Distributed File Systems

Mauro Fruet

University of Trento - Italy

2011/12/19

Outline

1. Distributed File Systems
2. The Google File System (GFS)
3. The Hadoop Distributed File System (HDFS)
4. Conclusions
5. Bibliography

Introduction

Definition of DFS

File system that allows access to files from multiple hosts via a computer network

Features

- Share files and storage resources on multiple machines
- No direct access to data storage for clients
- Clients interact using a protocol
- Access restrictions to file system

DFS Goals

- Access transparency
- Location transparency
- Concurrent file updates
- File replication
- Hardware and software hetereogenity
- Fault tolerance
- Consistency
- Security
- Efficiency

Most Important DFS

NFS (Network File System), developed by Sun Microsystems in 1984

- Small number of clients, very small number of servers
- Any machine can be a client and/or a server
- Stateless file server with few exceptions (file locking)
- High performance

AFS (Andrew File System), developed by the Carnegie Mellon University

- Support sharing on a large scale (up to 10,000 users)
- Client machines have disks (cache files for long periods)
- Most files are small
- Reads more common than writes
- Most files read/written by one user

Need for Large Streams of Data

Standard DFS not adequate to manage large streams of data:

- Facebook: 25 TB/day for logs
- CERN: 40 TB/day - 15 PB/year

Such workloads need different assumptions and goals. Proposed solutions:

- The Google File System (GFS)
- Hadoop Distributed File System (HDFS) by Apache Software Foundation

The Google File System (GFS) [1]

- Proprietary DFS, only key ideas available
- Developed and implemented by Google in 2003
- Shares many goals with standard DFS (performance, scalability, reliability, availability,...)
- Needed to manage data-intensive applications
- Provides fault tolerance
- Provides high performance

Assumptions

- Composed of inexpensive commodity components that often fail
- Hundreds of thousands of commodity machines
- Modest number of huge files (many GBs)
- Large streaming reads and small random reads
- Append to files rather than overwrite
- Files seldom modified again

Architecture

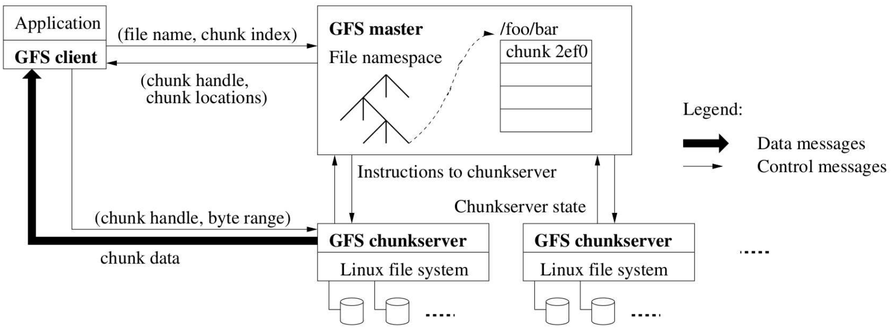


Figure: GFS Architecture

Architecture

- Master has file system metadata:
  - Namespace
  - Access control information
  - Mapping from files to chunks
  - Current locations of chunks
- Master controls:
  - Chunk lease management
  - Garbage collection
  - Chunk migration
- Periodical HeartBeat messages between master and chunkservers

Metadata

- All metadata in main memory of the master
- Mutations logged to an operation log
- No persistent record of replicas locations
- Operation Log:
    - Contains critical metadata changes
    - Defines order of concurrent operations
    - Replicated on multiple remote machines
    - Used to recover the master
- Checkpoint of master if log beyond certain size:
    - Master switches to a new log
    - Checkpoint created with all mutations before switch

Consistency Model

- File namespace mutations are atomic
- Namespace locking guarantees atomicity and correctness
- Mutations applied in same order on all replicas
- Chunk version numbers used to detect any stale replica (missed mutations while chunkserver was down)
- Stale replicas are garbage collected
- Checksum for detection of data corruption

Leases and Mutation Order

- Use of leases to maintain consistent mutation order
- Master grants a chunk lease to a replica, called primary
- Primary chooses serial order
- Lease grant order defined by the master
- Order within a lease defined by the primary

Control Flow of a Write

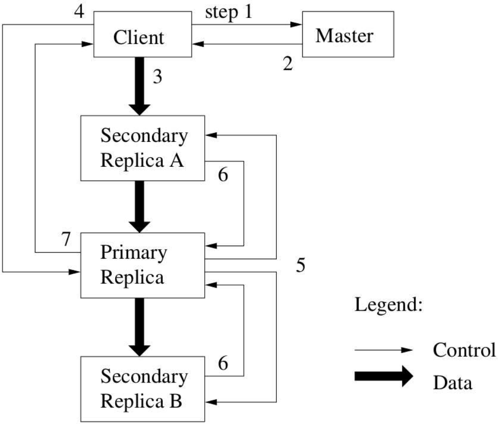


Figure: Write Control and Data Flow

Data Flow

- Separated flow of data and control
- Goals:
  1. Fully use outgoing bandwidth of each machine:
     - Data pushed in a chain of chunkservers
  2. Avoid network bottlenecks and high-latency links:
     - Each machine forwards data to “closest” machine
  3. Minimize latency:
     - Once a chunkserver receives data, it starts forwarding immediately

Snapshots

- Used to:
  1. Quickly create copies of huge data sets
  2. Checkpoint current state
- The master:
  1. Revokes leases on chunks of the files it is about to snapshot
  2. Logs the operation to disk
  3. Duplicates the corresponding metadata
- New snapshot file points to same chunk C of source file
- New chunk C' created for first write after snapshot

Namespace Management and Locking

- No per-directory data structure
- No hard or symbolic links
- Namespace maps full pathnames to metadata
- Each master operation acquires a set of locks before it runs
- Read lock on internal nodes and read/write lock on the leaf
- Allowed concurrent mutations in the same directory
- Read lock on directory prevents its deletion, renaming or snapshot

Replica Placement

- Hundreds of chunkservers across many racks
- Chunkservers accessed from hundreds of clients
- Goals:
  - Maximize data reliability and availability
  - Maximize network bandwidth utilization
- Replicas spread across machines and racks

Creation, Re-replication and Rebalancing

Creation

- Place new replicas on chunkservers with below-average disk usage
- Limit number of recent creations on each chunkservers

Re-replication

- When number of available replicas falls below a user-specified goal
- Prioritization: based on how far it is from its replication goal

Rebalancing

- Periodically, for better disk utilization and load balancing
- Distribution of replicas is analyzed

Garbage Collection

- File deletion logged by master
- File renamed to a hidden name with deletion timestamp
- Master regularly deletes files older than 3 days (configurable)
- Until then, hidden file can be read and undeleted
- When a hidden file is removed, its in-memory metadata is erased
- Orphaned chunks identified, corresponding metadata erased
- Safety against accidental irreversible deletion

Stale Replica Detection

- Need to distinguish between up-to-date and stale replicas
- Chunk version number:
  - Increased when master grants new lease on the chunk
  - Not increased if replica is unavailable
- Stale replicas deleted by master in regular garbage collection

High Availability

Fast Recovery

Master and chunkservers have to restore their state and start in seconds no matter how they terminated

Master Replication

- Master state replicated for reliability on multiple machines
- When master fails:
  - It can restart almost instantly
  - A new master process is started elsewhere
- "shadow" (not mirror) master provides only read-only access to file system when primary master is down

Data Integrity

- Checksums to detect corruption of stored data
- Integrity verified by each chunkserver
- Corruptions not propagated to other chunkservers

The Hadoop Distributed File System (HDFS) [2]

- Sub-project of Apache Hadoop project
- Primary storage system used by Hadoop applications
- Creates multiple replicas of data blocks
- Distributes replicas on nodes
- Highly fault-tolerant
- Designed to run on commodity hardware
- Suitable for applications for large data sets

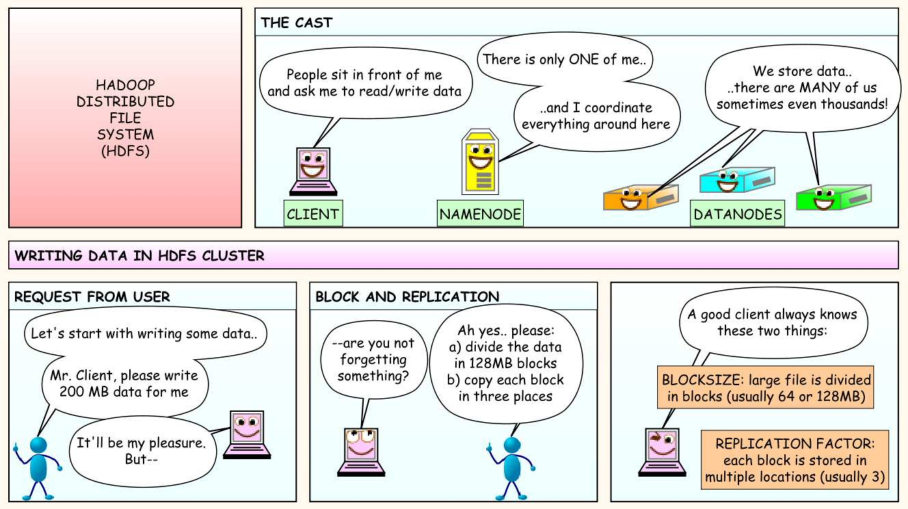
Powered By

- HDFS used by some of the world's biggest Web sites
- Controls the top search engines

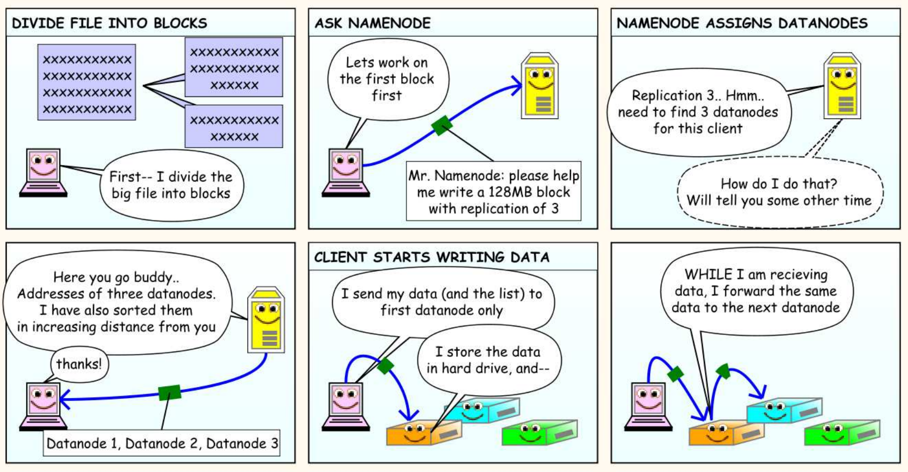
Powered By

- Adobe since 2008: 30 nodes; clusters from 5 to 14 nodes; plan on a 80 nodes cluster
- eBay: 532 nodes cluster; 8*532 cores, 5.3 PB
- Facebook: one 1,100 nodes cluster with 8,800 cores and 12 PB; one 300 nodes cluster with 2,400 cores and 3 PB
- Twitter: 12 TB/day
- Yahoo!: more than 100,000 CPUs in more than 40,000 PCs; biggest cluster: 4,500 nodes

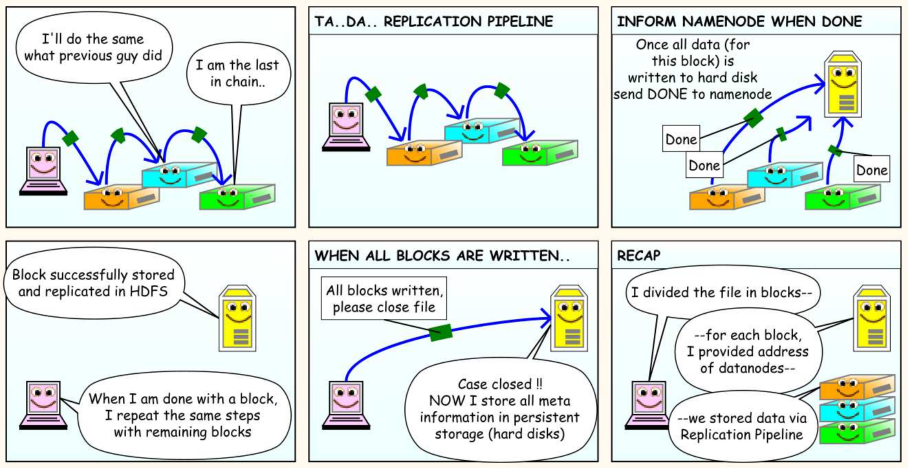
Writing Data in HDFS Cluster



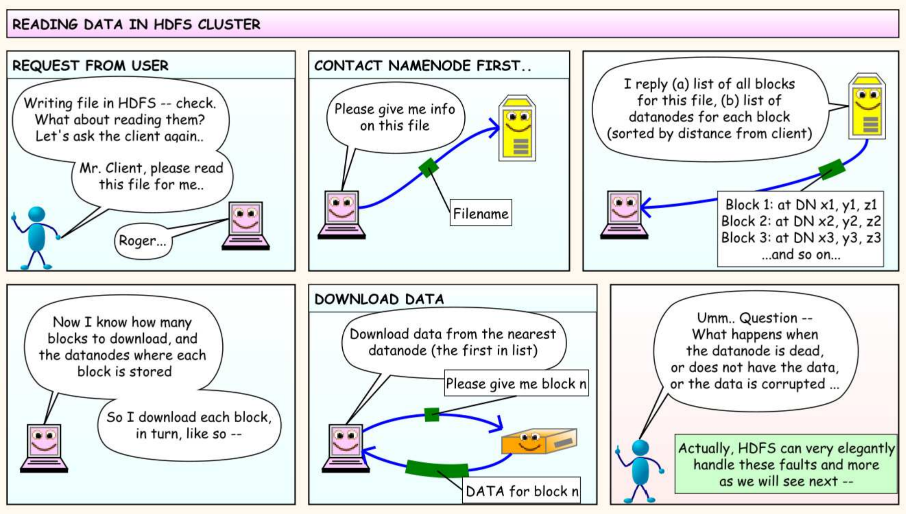
Writing Data in HDFS Cluster



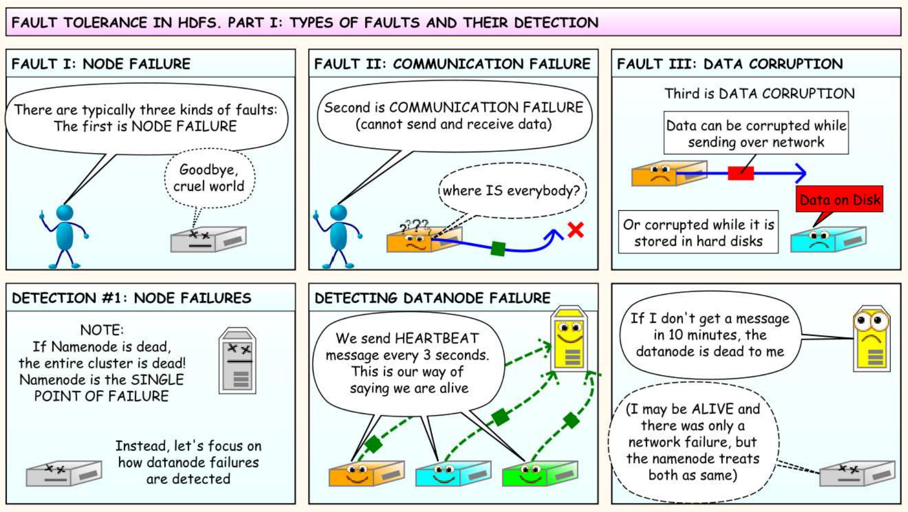
Writing Data in HDFS Cluster



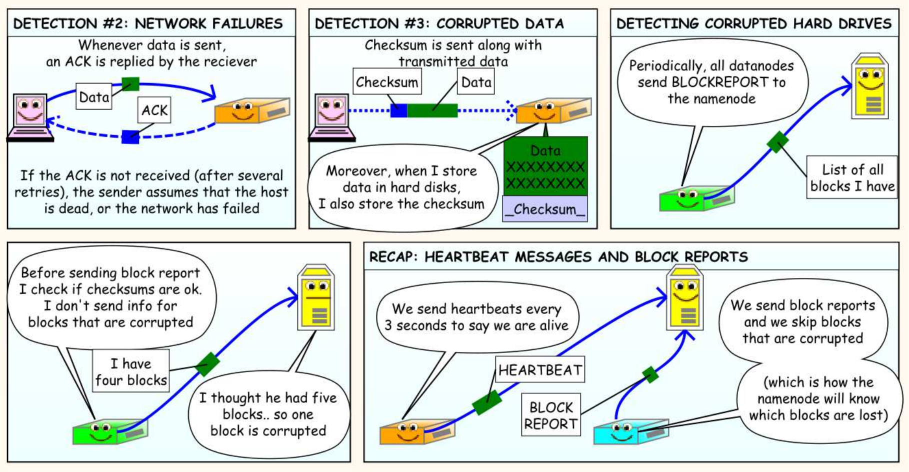
Reading Data in HDFS Cluster



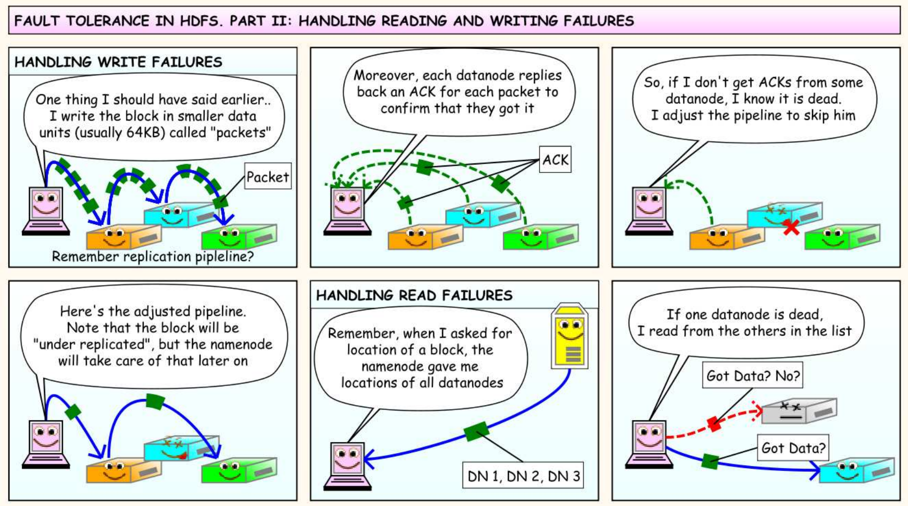
Types of Faults and Their Detection



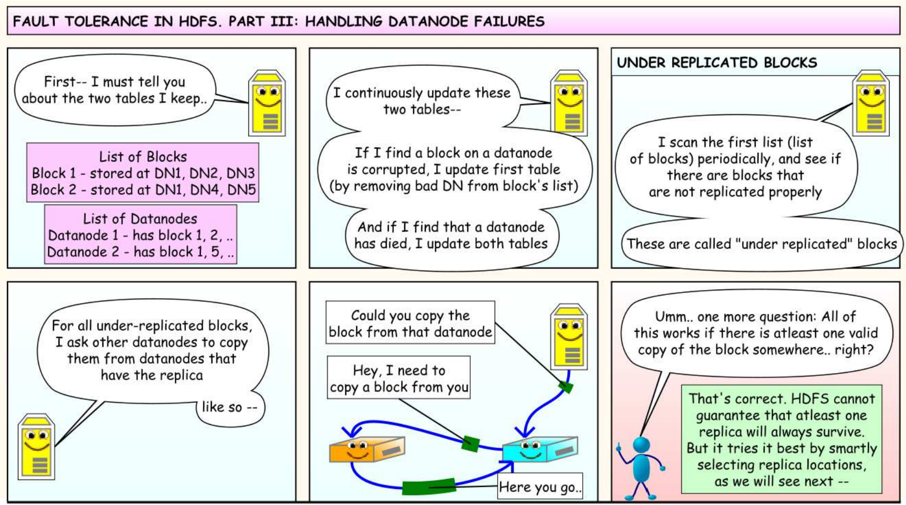
Types of Faults and Their Detection



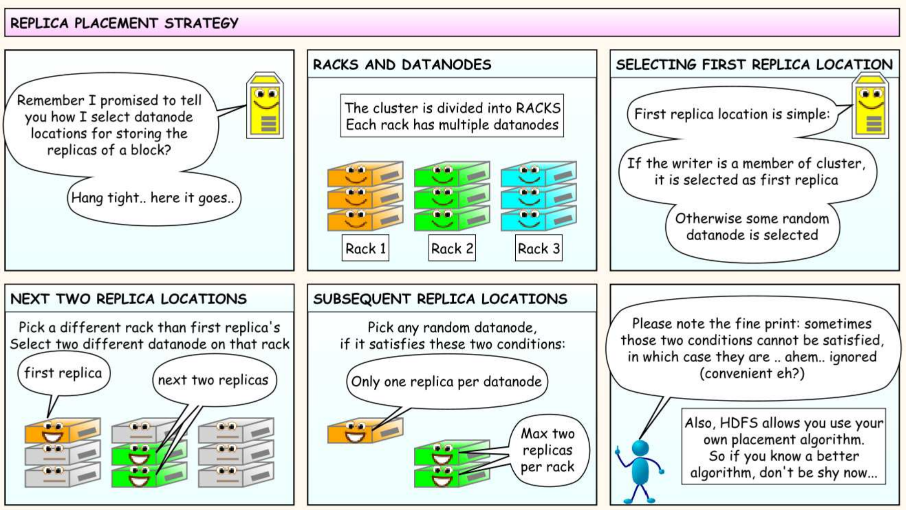
Handling Reading and Writing Failures



Handling DataNode Failures



Replica Placement Strategy



Cluster Rebalancing

- Balanced cluster: no under-utilized or over-utilized DataNodes
- Common reason: addition of new DataNodes

Requirements

- No block loss
- No change of the number of replicas
- No reduction of the number of racks a block resides in
- No saturation of the network

- Rebalancing issued by an administrator
- Rebalancing decisions made by a Rebalancing Server
- Goal of each iteration: reduce imbalance in over/under-utilized DataNodes
- Source and destination selection based on some priorities

Persistence of File System Metadata

To maintain file system namespace:

- `fsimage`: latest checkpoint of the namespace
- `edits`: log of changes to namespace since last checkpoint

- When NameNode starts up, it merges fsimage and edits
- Then, it overwrites fsimage with the new HDFS state and begins a new edits log

Checkpoint Node and Backup Node

CheckPoint Node

- Periodically creates checkpoints of the namespace
- Usually runs on a different machine wrt NameNode
- Multiple CheckPoint Nodes may be used simultaneously

Backup Node

- As Checkpoint Node, but it applies edits to its own namespace copy
- Only one Backup Node at a time, no Checkpoint Nodes allowed
- Work in progress: concurrently use multiple Backup Nodes

Conclusions

GFS

- Proprietary file system
- All the details in the paper of 2003

HDFS

- Open-source project
- Inspired by the Google File System
- Used by many big companies

References

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.
The google file system.
In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

The Apache Software Foundation.
Hadoop distributed file system.
http://hadoop.apache.org/hdfs/, 2011.