their status, and non-operational nodes (see 4).
   HADOOP-250 Done.
   d. Show data node configuration and its extended status.
   e. List data node blocks and file names they belong to.
   f. Report the name node configuration parameters.
   g. History of data node failures, restarts, etc.
12. (Scalability) Nodes with **multiple disks** should maintain local disks data distribution internally.
   HADOOP-64 Done.
13. (Scalability) **Select-based communication** for the DFS name node.
14. (Functionality) Currently, if we want to remove x nodes from the DFS cluster, we need to remove them at most two at a time, and wait until re-replication happens, and there's no feedback on that. It would be good to specify a list of nodes to remove, have their data re-replicated while they're still online, and get a confirmation on completion.
   HADOOP-681 Done.
15. (Specification) Define **invariants for read and append** commands. A formalization of DFS consistency model with underlying assumptions and the result guarantees.
16. (Performance) Check sum data should not be stored as a separate DFS **crc-file**, but rather maintained by a data node per locally stored block copy. This will reduce name node operations and improve read data locality for maps
   HADOOP-1134 Done.
   a. **CRC scanning**. We should dedicate up to 1% of the disk bandwidth on a data node to reading back the blocks and validating their CRCs. The results should be logged and reported in the DFS UI
   HADOOP-2012 Done.
17. (Performance) DFS should operate with **constant size file blocks**.
   Currently internally DFS supposes that blocks of the same file can have different sizes. In practice all of them except for the last one have the same size. The code could be optimized if the above assumption is removed.
   Each block can be of any size up to the file's fixed block size. The DFS client provides an API to report gaps and/or an API option to skip gaps or see them as NULLs. The reporting is done at the data node level allowing us to remove all the size data & logic at the name node level.
18. (Performance) Client writes should **flush directly to DFS** based on the buffer size set at creation of the stream rather than collecting data in a temporary file on a local disk.
   HADOOP-66 Done.
19. (Performance) Currently **data nodes report** the entire list of stored blocks to the name node once in an hour. Most of this information is redundant. Processing of large reports reduces the name node availability for application tasks.
   Possible solutions:
   a. Data nodes report a portion (e.g. 20%, or bounded by the total size of transmitted data) of their blocks but (5 times) more often.
   b. Data nodes report just the delta with the removed blocks being explicitly marked as such.
   On startup the name node restores its state from a checkpoint. The checkpoint stores information about files and their blocks, but not the block locations. The locations are restored from the data node reports. That is why, at startup data nodes need to report complete lists of stored blocks. Subsequent reports do not need to contain all blocks, just the ones that have been modified since the last report.
   Each data node reports its blocks in one hour intervals. In order to avoid traffic jams the name node receives reports from different data nodes at different randomized times. Thus, on e.g. a 600 node cluster the name node receives 10 reports per minute, meaning that the block list validation happens 10 times a minute. We think it is important to minimize the reporting data size mostly from the point of view of the receiver.
   The name node should have means to request complete reports from data nodes, which is required in case the name node restarts.
   HDFS-395
20. (Performance) **Fine grained name node synchronization.** Rather than locking the whole name node for every namespace update, the name node should have only a few synchronized operations. These should be very efficient, not performing i/o and allocating few if any objects. This synchronous name node kernel should be well-defined, so that developers were aware of its boundaries.
   HADOOP-814 Done.
21. (Performance) **Compact name node** data structure. In order to support large namespaces the name node should efficiently represent internal data, which particularly mean eliminating redundant block mappings.
   Currently DFS supports the following blocks mappings:
   a. *Block to data node map (FSNamesystem.blocksMap)*
   b. *Data node to block map (FSNamesystem.datanodeMap)*
   c. *INode to block map (INode.blocks)*

    d. *Block to INode map (FSDirectory.activeBlocks)*
       HADOOP-1687 Done.
22. (Performance) Improved **block allocation schema**.
    Currently DFS randomly selects nodes from the set of data nodes that can fit the required amount of data (a block).
    Things we need:
    a. Rack locality awareness. First replica is placed on the client's local node, the second replica is placed on a node in the same rack as the client, and all other replicas are placed randomly on the nodes outside the rack.
       HADOOP-692 Done.
       Current replication policy is to place the first replica on the local node, to place the second replica on a remote rack, and to place the third replica on the same rack as the second one.
    b. Nodes with high disk usage should be avoided for block placement.
    c. Nodes with high workload should be avoided for block placement.
    d. Distinguish between fast and slow nodes (performance– and communication–wise).
23. (Performance) Equalize **disk space usage** between the nodes. The name node should regularly analyze data node disk states, and re-replicate blocks if some of them are "unusually" low or high on storage.
    HADOOP-1652 Done.
24. (Performance) Commands "open" and "list" should **not return the entire list of block locations**, but rather return a fixed number of initial blocks. Reads will fetch required block location when and if necessary.
    HADOOP-894 Done.
25. (Performance) When the **name node is busy** it should reply to data nodes that they should retry reporting their blocks later. The data nodes should retry reporting the blocks in this case earlier than the regular report would occur.
26. (Functionality) Implement **atomic append** command, also known as "record append".
27. (Functionality) Hadoop **command shell** should conform to the common shell conventions.
    a. Hadoop commands should support common options like -D, -conf, -fs, etc. Each command at the same time can have its specific options.
    b. Commons CLI should be used to implement the generic options parser and a specific command's option parsers.
    c. Support for meta-characters and regular expressions in general for file names (cp, mv, rm, ls).
    d. Interactive mode: ability to issue several commands in the same session, while keeping track of the context (such as pwd).
    e. Scripts: the ability to execute several commands recorded in a text file.
28. (Interoperability) **Slim client** design. The majority of the current client logic should be placed into a data node, called "primary", which controls the process of data transfer to other nodes, lease extension and confirmation or failure reporting if required. A thin client makes it easier to keep Java, C, etc. client implementations in sync.
    Currently the client plays the role of the primary node.
29. (Interoperability) Implement **WebDav and NFS server** mounting capabilities for DFS.

Projects yet to be prioritized:

- Data nodes should store the per-block metadata in order to make possible a **partial metadata recovery** in case the name node checkpoint information is lost.
  1. The "original" file name or id, an invariant preserved during file renames, could be stored in an additional file associated with each block, like the crc-file (see 15).
  2. Block offset (the block sequence number) could be encoded as a part of the block id, e.g.,
     `<block id> = <random unique per file #><block sequence number>`
  3. Adding concurrent file append and truncate features will require a block generation number to be stored as a part of the block file name.
- Design a **DFS backup scheme**.
  The backup is intended to prevent from data loss related to file system software bugs, particularly during the system upgrades.
  The backup might not need to store the entire data set; some applications require just a fraction of critical data so that the rest can be effectively restored.

Resources:
Video Optimization