# Scalability Limits of HDFS

Christopher Chute, David Brandfonbrener, Leo Shimonaka, Matthew Vasseur

May 2, 2016

**Abstract**

## 1 Introduction

## 2 HDFS Architecture

Hadoop claims to be built on five core principles that govern the architectural decisions. These are (1) hardware failure is the norm. As a result, HDFS wants to be highly reliable in the face of hardware failure so the architecture incorporates replication of data across multiple nodes. Then HDFS wants to (2) allow streaming data access and (3) support very large data sets. With these goals in mind, HDFS wants to use a distributed system to accommodate large files and implement distributed reads to prevent streaming access from clogging the system. Next HDFA wants to (4) provide simple concurrency control with a write-once read-many model. This is less a feature and more a decision to weigh reads over writes by providing highly available and concurrent reads at the expense of allowing concurrent writes. And lastly, HDFS wants (5) portability, which makes sense as HDFS must work on different computer architectures

To satisfy these principles, the HDFS architecture was designed as follows. An HDFS cluster will have one unique NameNode and many DataNodes. Briefly, the NameNode holds the metadata of each file in the system. This metadata consists of (inode, mapping) pairs where the inode is a UNIX-style inode representing the file and the mapping holds the information about where the file resides in disk on the DataNodes. The NameNode holds all of this metadata in memory to facilitate fast metadata operations.

This single NameNode architecture is beneficial in a few ways. First, this facilitates large reads since the reads are distributed among the DataNodes and the relatively inexpensive metadata operations all happen on the NameNode and never block each other. If each node had to handle its own metadata operations, the metadata operation to indicate which node to read from could be slowed by threads performing large data operations at the node being queried. Second, a single NameNode architecture allows for a simple durability scheme via replication of each block of data across many DataNodes and organized by the central NameNode. If the metadata was not centralized, this replication operation and updating all data mappings would become complicated, and potentially overuse the network connecting the nodes.

Now we will examine certain aspects of the specific HDFS architecture in more detail.

### 2.1 NameNode

All metadata is kept in memory on the NameNode. As explained above, the metadata holds a UNIX-style inode and location on DataNodes for each file. These locations are represented as block locations in virtual memory of each of the DataNodes. So, upon creation of a file, the NameNode will allocate a block for the file in virtual memory on however many DataNodes the file will be replicated across (the default is 3 replicas). Then, the NameNode is responsible to pointing clients to the appropriate blocks to write or read files, and also to maintain the namespace. The NameNode keeps these maps to data blocks up to date by communicating with the DataNodes via heartbeats. The DataNodes are responsible for sending periodic heartbeats to indicate that they are functioning correctly that contain block reports. The block report

Clearly durability of the NameNode is very important to the system. There are various schemes to ensure a durable NameNode. A write ahead log is maintained on stable storage and this can be used to build a BackupNode or CheckpointNode that essentially maintain a copy of the NameNode.

### 2.2 DataNode

### 2.3 Reads and Writes in HDFS

## 3 Limitations of Single Namenode Architecture

### 3.1 Physical Memory Size

### 3.2 Namenode CPU Bottleneck

## 4 Performance Evaluation

### 4.1 Experiment 1: Memory Experiments

### 4.2 Experiment 2: CPU Experiments

### 4.3 Analysis

## 5 Improvements to Single Namenode Architecture

## 6 Conclusion