Aalto University
School of Science
and Technology

The Hadoop Distributed File System

Tero Laitinen

Department of Information and Computer Science
Aalto University
tero.laitinen@tkk.fi

November 17, 2010

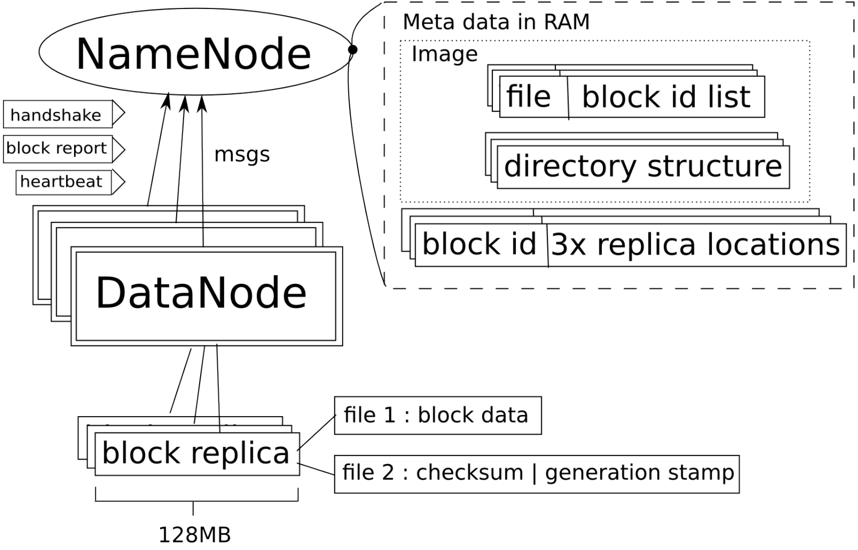
Outline

1. HDFS architecture
2. benchmarks by Yahoo!
3. HDFS bottleneck in MapReduce use?

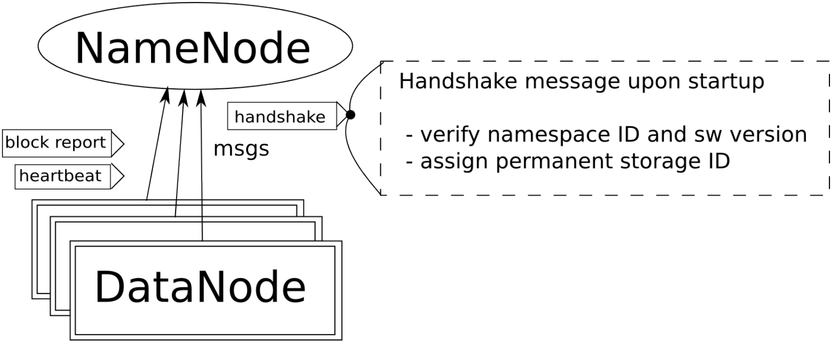
Hadoop Distributed File System

- component of Hadoop Apache project
- large-scale fault-tolerant distributed file system
- inspired by GFS
- portable Java implementation
- mostly by Yahoo!
- used by 100+ organizations
- HDFS at Yahoo!
  - 25k servers
  - 25 petabytes
  - largest cluster 3,5k servers

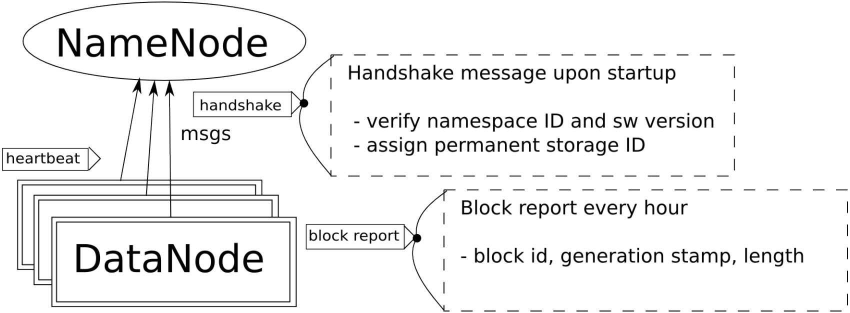
Architecture



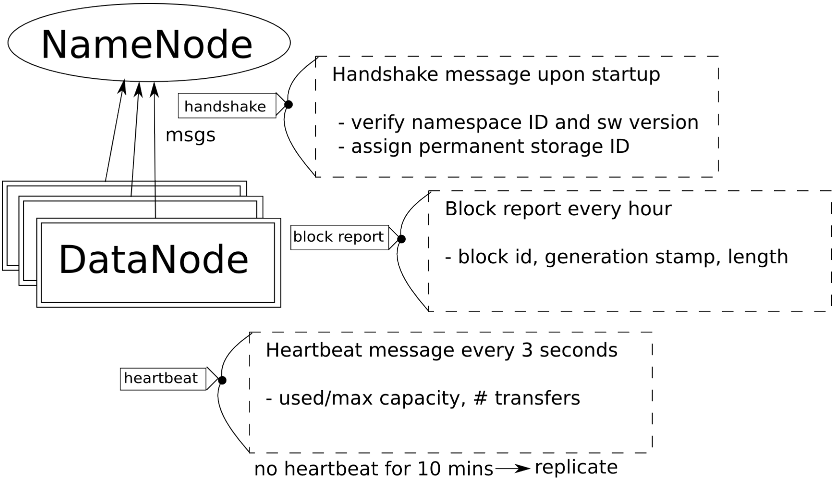
Architecture : handshake



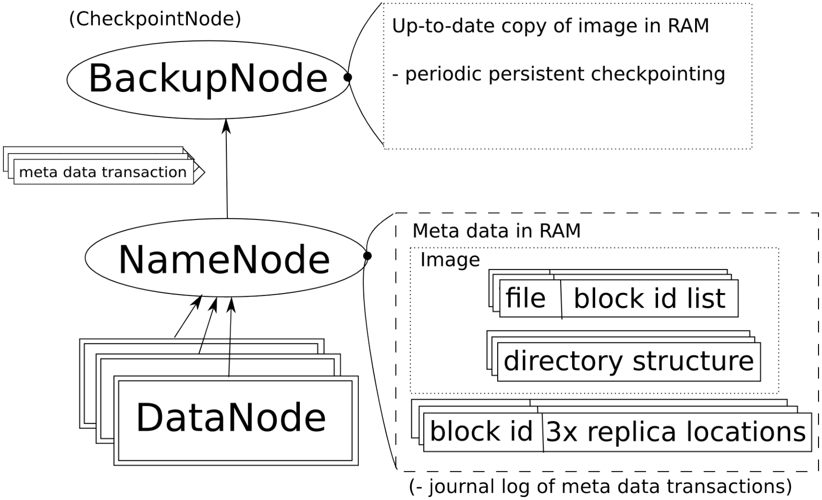
Architecture : block report



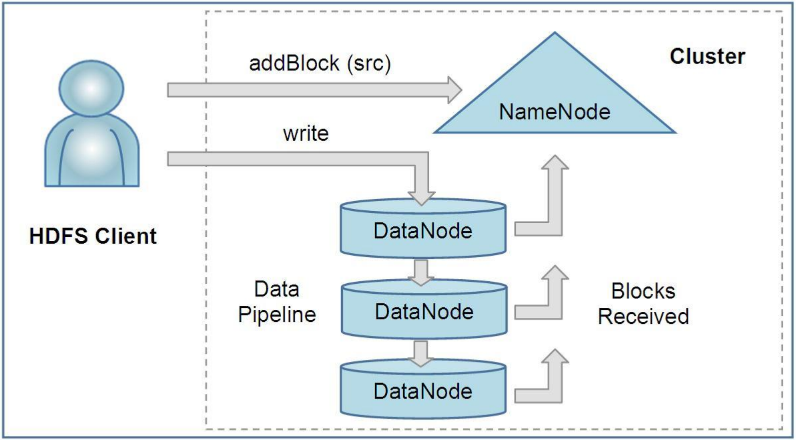
Architecture : heartbeat



CheckpointNode, BackupNode



HDFS client


addBlock (src)
Cluster
NameNode
write
HDFS Client
DataNode
Data
Pipeline
DataNode
Blocks
Received
DataNode

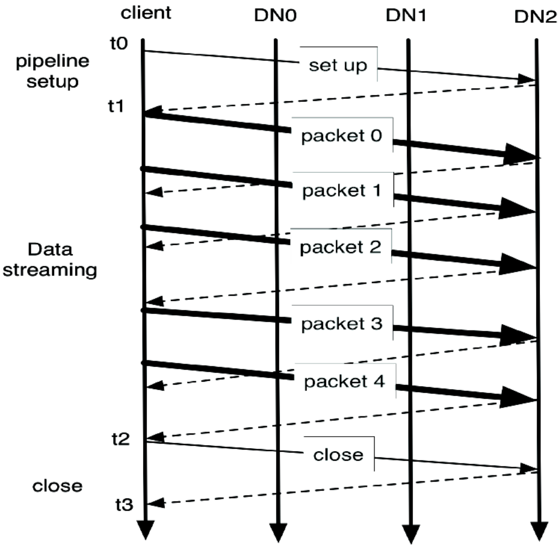
Upgrades, Snapshots

- snapshot = a state of the entire file system
- only one snapshot can exist
- created to minimize damage during upgrades upon startup
- hard links + copy-on-write
- storage directories contain layout version for automatic data conversion

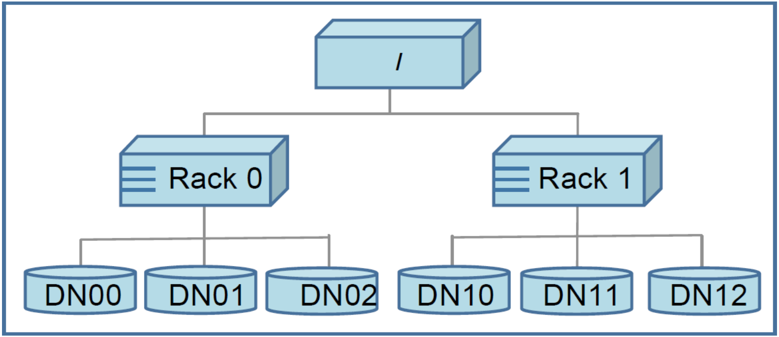
File I/O

- files are append-only, existing data cannot be changed
- writer is granted a lease (renewed using heartbeats)
- lease soft limit (can be pre-empted after), hard limit 1h

Data Pipeline



Cluster topology and data replication



1. No DataNode contains more than one replica of any block.
2. No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster.

Balancer

- disk usage not taken into account in replica placement
- balancer moves replicas if disk usage ratio of DataNode differs from average too much
- max bandwidth usage can be set

Block Scanner

- DataNode's block scanner verifies checksums of blocks
- corrupt replicas are deleted when enough good replicas

Other features

- decommissioning of DataNodes (safe removal)
- mirroring HDFS cluster as a MapReduce job
- permission framework (owner, group, read/read-write)
- per-directory / per-subtree quota

A benchmark

- DFSIO read 64 MB/s per node
- DFSIO write 40 MB/s per node
- sort competition at Yahoo!

Bytes(TB)	Nodes	Time	Replication	Node (MB/s)
1	1460	62 s	1x	22.1
1000	3658	58 500 s	2x	9.35

NameNode 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

- Note: without RPC overhead

Known issues in HDFS

- NameNode is single point of failure
- scalability of NameNode (Java GC), multiple NameNodes

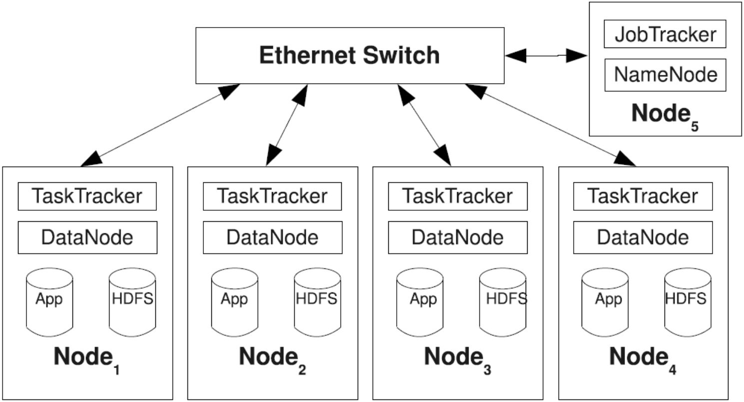
HDFS is bottleneck for MapReduce

- efficiency of (Hadoop) MapReduce paradigm questioned
- parallel databases shown better on some benchmarks
- HDFS is a performance bottleneck for MapReduce

HDFS + MapReduce Node hardware

- 2-processor Opteron server 2.4Ghz 4GB RAM
- gigabit Ethernet
- FreeBSD 7.2, Hadoop 0.20.0, Java 1.6.0
- 2x Seagate Barracude 7200.11 500GB
- UFS2 filesystem, 16kB block size
- no HDFS replication

HDFS + MapReduce Cluster setup



Baseline for I/O speed comparison

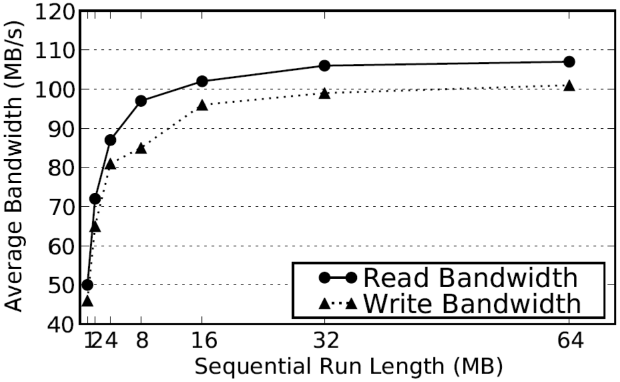


Figure: async I/O test written in C on the raw disk, seek every *n* MB

Software architectural bottleneck : scheduler

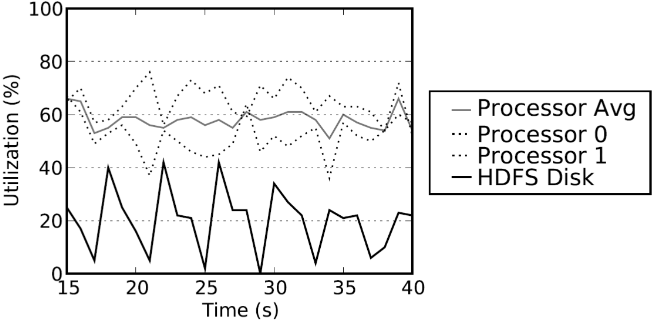


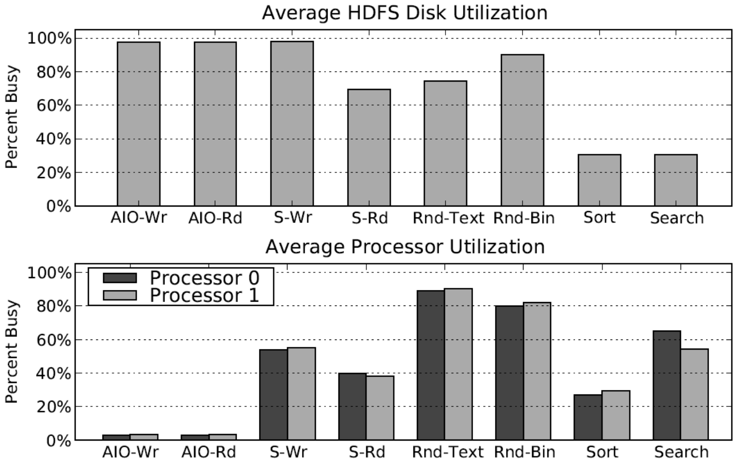
Figure: % of time disk had I/O request (simple search on 40GB)

- causes: i) task / block ii) JVM / task

Software architectural bottleneck : scheduler

- quick fixes:
  - increase block size
  - re-use JVM
- better solution?
  - block and task prefetching
  - multiple tasks / node

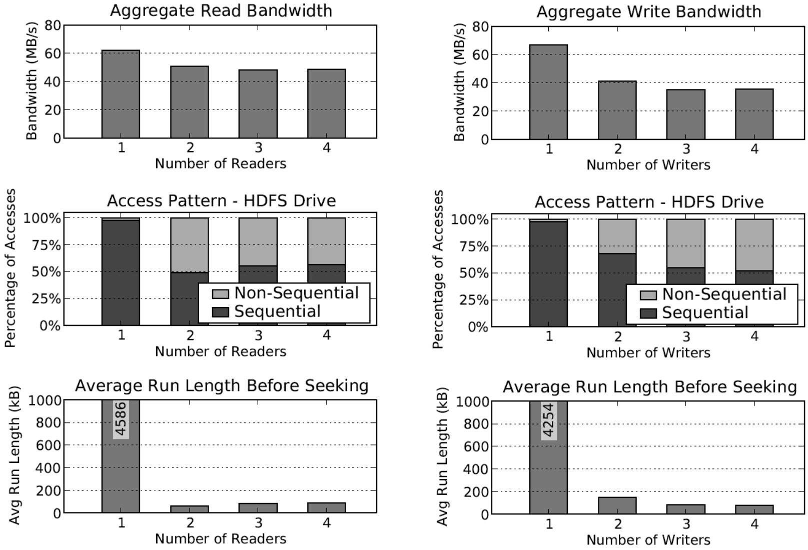
Portability limitations



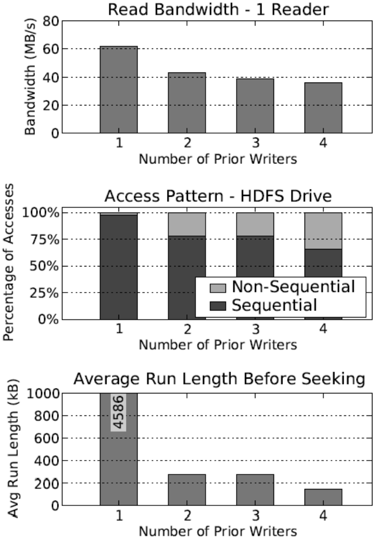
Portability limitations : filesystem and caching

Metric	Read		Write	
	Raw	**Filesystem**	**Raw**	**Filesystem**
Bandwidth (MB/s)	99.9	98.4	98.1	94.9
Processor (total)	7.4%	13.8%	6.0%	15.6%
Processor (FS+cache)	N/A	4.4%	N/A	7.2%

Portability assumptions : I/O scheduling



Portability assumptions : fs fragmentation



Effect of file system

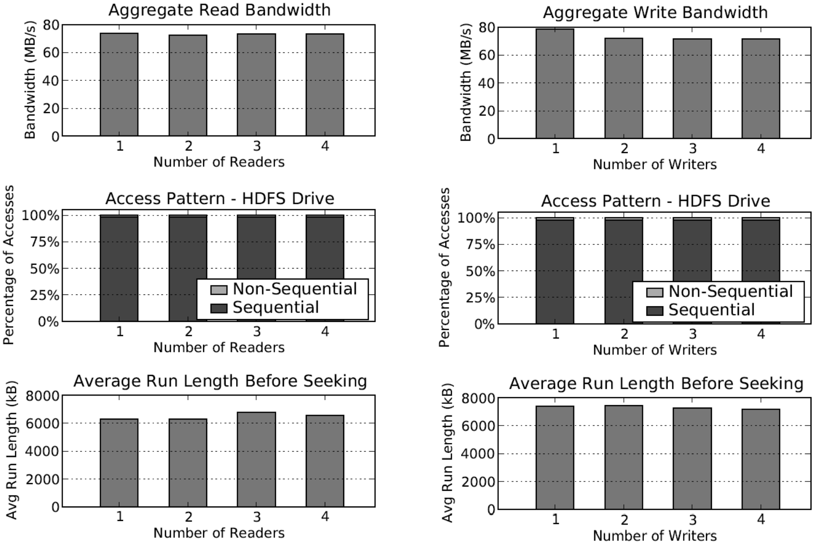
File System	Degradation	
	Read	Write
UFS2	21%	47%
ext4	42%	8%
XFS	43%	0%

Figure: Degradation between 1 and 4 readers/writers

Solution : application-level disk scheduling

- old implementation:
  - one thread / HDFS client
- new implementation:
  - one I/O thread / disk
  - one thread / HDFS client

Solution : application-level disk scheduling



Solution : application-level disk scheduling

- sequential run length 6-8 MB
- ideally 32MB+ $\rightsquigarrow$ non-portable techniques

Non-portable techniques

- OS hints
  - file pre-allocation
- File system selection
  - extents as a requirement
- Cache bypass
  - DMA-transfer from disk to user space
- Raw disk usage
  - 1 seek / HDFS block, 64MB sequential run length

Summary

- Hadoop Distributed File System (HDFS) is a fault-tolerant large-scale file system inspired by GFS.
- The main application is MapReduce.
- HDFS widely in production use.
- Portable Java hinders effective hard disk usage in typical MapReduce use.
- Application-level I/O scheduling circumvents some issues.
- Further speedups possible by non-portable solutions.

Thank you for your attention

Questions?

References

- Jeffrey Shafer, Scott Rixner, and Alan L. Cox. The Hadoop distributed filesystem: Balancing portability and performance. In IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2010), pages 122-133, 2010
- Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST2010), pages 1-10, 2010