The Hadoop Distributed File System

Konstantin Shvachko, Hairong Kuang,
Sanjay Radia, Robert Chansler

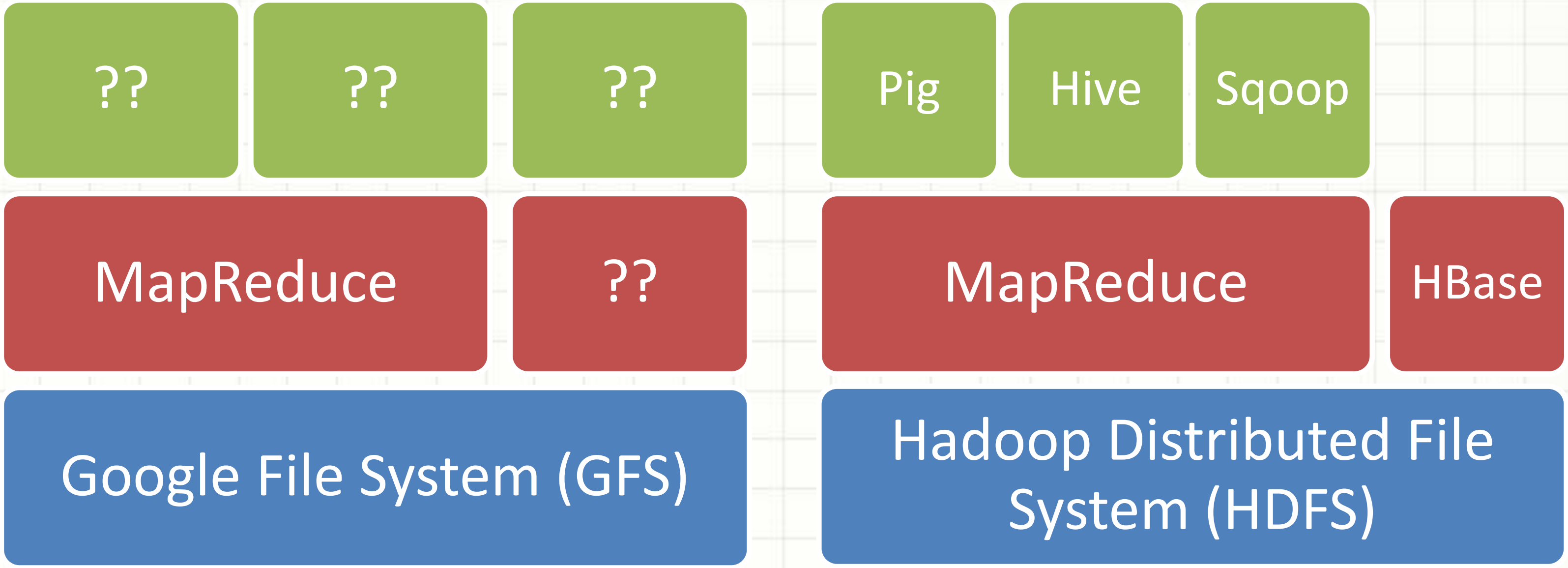
Presented by Alexander Pokluda
October 7, 2013

Outline

- Motivation and Overview of Hadoop
- Architecture, Design & Implementation of the Hadoop Distributed File System (HDFS)
  - Comparison with Google File System (GFS)
- Performance Benchmarks
- Conclusion

Motivation and Overview

Motivation and Overview



- In the early 2000's, Google developed the "Google File System" to support large distributed data-intensive applications
- Shortly after, they developed "MapReduce" to allow developers to easily carry out large scale parallel computations
  - Examples: processing crawled documents, web request logs, etc. to produce inverted indices, statistics, etc.
- **Hadoop** is an open source implementation of Google's proprietary MapReduce framework; **HDFS** is the file system component of Hadoop

Architecture, Design and Implementation

HDFS Architecture

NameNode

Maintains namespace hierarchy and file system metadata such as block locations

Namespace and metadata is stored in RAM but periodically flushed to disk. Modification log keeps on-disk image up to date.

DataNodes

Stores HDFS file data in local file system

Receives commands from *NameNode* that instruct it to:

- Replicate blocks to other nodes
- Remove local block replicas
- Re-register or shutdown
- Send immediate block report

HDFS Client

Code library that exports HDFS file system interface to applications

Reads data by transferring data from a *DataNode* directly

Writes data by setting up a node-to-node pipeline and sends data to the first *DataNode*

Redundancy Mechanisms

Image and Journal

- An image is the file system metadata that describes organization of application data as directories and files
- A persistent record of it written to disk is called a *checkpoint*
- The *journal* is a write-ahead commit log for changes that must be persistent

CheckpointNode and BackupNode

- A NameNode can alternatively be run as a *CheckpointNode* or *BackupNode*
- The *CheckpointNode* periodically combines the existing checkpoint and journal to create a new checkpoint and empty journal
- A *BackupNode* acts like a shadow of the *NameNode* and keeps an up-to-date copy of the image in memory

File I/O Operations and Replica Management

File Read and Write

- An application adds data to HDFS by creating a new file and writing data to it
- All files are read and append only
- HDFS implements a single-writer, multiple-reader model

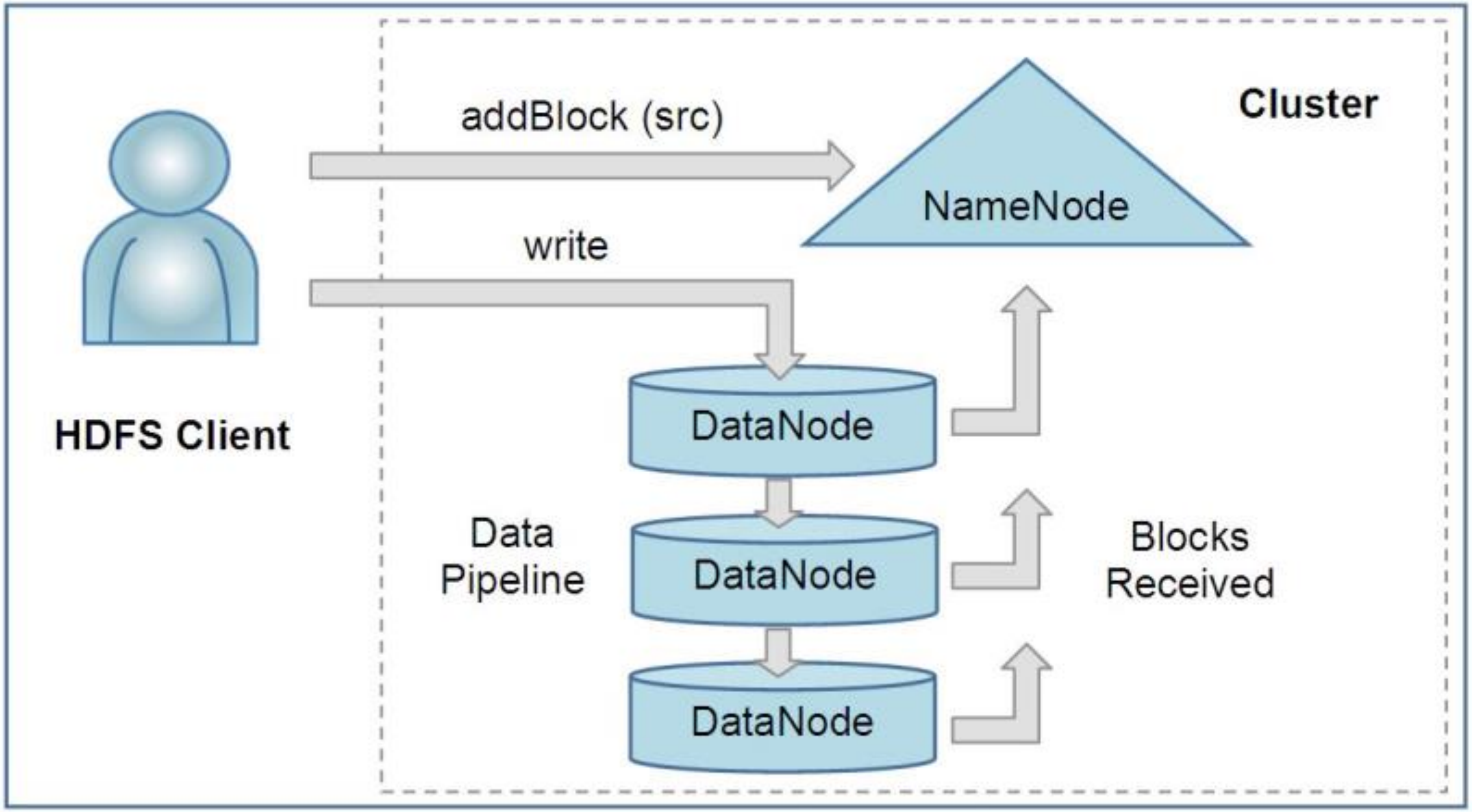
Data Streaming

- When there is need for a new block, the *NameNode* allocates a new block ID and determines a list of *DataNodes* to host replicas of the block
- Data is sent to the *DataNodes* in a pipeline fashion
- Data may not be visible to readers until the file is closed

Block Placement

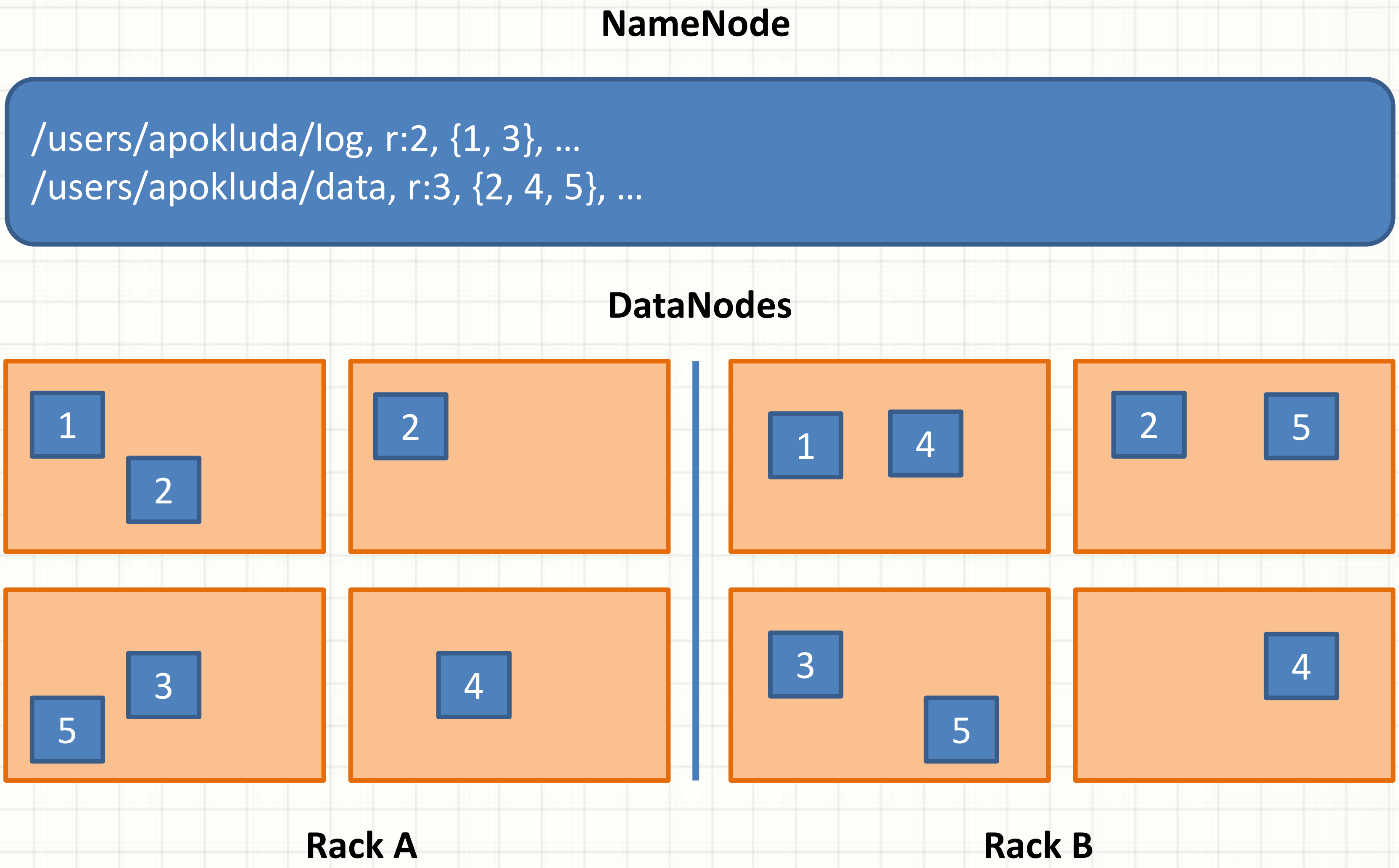
- Default Strategy ensures:
  - No *DataNode* contains more than one replica of any block
  - No rack contains more than two replicas of the same block

File Write Operation



Source: The Hadoop Distributed File System

Data Replication



NameNode

/users/apokluda/log, r:2, {1, 3}, …
/users/apokluda/data, r:3, {2, 4, 5}, …

DataNodes

Hadoop distributed File System vs Google File System

HDFS vs GFS

Implementation

	Hadoop Distributed File System	Google File System
Platform	Cross-platform (Java)	Linux (C/C++)
License	Open source (Apache 2.0)	Proprietary (in-house use only)
Developer(s)	Yahoo! and open source community	Google

Architecture

	Hadoop Distributed File System	Google File System
Architecture Pattern	Single *NameNode* has a global view of the entire file system	
Deployment Hardware	Commodity servers (design to tolerate component failures)	
Inter-Node Communication	*NameNode* uses heartbeats to send commands to *DataNodes*	
DataNode Design	User-level server process stores blocks as files in local file system	

HDFS vs GFS

File System State

	Hadoop Distributed File System	Google File System
File Index State	File index state and mapping of files to blocks kept in memory at *NameNode* and periodically flushed to disk; modification log records changes in between checkpoints	
Block Location State	*NameNode* maintains and persistently stores block location information	Block location information sent to *NameNode* by *DataNodes* on startup; not stored persistently at *NameNode*
Data Integrity	Checksums verified by clients	Checksums verified by *DataNodes*

HDFS vs GFS

File System Operations

	Hadoop Distributed File System	Google File System
Write Operations	• Append only	• Random offset write • Record append • Append
Write Consistency Guarantees	Single-writer model ensures files are always *defined* and *consistent*	• Successful concurrent writes create *consistent* but *undefined* regions • Successful concurrent record appends create *defined* regions interspersed with *inconsistent*
Deletion	Deleted files renamed to a special Trash/Recycling Bin-like folder and removed lazily by garbage collection process	
Snapshots	HDFS 2 allows each directory to have up to 65,536 snapshots	Can snapshot individual files and directories
Block Size	128 MB default but user configurable per file	64 MB default but user configurable per file

HDFS vs GFS

Use Cases

<table>
<tr><th></th><th>Hadoop Distributed File System</th><th>Google File System</th></tr>
<tr><td><b>Primary Use</b></td><td colspan="2">General purpose (production services, R&D) and MapReduce jobs</td></tr>
<tr><td><b>Data Access Pattern</b></td><td colspan="2">Random access reads supported but optimized for streaming</td></tr>
<tr><td><b>File Size</b></td><td colspan="2">Optimized for Large Files</td></tr>
<tr><td><b>Replication</b></td><td colspan="2">User configurable per file, but 3 replicas stored by default</td></tr>
<tr><td><b>Client API</b></td><td colspan="2">Custom library and command line utilities</td></tr>
</table>

Performance Benchmarks

Performance Benchmarks

DFSIO

- Read: 66 MB/s per node
- Write: 40 MB/s per node

Production Cluster

- Read: 1.02 MB/s per node
- Write: 1.09 MB/s per node

Sort

- 1 TB sort
    - 22.1 MB/s per node (RW)
- 1 PB sort
    - 9.35 MB/s per node (RW)

Operation	Throughput (Ops/s)
Open File for Read	126,100
Create File	5600
Rename File	8300
Delete File	20,700
DataNode Heartbeat	300,000
Blocks Report (blocks/s)	639,700

CONCLUSION

Conclusion

- The **Hadoop Distributed File System** is designed to store very large data sets reliably and to stream these datasets to user applications at high bandwidth
- The **Hadoop MapReduce framework** is designed to distribute storage and computation tasks across thousands of servers to enable resources to scale with demand while maintaining economical in size
- The **HDFS architecture** consists of a single *NameNode*, many *DataNodes* and the *HDFS client*
- **Hadoop** is an open source project that was inspired by Google's proprietary *Google File System* and *MapReduce framework*

Discussion