The Hadoop Distributed File System

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler
Yahoo!
Sunnyvale, California USA
{Shv, Hairong, SRadia, Chansler}@Yahoo-Inc.com

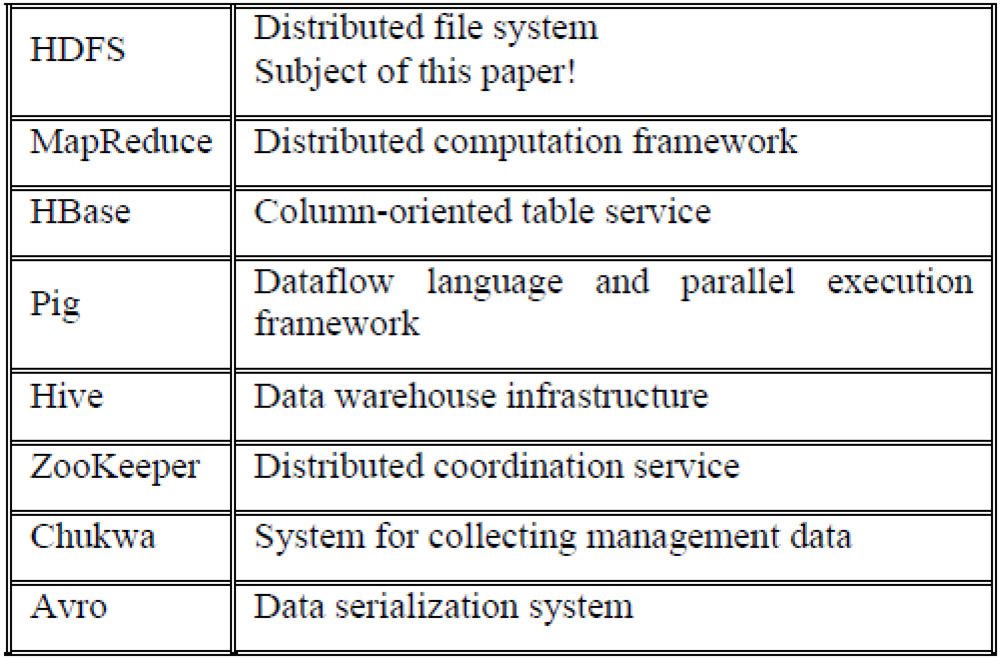
Presenter: Alex Hu

HDFS

- Introduction
- Architecture
- File I/O Operations and Replica Management
- Practice at YAHOO!
- Future Work
- Critiques and Discussion

Introduction and Related Work

- What is *Hadoop*?
  - Provide a distributed file system and a framework
  - Analysis and transformation of very large data set
  - *MapReduce*



HDFS	Distributed file system Subject of this paper!
MapReduce	Distributed computation framework
HBase	Column-oriented table service
Pig	Dataflow language and parallel execution framework
Hive	Data warehouse infrastructure
ZooKeeper	Distributed coordination service
Chukwa	System for collecting management data
Avro	Data serialization system

Table 1. Hadoop project components

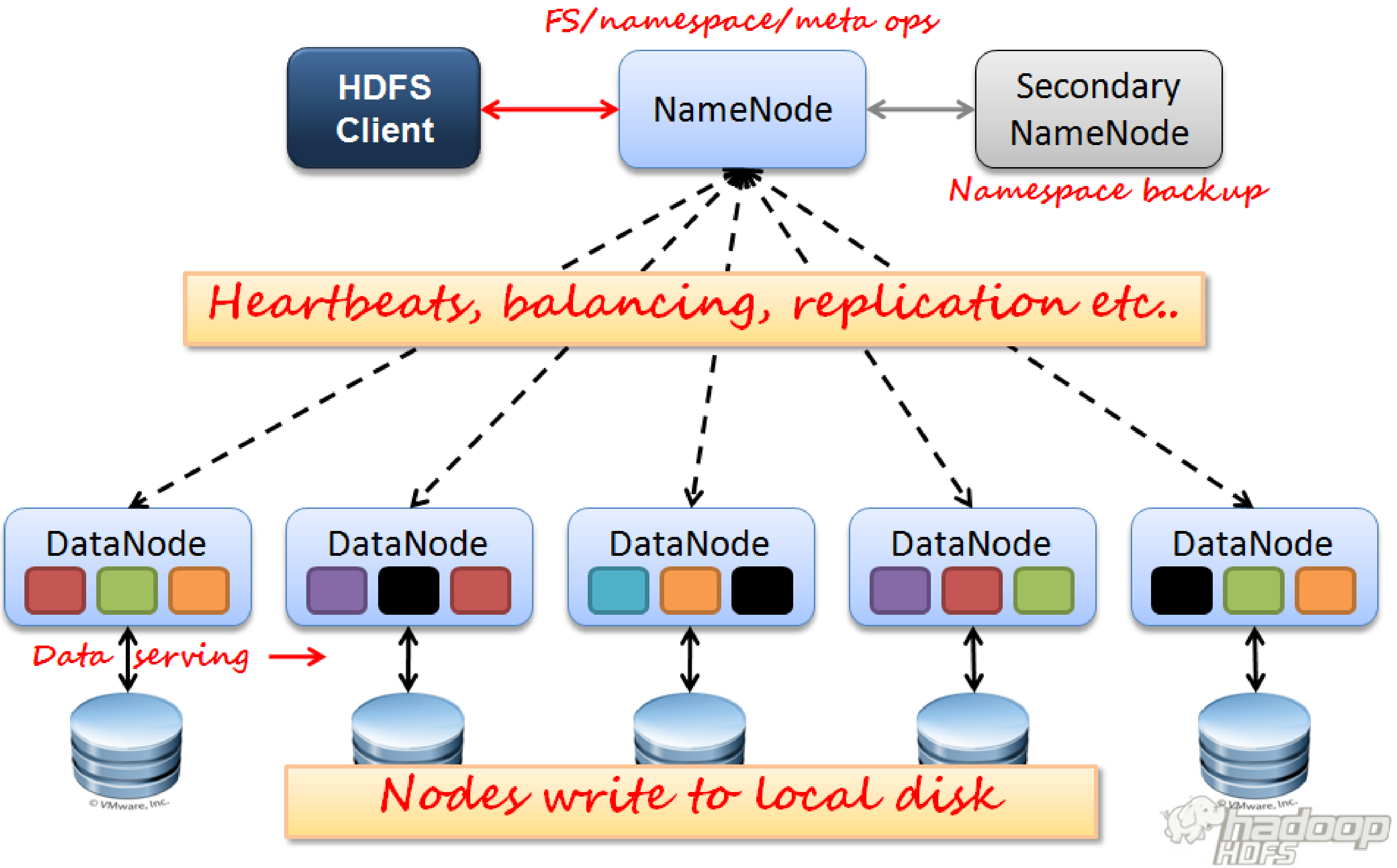
Introduction (cont.)

- What is *Hadoop Distributed File System (HDFS)* ?
  - File system component of Hadoop
  - Store metadata on a dedicated server *NameNode*
  - Store application data on other servers *DataNode*
  - TCP-based protocols
  - Replication for reliability
  - Multiply data transfer bandwidth for durability

Architecture

- NameNode
- DataNodes
- HDFS Client
- Image Journal
- CheckpointNode
- BackupNode
- Upgrade, File System Snapshots

Architecture Overview



FS/namespace/meta ops

HDFS Client

NameNode

Secondary NameNode

Namespace backup

Heartbeats, balancing, replication etc..

DataNode

DataNode

DataNode

DataNode

DataNode

Data serving

Nodes write to local disk

© VMware, Inc.

© VMware, Inc.

hadoop HDFS

NameNode – one per cluster

- **Maintain** ***The HDFS namespace***, a hierarchy of files and directories represented by *inodes*
- **Maintain the mapping of file blocks to DataNodes**
  - Read: ask NameNode for the location
  - Write: ask NameNode to nominate DataNodes
- ***Image and Journal***
- ***Checkpoint***: native files store persistent record of images (no location)

DataNodes

- Two files to represent a block replica on DN
  - The data itself – length flexible
  - *Checksums* and *generation stamp*
- ***Handshake*** when connect to the NameNode
  - Verify *namespace ID* and *software version*
  - New DN can get one namespace ID when join
- ***Register*** with NameNode
  - *Storage ID* is assigned and never changes
  - *Storage ID* is a unique internal identifier

DataNodes (cont.) - control

- ***Block report***: identify block replicas
  - *Block ID*, the *generation stamp*, and the length
  - Send first when register and then send per hour
- ***Heartbeats***: message to indicate availability
  - Default interval is three seconds
  - DN is considered “dead” if not received in 10 mins
  - Contains Information for space allocation and load balancing
    - Storage capacity
    - Fraction of storage in use
    - Number of data transfers currently in progress
  - NN replies with instructions to the DN
  - Keep frequent. Scalability

HDFS Client

- A code library exports HDFS interface
- Read a file
  - Ask for a list of DN host replicas of the blocks
  - Contact a DN directly and request transfer
- Write a file
  - Ask NN to choose DNs to host replicas of the first block of the file
  - Organize a pipeline and send the data
  - Iteration
- Delete a file and create/delete directory
- Various APIs
  - Schedule tasks to where the data are located
  - Set replication factor (number of replicas)

HDFS Client (cont.)

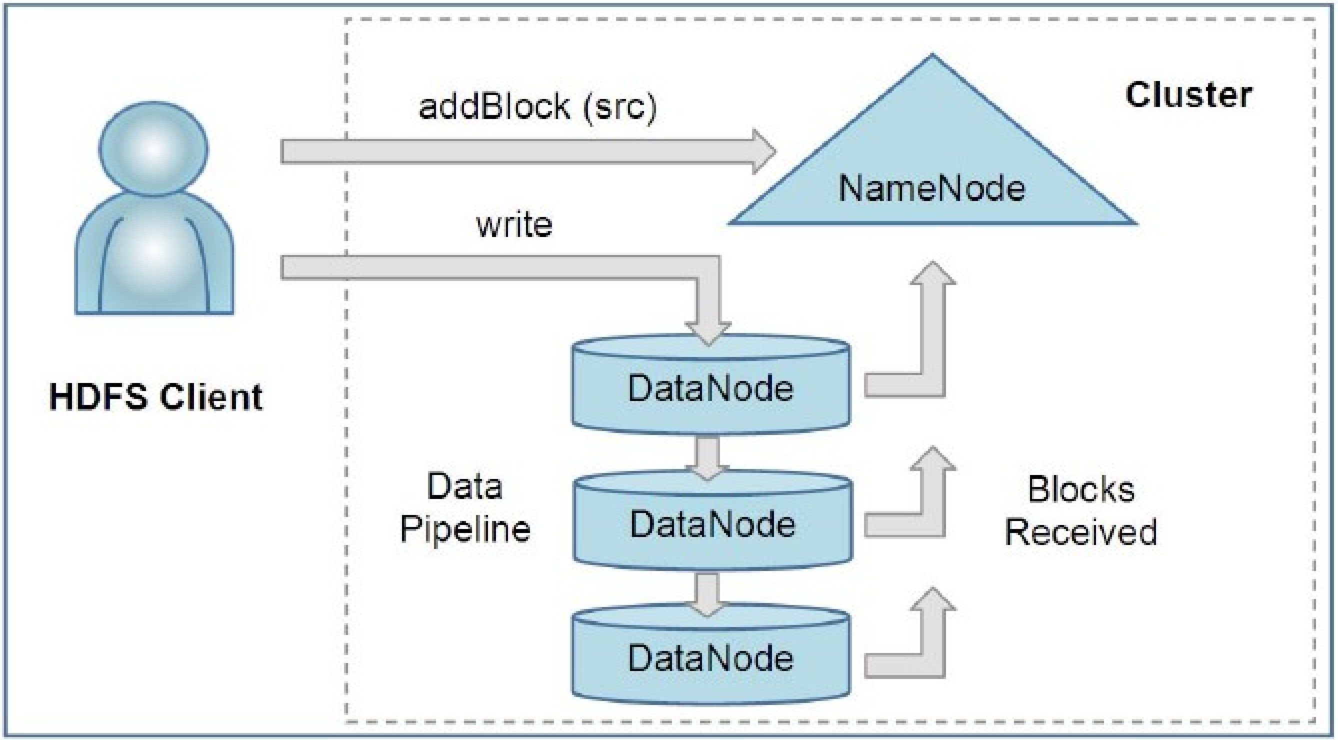


Image and Journal

- *Image*: metadata describe organization
  - Persistent record is called *checkpoint*
  - *Checkpoint* is never changed, and can be replaced
- *Journal*: log for persistence changes
  - Flushed and synched before change is committed
- **Store in multiple places to prevent missing**
  - NN shut down if no place is available
- **Bottleneck**: threads wait for flush-and-sync
  - Solution: batch

CheckpointNode

- ***CheckpointNode* is NameNode**
- **Runs on different host**
- **Create new checkpoint**
  - Download current checkpoint and journal
  - Merge
  - Create new and return to NameNode
  - NameNode truncate the tail of the journal
- **Challenge**: large journal makes restart slow
  - Solution: create a daily checkpoint

BackupNode

- Recent feature
- Similar to CheckpointNode
- Maintain an in memory, up-to-date image
  - Create checkpoint without downloading
- Journal store
- Read-only NameNode
  - All metadata information except block locations
  - No modification

Upgrades, File System and Snapshots

- Minimize damage to data during upgrade
- Only one can exist
- NameNode
  - Merge current checkpoint and journal in memory
  - Create new checkpoint and journal in a new place
  - Instruct DataNodes to create a local snapshot
- DataNode
  - Create a copy of storage directory
  - Hard link existing block files

Upgrades, File System and Snapshots – Rollback

- NameNode recovers the checkpoint

- DataNode resotres directory and delete replicas after snapshot is created

- The *layout version* stored on both NN and DN
  - Identify the data representation formats
  - Prevent inconsistent format

- Snapshot creation is all-cluster effort
  - Prevent data loss

File I/O Operations and Replica Management

- File Read and Write
- Block Placement and Replication management
- Other features

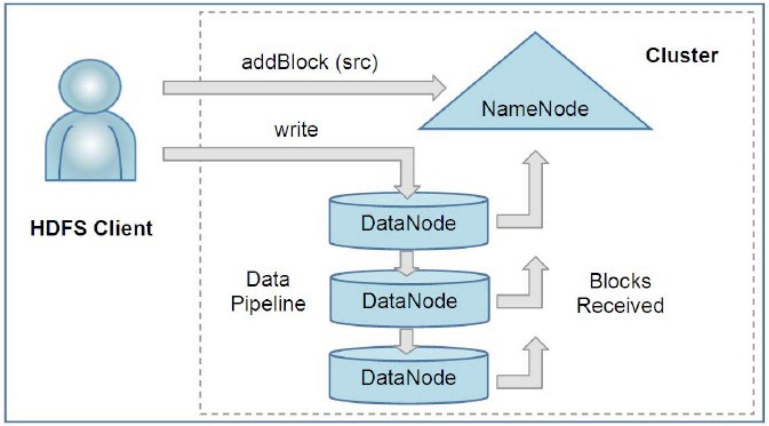
File Read and Write

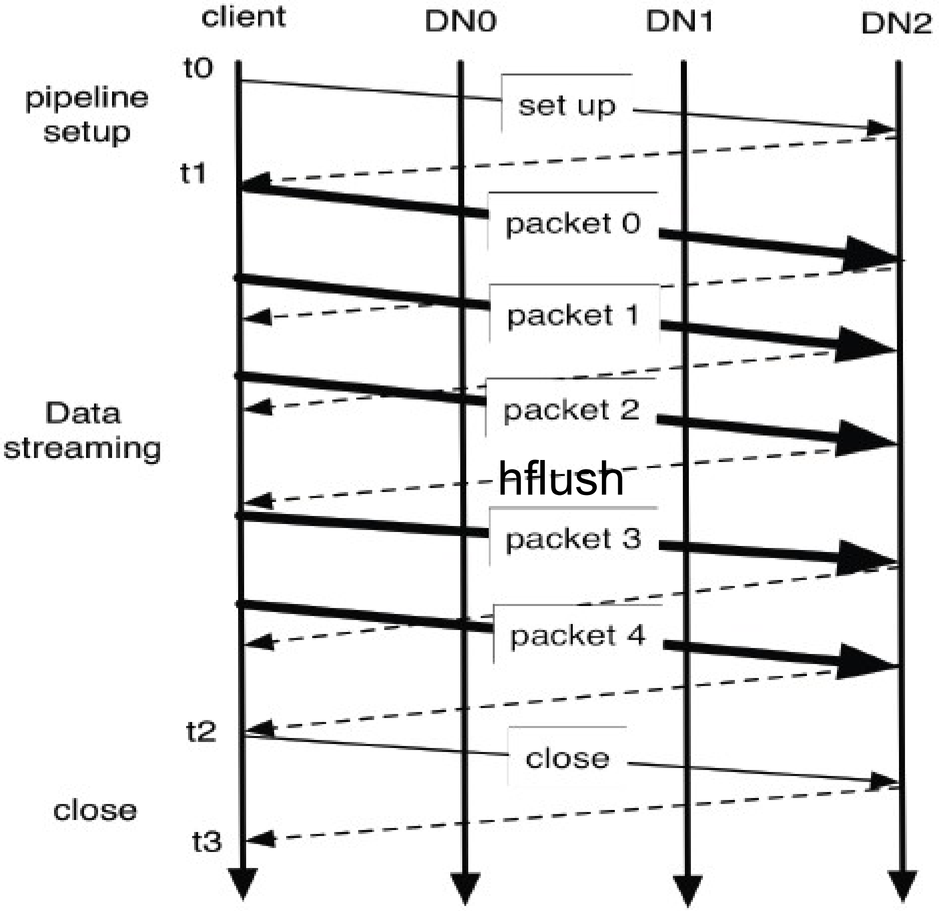
- **Checksum**
  - Read by the HDFS client to detect any corruption
  - DataNode store checksum in a separate place
  - Ship to client when perform HDFS read
  - Clients verify checksum
- **Choose the closet replica to read**
- **Read fail due to**
  - Unavailable DataNode
  - A replica of the block is no longer hosted
  - Replica is corrupted
- **Read while writing**: ask for the latest length

File Read and Write (cont.)

- New data can only be appended
- *Single-writer, multiple-reader*
- **Lease**
  - Who open a file for writing is granted a lease
  - Renewed by heartbeats and revoked when closed
  - Soft limit and hard limit
  - Many readers are allowed to read
- Optimized for sequential reads and writes
  - Can be improved
    - Scribe: provide real-time data streaming
    - Hbase: provide random, real-time access to large tables

Add Block and The *hflush*

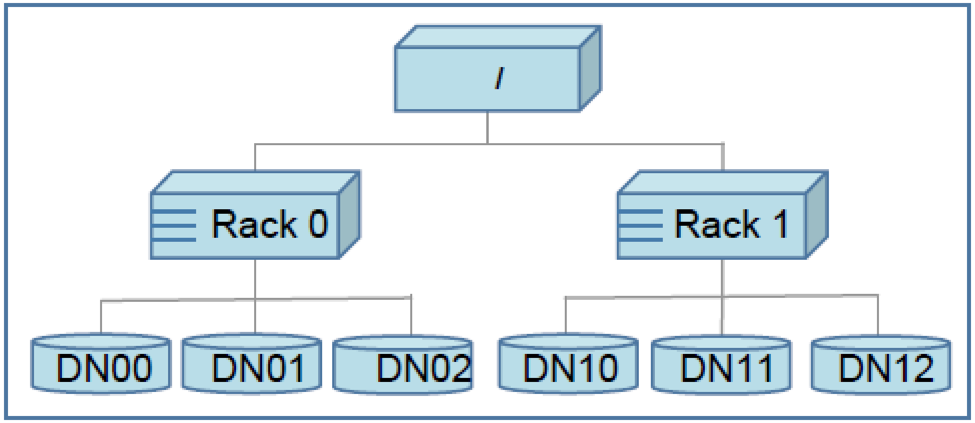




- Unique block ID
- Perform write operation
- new change is not guaranteed to be visible
- The *hflush*

Block Replacement

- Not practical to connect all nodes
- Spread across multiple racks
  - Communication has to go through multiple switches
  - Inter-rack and intra-rack
  - Shorter distance, greater bandwidth
- NameNode decide
  - Configure script



Replica Replacement Policy

- Improve data reliability, availability and network bandwidth utilization
- Minimize write cost
- Reduce inter-rack and inter-node write
- *Rule1: No Datanode contains more than one replica of any block*
- *Rule2: No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster*

Replication management

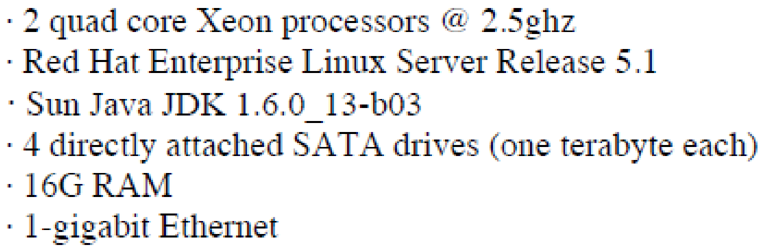
- Detected by NameNode
- Under-replicated
  - Priority queue (node with one replica has the highest)
  - Similar to replication replacement policy
- Over-replicated
  - Remove the old replica
  - Not reduce the number of racks

Other features

- Balancer
  - Balance disk space usage
  - Bandwidth consuming control
- Block Scanner
  - Verification of the replica
  - Corrupted replica is not deleted immediately
- Decommissioning
  - Include and exclude lists
  - Re-evaluate lists
  - Remove decommissioning DataNode only if all blocks on it are replicated
- Inter-Cluster Data Copy
  - DistCp – MapReduce job

Practice At Yahoo!

- 3500 nodes and 9.8PB of storage available
- Durability of Data
  - Uncorrelated node failures
    - Chance of losing a block during one year: <.5%
    - Chance of node fail each month: .8%
  - Correlated node failures
    - Failure of rack or switch
    - Loss of electrical power
- Caring for the commons
  - Permissions – modeled on UNIX
  - Total space available



· 2 quad core Xeon processors @ 2.5ghz
· Red Hat Enterprise Linux Server Release 5.1
· Sun Java JDK 1.6.0_13-b03
· 4 directly attached SATA drives (one terabyte each)
· 16G RAM
· 1-gigabit Ethernet

Benchmarks

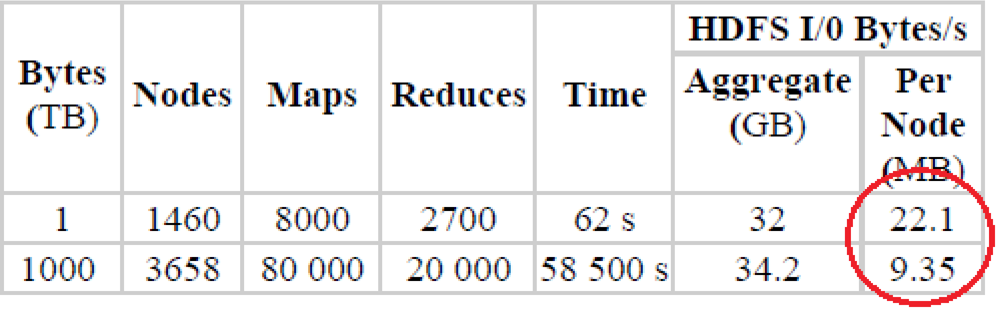
DFSIO benchmark

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

Production cluster

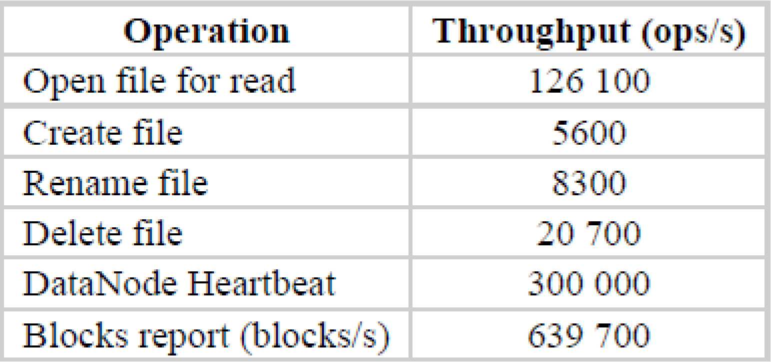
- Busy Cluster Read: 1.02MB/s per node
- Busy Cluster Write: 1.09MB/s per node

Sort benchmark



Bytes (TB)	Nodes	Maps	Reduces	Time	HDFS I/0 Bytes/s	
					Aggregate (GB)	Per Node (MB)
1	1460	8000	2700	62 s	32	22.1
1000	3658	80 000	20 000	58 500 s	34.2	9.35

Operation Benchmark



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

Future Work

- Automated failover solution
  - Zookeeper
- Scalability
  - Multiple namespaces to share physical storage
  - Advantage
    - Isolate namespaces
    - Improve overall availability
    - Generalizes the block storage abstraction
  - Drawback
    - Cost of management
  - Job-centric namespaces rather than cluster centric

Critiques and Discussion

- Pros
  - **Architecture**: NameNode, DataNode, and powerful features to provide kinds of operations, detect corrupted replica, balance disk space usage and provide consistency.
  - **HDFS is easy to use**: users don't have to worry about different servers. It can be used as local file system to provide various operations
  - **Benchmarks are sufficient**. They use real data with large number of nodes and storage to provide kinds of experiments.
- Cons
  - **Fault—tolerance** is not very sophisticated. All the recoveries introduced are based on the assumption that NameNode is alive. No proper solution currently in this paper handles the failure of NameNode
  - **Scalability**, especially the handling of replying heartbeats with instructions. If there are too many messages come in, the performance of NameNode is not proper measured in this paper
  - **The test of correlated failure** is not provided. We can't get any information of the performance of HDFS after correlated failure is encountered.

- Thank you very much