Hadoop Distributed File System (HDFS) Overview

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

Agenda

- Introduction
- Architecture and Concepts
- Access Options

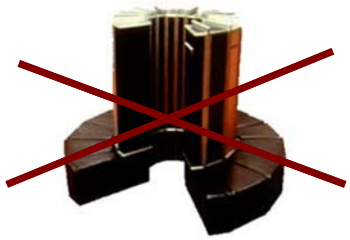
HDFS

- Appears as a single disk
- Runs on top of a native filesystem
  - Ext3,Ext4,XFS
- Based on Google's Filesystem GFS
- Fault Tolerant
  - Can handle disk crashes, machine crashes, etc...
- Based on Google's Filesystem (GFS or GoogleFS)
  - gfs-sosp2003.pdf
  - http://en.wikipedia.org/wiki/Google_File_System

Use Commodity Hardware

- **"Cheap" Commodity Server Hardware**
  – No need for super-computers, use commodity unreliable hardware
  – Not desktops!

NOT



BUT

HDFS is Good for...

- **Storing large files**
  – Terabytes, Petabytes, etc...
  – Millions rather than billions of files
  – 100MB or more per file
- **Streaming data**
  – Write once and read-many times patterns
  – Optimized for streaming reads rather than random reads
  – Append operation added to Hadoop 0.21
- **"Cheap" Commodity Hardware**
  – No need for super-computers, use less reliable commodity hardware

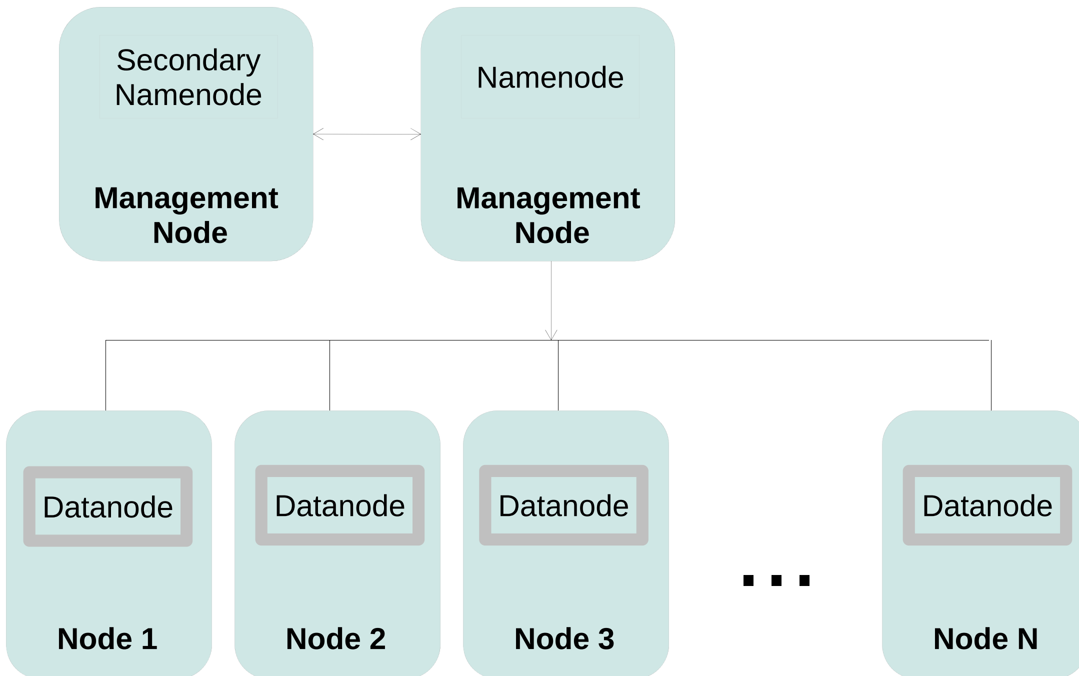
HDFS is not so good for...

- **Low-latency reads**
  - High-throughput rather than low latency for small chunks of data
  - HBase addresses this issue
- **Large amount of small files**
  - Better for millions of large files instead of billions of small files
    - For example each file can be 100MB or more
- **Multiple Writers**
  - Single writer per file
  - Writes only at the end of file, no-support for arbitrary offset

HDFS Daemons

- **Filesystem cluster is manager by three types of processes**
  - Namenode
    - manages the File System's namespace/meta-data/file blocks
    - Runs on 1 machine to several machines
  - Datanode
    - Stores and retrieves data blocks
    - Reports to Namenode
    - Runs on many machines
  - Secondary Namenode
    - Performs house keeping work so Namenode doesn't have to
    - Requires similar hardware as Namenode machine
    - Not used for high-availability – not a backup for Namenode

HDFS Daemons

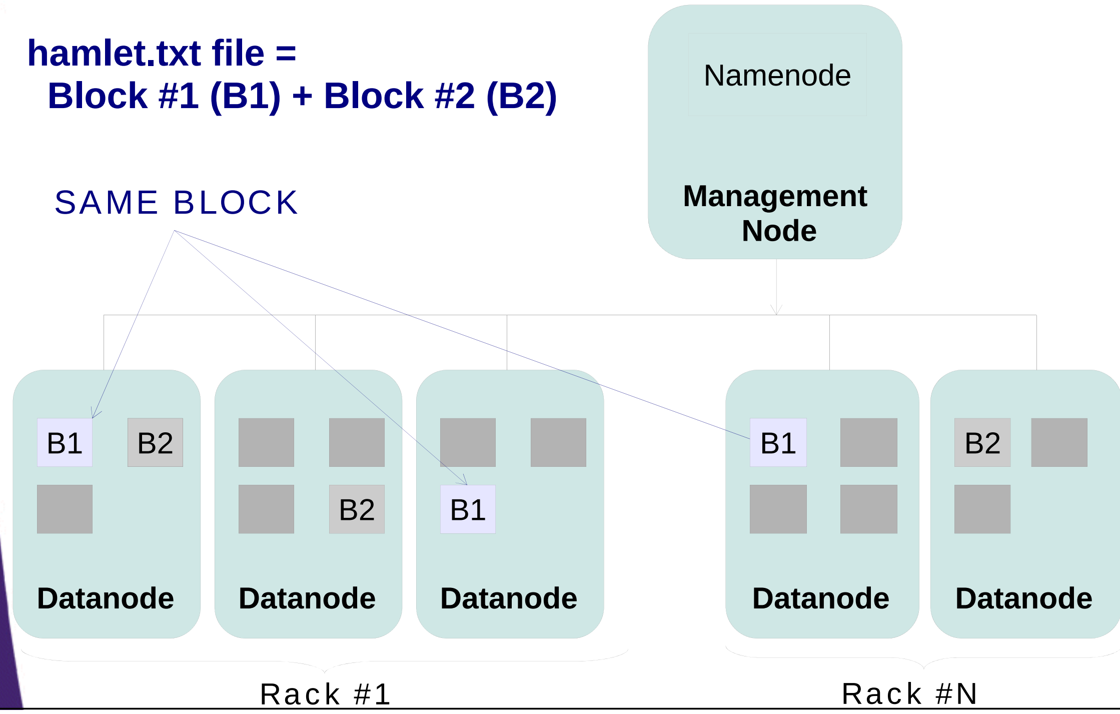


Files and Blocks

- **Files are split into blocks (single unit of storage)**
  – Managed by Namenode, stored by Datanode
  – Transparent to user
- **Replicated across machines at load time**
  – Same block is stored on multiple machines
  – Good for fault-tolerance and access
  – Default replication is 3

Files and Blocks

**hamlet.txt file =
Block #1 (B1) + Block #2 (B2)**



HDFS Blocks

- **Blocks are traditionally either 64MB or 128MB**
  - Default is 64MB
- **The motivation is to minimize the cost of seeks as compared to transfer rate**
  - 'Time to transfer' > 'Time to seek'
- **For example, lets say**
  - seek time = 10ms
  - Transfer rate = 100 MB/s
- **To achieve seek time of 1% transfer rate**
  - Block size will need to be = 100MB

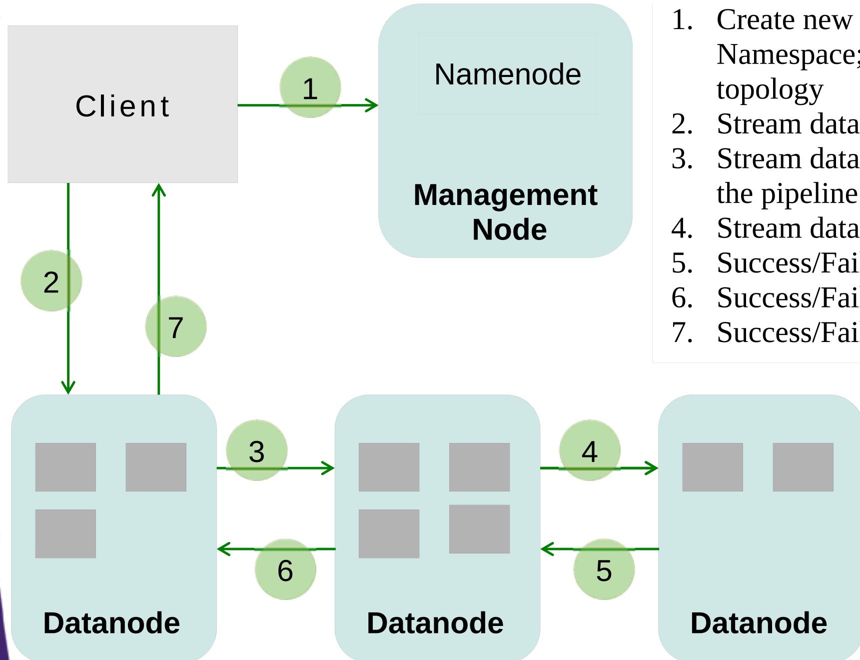
Block Replication

- **Namenode determines replica placement**
- **Replica placements are rack aware**
  - Balance between reliability and performance
    - Attempts to reduce bandwidth
    - Attempts to improve reliability by putting replicas on multiple racks
  - Default replication is 3
    - 1st replica on the local rack
    - 2nd replica on the local rack but different machine
    - 3rd replica on the different rack
  - This policy may change/improve in the future

Client, Namenode, and Datanodes

- **Namenode does NOT directly write or read data**
  - One of the reasons for HDFS's Scalability
- **Client interacts with Namenode to update Namenode's HDFS namespace and retrieve block locations for writing and reading**
- **Client interacts directly with Datanode to read/write data**

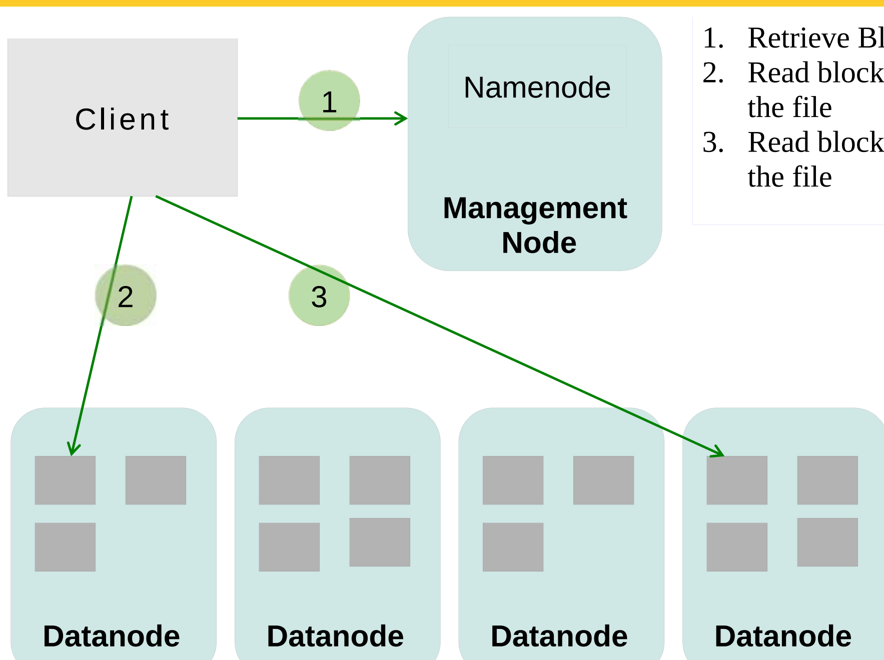
HDFS File Write



1. Create new file in the Namenode's Namespace; calculate block topology
2. Stream data to the first Node
3. Stream data to the second node in the pipeline
4. Stream data to the third node
5. Success/Failure acknowledgment
6. Success/Failure acknowledgment
7. Success/Failure acknowledgment

Source: White, Tom. Hadoop The Definitive Guide. O'Reilly Media. 2012

HDFS File Read



1. Retrieve Block Locations
2. Read blocks to re-assemble the file
3. Read blocks to re-assemble the file

Source: White, Tom. Hadoop The Definitive Guide. O'Reilly Media. 2012

Namenode Memory Concerns

- **For fast access Namenode keeps all block metadata in-memory**
  - The bigger the cluster - the more RAM required
    - Best for millions of large files (100mb or more) rather than billions
    - Will work well for clusters of 100s machines
- **Hadoop 2+**
  - Namenode Federations
    - Each namenode will host part of the blocks
    - Horizontally scale the Namenode
  - Support for 1000+ machine clusters
    - Yahoo! runs 50,000+ machines
  - Learn more @ http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/Federation.html

Namenode Memory Concerns

- **Changing block size will affect how much space a cluster can host**
  - 64MB to 128MB will reduce the number of blocks and significantly increase how much space the Namenode will be able to support
  - Example:
    - Let's say we are storing 200 Terabytes = 209,715,200 MB
    - With 64MB block size that equates to 3,276,800 blocks
      - 209,715,200MB / 64MB = 3,276,800 blocks
    - With 128MB block size it will be 1,638,400 blocks
      - 209,715,200MB / 128MB = 1,638,400 blocks

Namenode's fault-tolerance

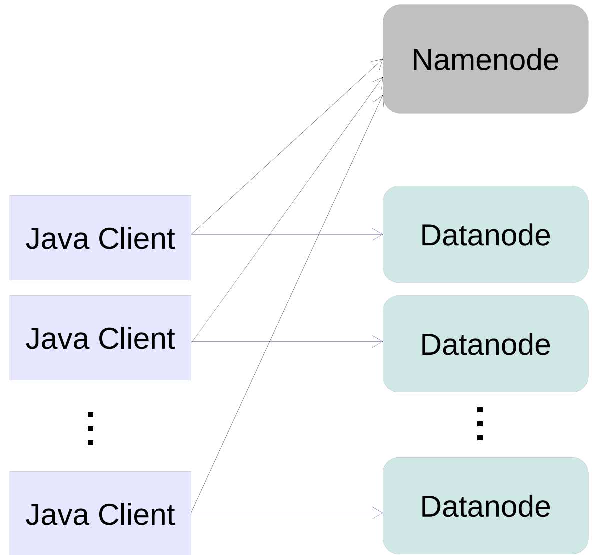
- **Namenode daemon process must be running at all times**
  - If process crashes then cluster is down
- **Namenode is a single point of failure**
  - Host on a machine with reliable hardware (ex. sustain a disk-failure)
  - Usually is not an issue
- **Hadoop 2+**
  - High Availability Namenode
    - Active Standby is always running and takes over in case main namenode fails
    - Still in its infancy
  - Learn more @ http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailability.html

HDFS Access

- **Access Patterns**
  - Direct
    - Communicate with HDFS directly through native client
    - Java, C++
  - Proxy Server
    - Access HDFS through a Proxy Server – middle man
    - REST, Thrift, and Avro Servers

Direct Access

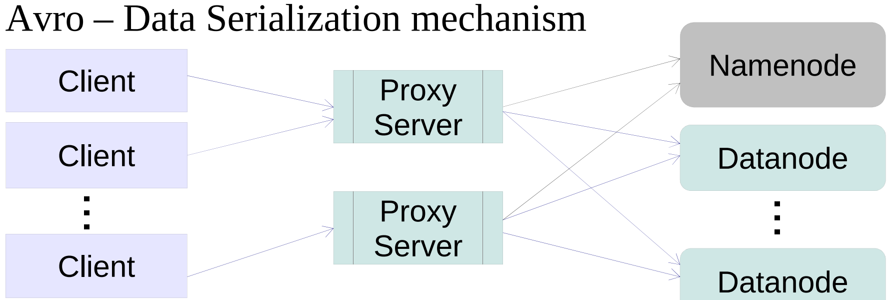
- **Java and C++ APIs**
- **Clients retrieve metadata such as blocks' locations from Namenode**
- **Client directly access datanode(s)**
- **Java API**
  - Most commonly used
  - Covered in this course
- **Used by MapReduce**



Source: White, Tom. Hadoop The Definitive Guide. O'Reilly Media. 2012

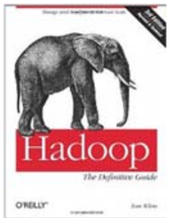
Proxy Based Access

- **Clients communicate through a proxy**
  - Strives to be language independent
- **Several Proxy Servers are packaged with Hadoop:**
  - Thrift – interface definition language
  - WebHDFS REST – response formatted in JSON, XML or Protocol Buffers
  - Avro – Data Serialization mechanism



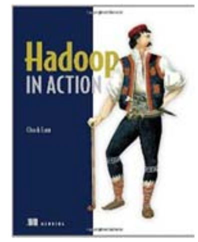
Source: White, Tom. Hadoop The Definitive Guide. O'Reilly Media. 2012

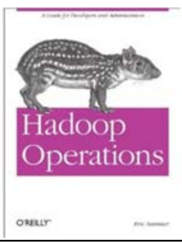
Resources: Books



Hadoop: The Definitive Guide
HDFS Chapters
Tom White (Author)
O'Reilly Media; 3rd Edition (May6, 2012)

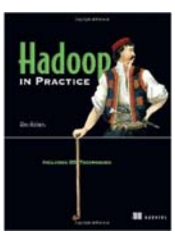
Hadoop in Action
HDFS Chapter
Chuck Lam (Author)
Manning Publications; 1st Edition (December, 2010)





Hadoop Operations
HDFS Chapters
Eric Sammer (Author)
O'Reilly Media (October 22, 2012)

Resources: Books



Hadoop in Practice
HDFS Chapters
Alex Holmes (Author)
Manning Publications; (October 10, 2012)

Resources

- **Home Page**
  – http://hadoop.apache.org
- **Mailing Lists**
  – http://hadoop.apache.org/mailing_lists.html
- **Wiki**
  – http://wiki.apache.org/hadoop
- **Documentation is sprinkled across many pages and versions:**
  – http://hadoop.apache.org/docs/hdfs/current
  – http://hadoop.apache.org/docs/r2.0.2-alpha/
  – HDFS Guides: http://hadoop.apache.org/docs/r0.20.2

Wrap-Up

Summary

- **We learned about**
  – HDFS Use Cases
  – HDFS Daemons
  – Files and Blocks
  – Namenode Memory Concerns
  – Secondary Namenode
  – HDFS Access Options

Questions?