

# Design and Evolution of the Apache Hadoop File System(HDFS)

[Dhruba Borthakur](#)

Engineer@Facebook  
Committer@Apache HDFS  
SDC, Sept 19 2011

- ❑ **Introduction**
- ❑ **Yet another file-system, why?**
  - ❑ Goals of Hadoop Distributed File System (HDFS)
  - ❑ Architecture Overview
- ❑ **Rational for Design Decisions**

# Who Am I?

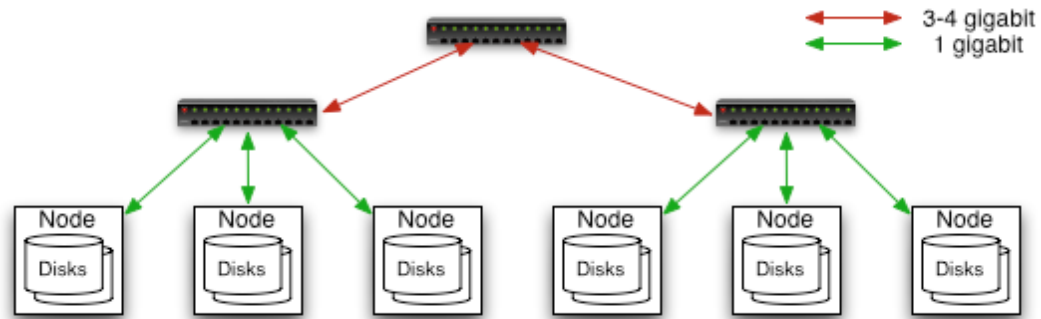
- ❑ **Apache Hadoop FileSystem (HDFS)**
  - ❑ Committer and PMC Member
  - ❑ Core contributor since Hadoop's infancy
- ❑ **Facebook** (Hadoop, Hive, Scribe)
- ❑ **Yahoo!** (Hadoop in Yahoo Search)
- ❑ **Veritas** (San Point Direct, Veritas File System)
- ❑ **IBM Transarc** (Andrew File System)
- ❑ **Univ of Wisconsin** Computer Science Alumni  
(Condor Project)

# Hadoop, Why?

- ❑ **Need to process Multi Petabyte Datasets**
- ❑ **Data may not have strict schema**
- ❑ **Expensive to build reliability in each application.**
  - ❑ Failure is expected, rather than exceptional.
  - ❑ Elasticity, # of nodes in a cluster is never constant.
- ❑ **Need common infrastructure**
  - ❑ Efficient, reliable, Open Source Apache License

- ❑ **Very Large Distributed File System**
  - ❑ 10K nodes, 1 billion files, 100 PB
- ❑ **Assumes Commodity Hardware**
  - ❑ Files are replicated to handle hardware failure
  - ❑ Detect failures and recovers from them
- ❑ **Optimized for Batch Processing**
  - ❑ Data locations exposed so that computations can move to where data resides
  - ❑ Provides very high aggregate bandwidth
- ❑ **User Space, runs on heterogeneous OS**

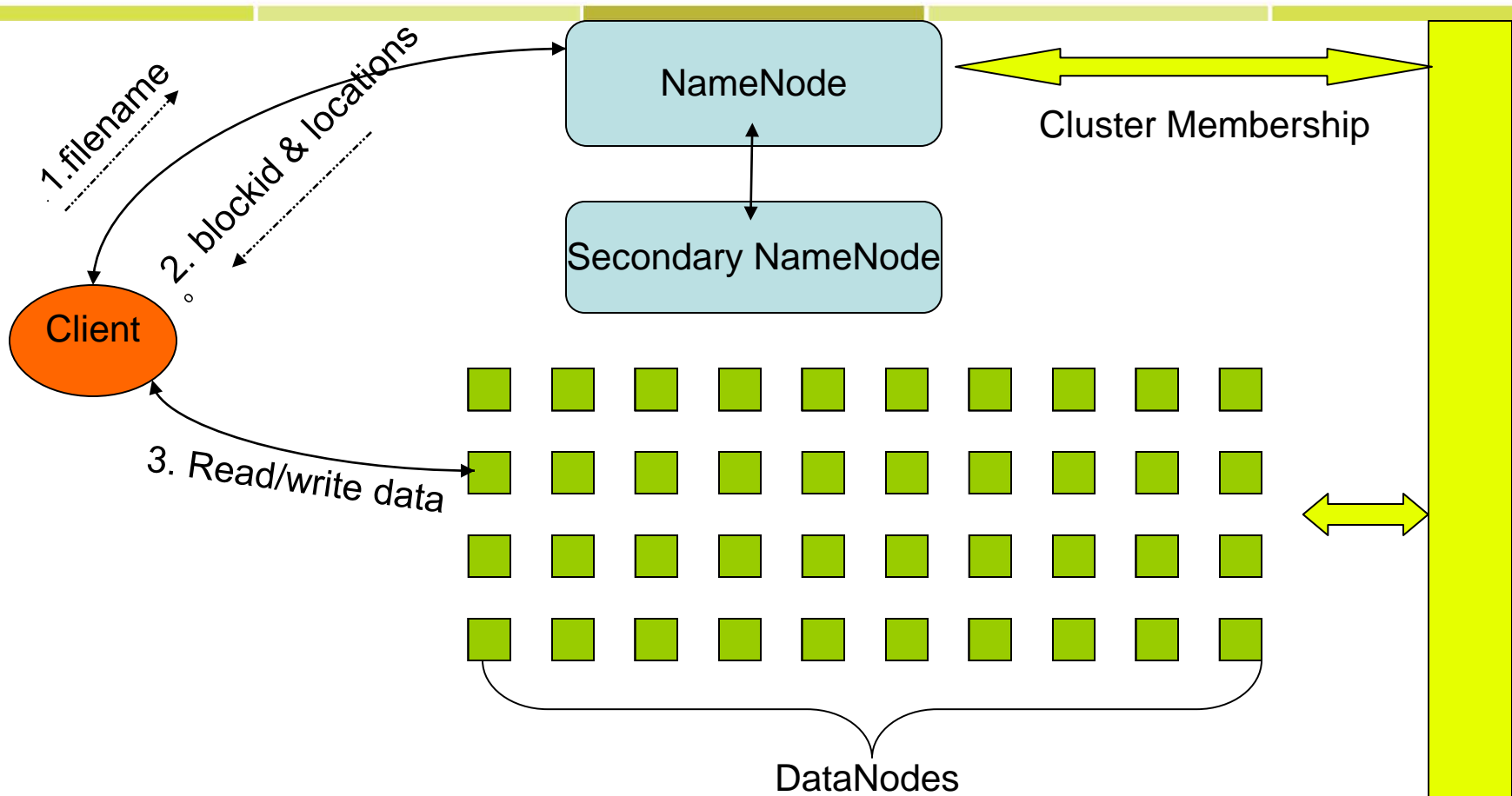
# Commodity Hardware



## Typically in 2 level architecture

- Nodes are commodity PCs
- 20-40 nodes/rack
- Uplink from rack is 4 gigabit
- Rack-internal is 1 gigabit

# HDFS Architecture



- NameNode : Maps a file to a file-id and list of DataNodes
- DataNode : Maps a block-id to a physical location on disk
- SecondaryNameNode: Periodic merge of Transaction log



# Distributed File System

- ❑ **Single Namespace for entire cluster**
- ❑ **Data Coherency**
  - ❑ Write-once-read-many access model
  - ❑ Client can only append to existing files
- ❑ **Files are broken up into blocks**
  - ❑ Each block replicated on multiple DataNodes
- ❑ **Intelligent Client**
  - ❑ Client can find location of blocks
  - ❑ Client accesses data directly from DataNode

# • Why all metadata in main-memory?

- ❑ **Most other FS (ext3, zfs, xfs, Vxfs, etc) keeps only the hotset of metadata in memory**
- ❑ **But entire HDFS Meta-data in RAM**
  - ❑ Information of all files, blocks, locations, etc.
- ❑ **Why is it not demand paged from disk?**
  - ❑ Metadata operation is low latency and fast
  - ❑ Nodes failure is unpredictable -- keep all block locations in memory for quick re-replication
  - ❑ IOPs on HDFS-transaction log disk is never a scalability bottleneck (no random reads)

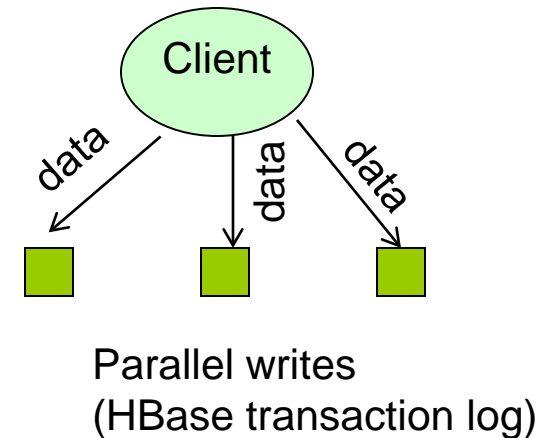
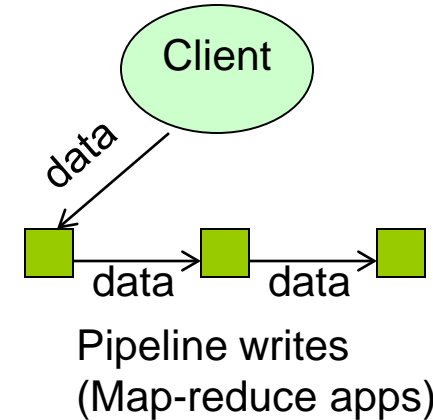
# Why are writes pipelined?

## □ What is a write-pipeline?

- Client writes data to the first DataNode
- The first DataNode forwards the data to the next DataNode in the pipeline, and so on

## □ Pipeline vs Parallel writes

- Saves inter-rack network bandwidth
- Multi-terabyte data sets are frequently copied from some outside location

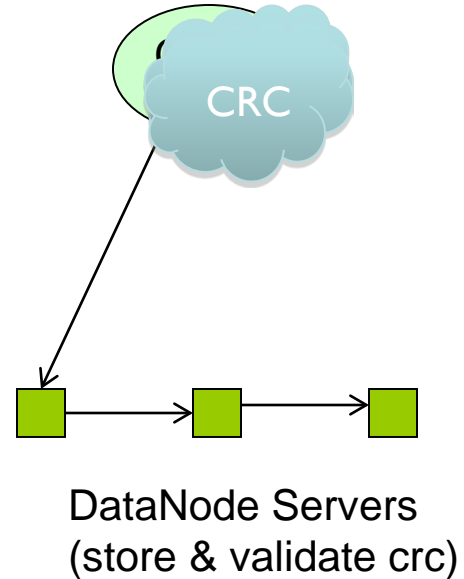


# How to place Block Replicas?

- ❑ **Block Replica placement tradeoffs**
  - ❑ Replicas on separate racks: higher availability
  - ❑ Replicas on one rack: better network utilization
- ❑ **HDFS chooses a little of both**
  - ❑ One replica on local node
  - ❑ Second and third replica on a remote rack
  - ❑ Additional replicas are randomly placed

# Why Checksum on HDFS Client?

- ❑ **HDFS Client provides data integrity**
  - ❑ Generate crc on file writes
  - ❑ Validate crc on file reads
- ❑ **Better than server-side checksums**
  - ❑ Detects corruptions in network transfer
  - ❑ Detects bit flips in main-memory on application server



# NameNode Transaction Logs

- ❑ **Why multiple copies of Transaction Log?**
  - ❑ Increase reliability
- ❑ **Transaction Log stored in multiple directories**
  - ❑ Local file system and a NFS location
- ❑ **What if NFS location is inaccessible?**
  - ❑ NameNode continues to function with local file system
  - ❑ Raises appropriate alarms for administrator
- ❑ **Can the dependency on NFS be removed?**
  - ❑ Maybe in future, needs reserved BlockIds

# High Availability

## □ Active-Standby Pair

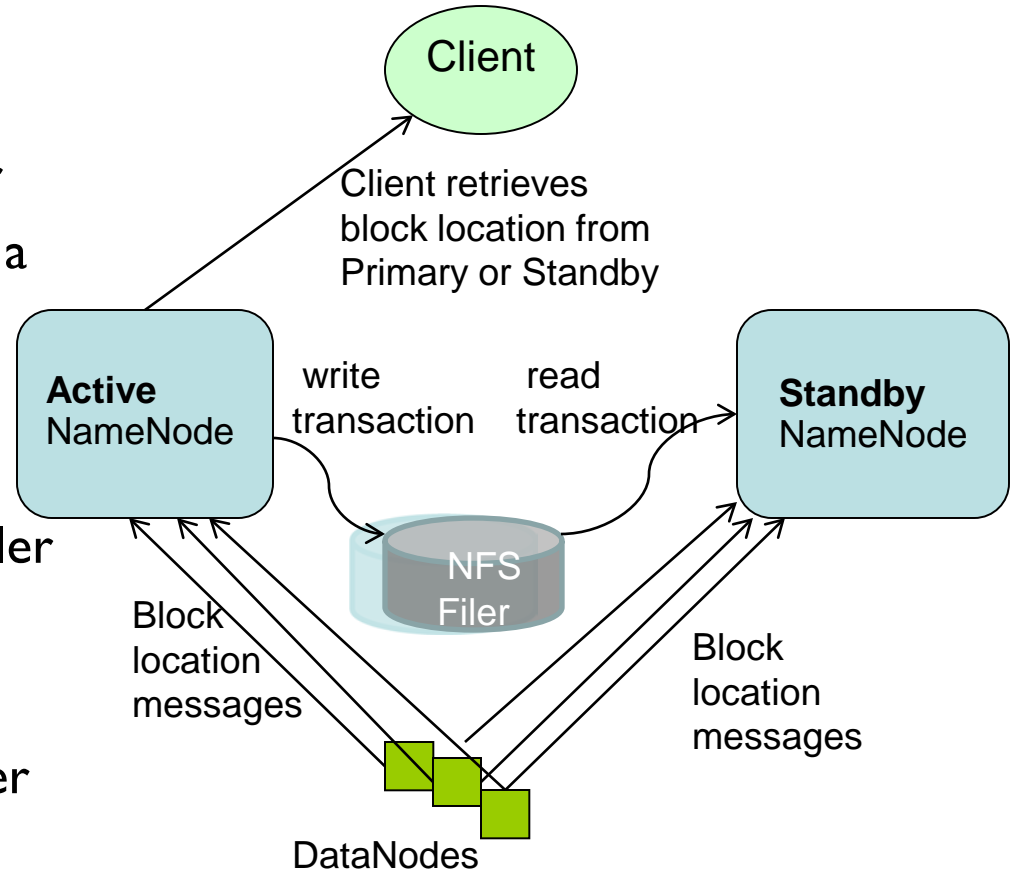
- Coordinated via zookeeper
- Failover in few seconds for a fs with 100 million files

## □ Active NameNode

- Writes transaction log to filer

## □ Standby NameNode

- Reads transactions from filer
- Latest metadata in memory



<http://hadoopblog.blogspot.com/2010/02/hadoop-namenode-high-availability.html>

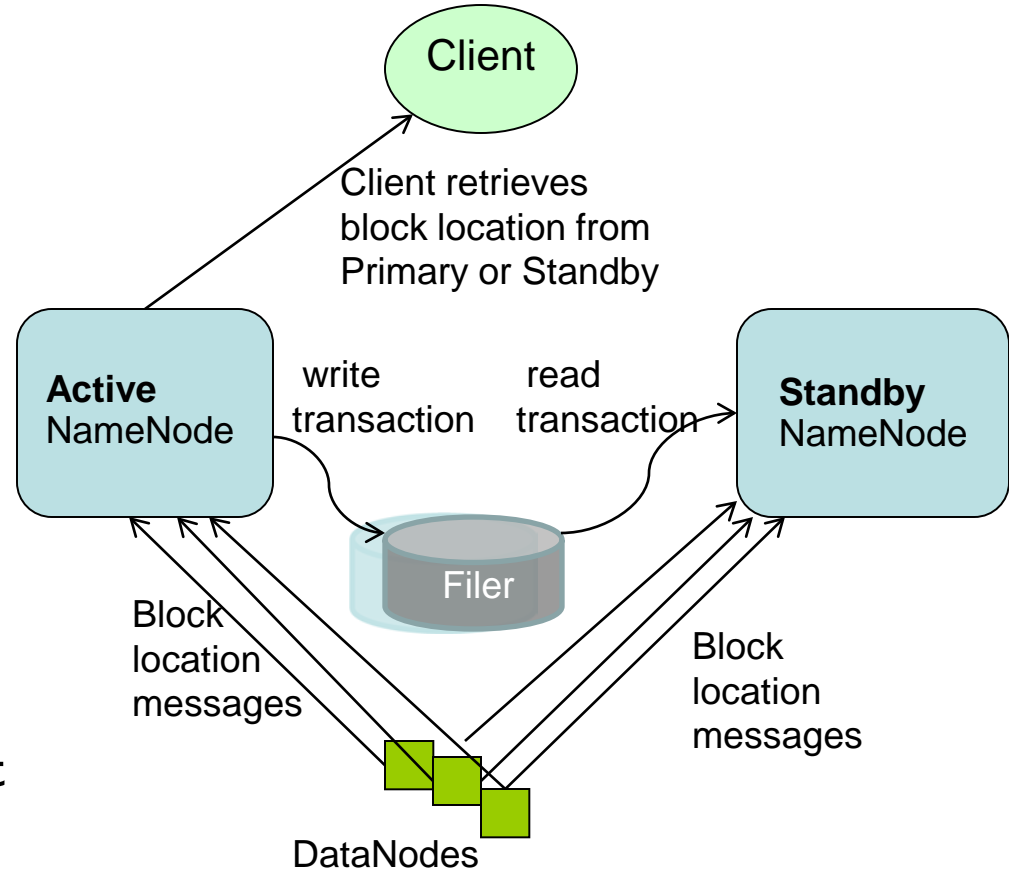
# NameNode High Availability

## Observations

- Most failures on NameNode occur because of bad nic card, failed memory bank, etc
- Only one failure in 4 years because of HDFS software bug

## Design Consideration

- Active-Passive pair, standby can serve stale read requests
- External filer provides elegant solution for IO fencing (for HDFS transaction log)

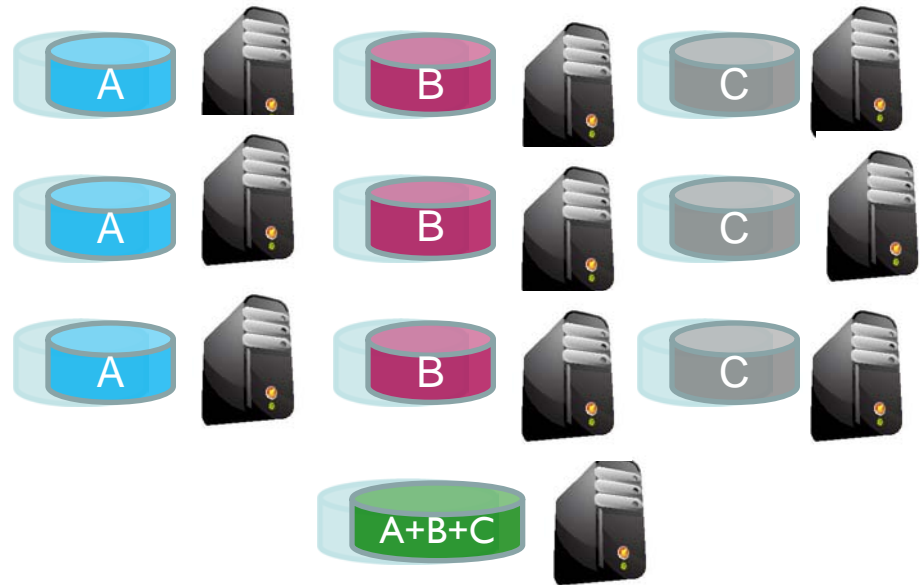


# How Elastic is HDFS?

- ❑ **Nodes constantly added to cluster**
  - ❑ New nodes to expand cluster capacity
  - ❑ Nodes coming back from repair
- ❑ **Automatic Rebalance**
  - ❑ % disk full on DataNodes should be similar
- ❑ **Disadvantages**
  - ❑ Does not rebalance based on access patterns or load
  - ❑ No support for automatic handling of hotspots of data

# HDFS Raid

- ❑ **Start the same:** triplicate every data block
- ❑ **Background encoding**
  - ❑ Combine third replica of blocks from a single file to create parity block
  - ❑ Remove third replica
- ❑ **Policy**
  - ❑ Triplicate most recent data
  - ❑ XOR encoding for files older than a week
  - ❑ Reed Solomon encoding for much older files



A file with three blocks A, B and C

<http://hadoopblog.blogspot.com/2009/08/hdfs-and-erasure-codes-hdfs-raid.html>

# Why is RAID-ing asynchronous?

- ❑ **More replicas for query performance**
  - ❑ A large percentage of queries access recent data
  - ❑ Keep three or more replicas for most recent data
- ❑ **Background encoding**
  - ❑ No impact on new writes to files
  - ❑ Intelligently schedule encoding when cluster is less busy
  - ❑ Throttle network bandwidth needed for encoding

# Why are HDFS blocks so big?

- ❑ Typically 128 MB to 256 GB for map-reduce applications
- ❑ Typically 1 GB – 2 GB for archival store
- ❑ Mostly sequential reads
- ❑ Sequential writes
- ❑ IOPs on disk is not a bottleneck

# HDFS read/write coherency

- ❑ No coherency between readers and writers
  - ❑ Very helpful for scaling out
  - ❑ Typically, 60K – 100K processes are reading different files from a single filesystem
- ❑ A client can read a file even when it is being written
- ❑ Typical use-cases do not need read/write consistency
  - ❑ Map-Reduce
  - ❑ HBase Database
  - ❑ Archival Store

# How to support larger clusters?

## ❑ Two main choices

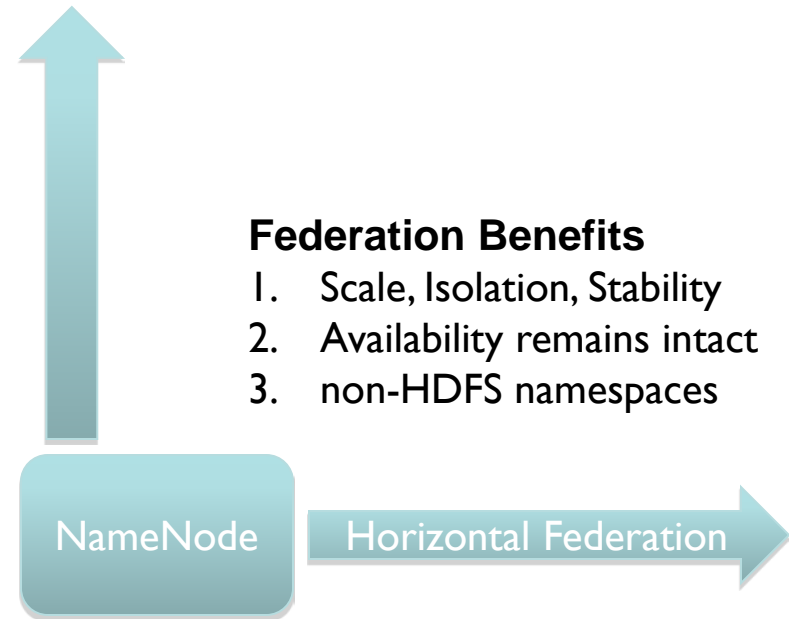
- ❑ Larger NameNode
- ❑ Multiple NameNodes

## ❑ Chose Horizontal Federation

- ❑ Partition namespace into multiple NameNodes
- ❑ A DataNode can be part of multiple NameNodes

## Vertical Scaling

1. More RAM, Efficiency in memory usage
2. First class archives (tar/zip like)
3. Partial namespace in main memory



## Federation Benefits

1. Scale, Isolation, Stability
2. Availability remains intact
3. non-HDFS namespaces

## □ **HDFS design choices**

- Focused on scalability, fault tolerance and performance
- Evolving according to current requirements

## □ **Contribute to HDFS**

- New code contributors are most welcome
- <http://wiki.apache.org/hadoop/HowToContribute>

## □ **HDFS Design:**

- [http://hadoop.apache.org/core/docs/current/hdfs\\_design.html](http://hadoop.apache.org/core/docs/current/hdfs_design.html)

## □ **My Hadoop Blog:**

- <http://hadoopblog.blogspot.com/>
- <http://www.facebook.com/hadoopfs>