Big Data Analytics

B. Distributed Storage / B.1 Distributed File Systems

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue.	9.4.	(1)	0. Introduction
			A. Parallel Computing
Tue.	16.4.	(2)	A.1 Threads
Tue.	23.4.	(3)	A.2 Message Passing Interface (MPI)
Tue.	30.4.	(4)	A.3 Graphical Processing Units (GPUs)
			B. Distributed Storage
Tue.	7.5.	(5)	B.1 Distributed File Systems
Tue.	14.5.	(6)	B.2 Partioning of Relational Databases
Tue.	21.5.	(7)	B.3 NoSQL Databases
			C. Distributed Computing Environments
Tue.	28.5.	(8)	C.1 Map-Reduce
Tue.	4.6.	—	— *Pentecoste Break* —
Tue.	11.6.	(9)	C.2 Resilient Distributed Datasets (Spark)
Tue.	18.6.	(10)	C.3 Computational Graphs (TensorFlow)
			D. Distributed Machine Learning Algorithms
Tue.	25.6.	(11)	D.1 Distributed Stochastic Gradient Descent
Tue.	2.7.	(12)	D.2 Distributed Matrix Factorization
Tue.	9.7.	(13)	Questions and Answers

Outline

1. Why do we need a Distributed File System?

2. What is a Distributed File System?

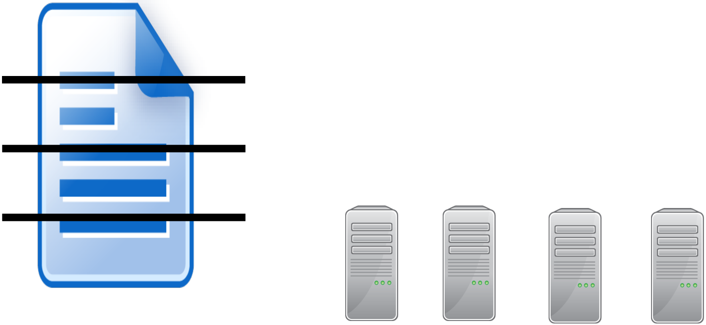
3. GFS and HDFS

4. Hadoop Distributed File System (HDFS)

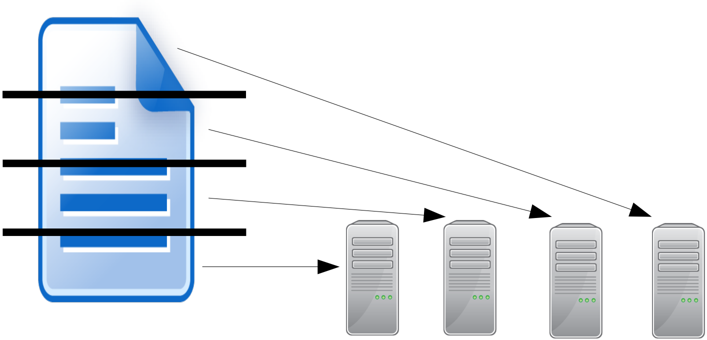
Outline

Why do we need a Distributed File System?

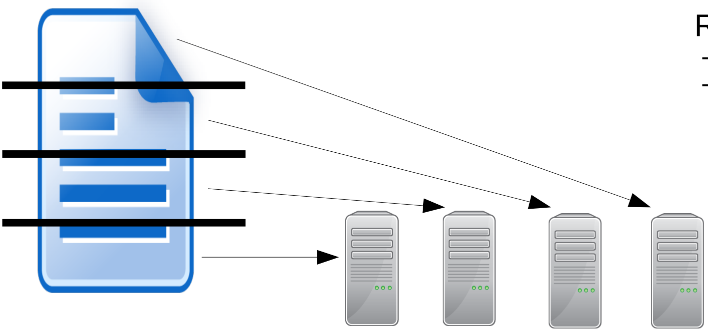
Why do we need a Distributed File System?



Why do we need a Distributed File System?

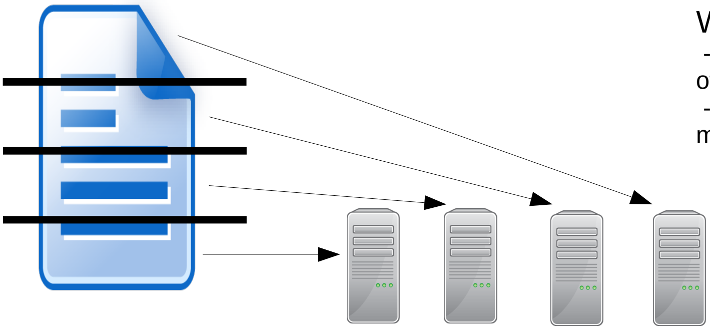


Why do we need a Distributed File System?



Read???
- Whole File?
- Specific part?

Why do we need a Distributed File System?



Write???
- Append to the end of the file?
- Insert content in the middle?

Why do we need a Distributed File System?

We want to:

- Read large data fast
  - **scalability**: perform multiple **parallel reads and writes**

Why do we need a Distributed File System?

We want to:

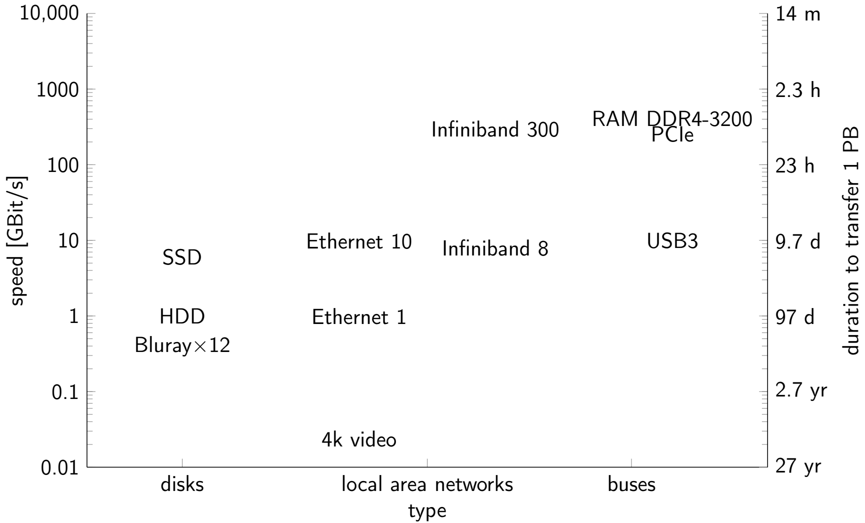
- Read large data fast
  - **scalability**: perform multiple **parallel reads and writes**
- Have the files available even if one computer crashes
  - **fault tolerance**: **replication**

Why do we need a Distributed File System?

We want to:

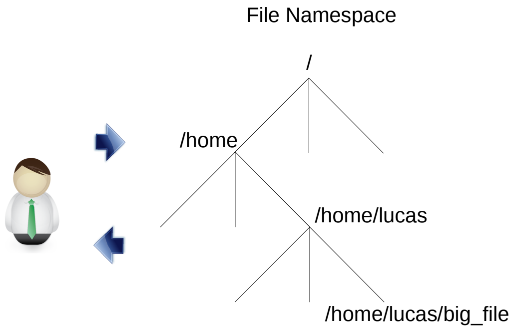
- Read large data fast
  - **scalability**: perform multiple **parallel reads and writes**
- Have the files available even if one computer crashes
  - **fault tolerance**: **replication**
- Hide parallelization and distribution details
  - **transparency**: clients can access it like a local filesystem

Data Transfer Rates

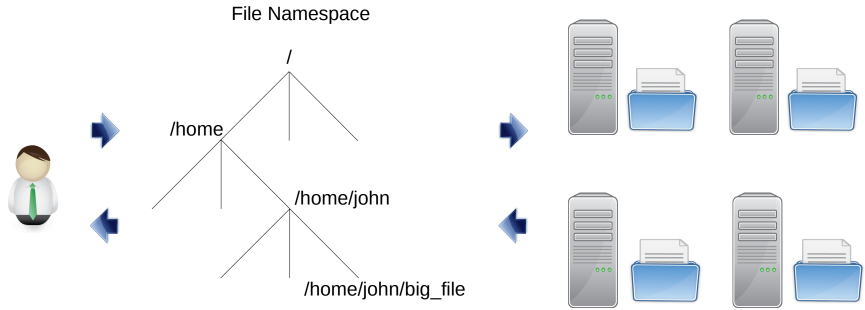


Outline

What is a Distributed File System?



What is a Distributed File System?



Examples

- Windows Distributed File System (DFS; Microsoft, 1996)
- GFS (Google, 2003)
- Lustre (Cluster File Systems, 2003)
- BeeGFS (Fraunhofer, 2005)
- HDFS (Apache Software Foundation, 2006)
- GlusterFS (Red Hat, 2007)
- Ceph (Inktank/Red Hat, 2007)
- MooseFS (Core Technology/Gemius, 2008)
- MapR File System (MapR Technologies, 2010)

Components

A typical distributed filesystem contains the following components

- Clients - they interface with the user

Components

A typical distributed filesystem contains the following components

- Clients - they interface with the user
- Chunk nodes - stores chunks of files

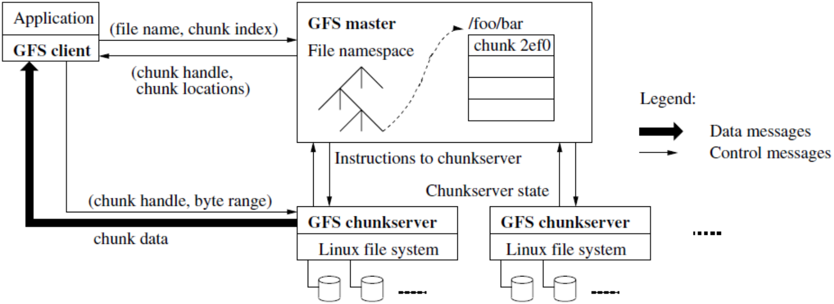
Components

A typical distributed filesystem contains the following components

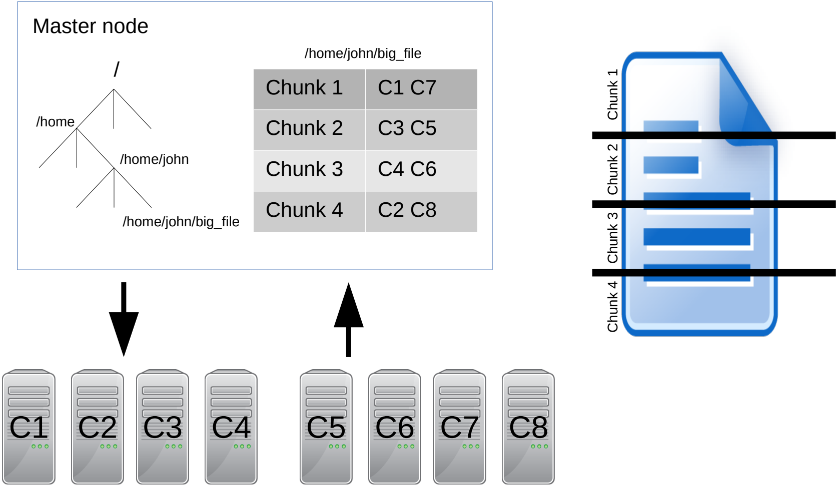
- Clients - they interface with the user
- Chunk nodes - stores chunks of files
- Master node - stores which parts of each file are on which chunk node

Distributed File Systems

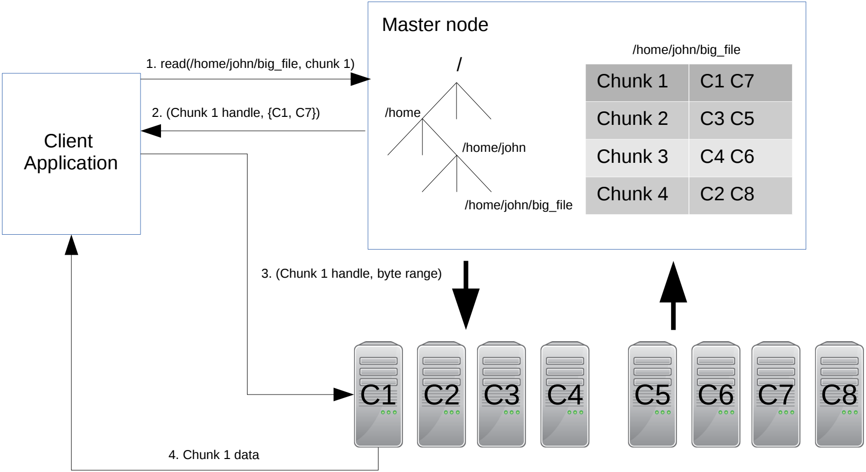
The Google File System Architecture



Distributed File Systems - Storing files



Read Example



Write Example

- Make sure each replica contains the same data all the time

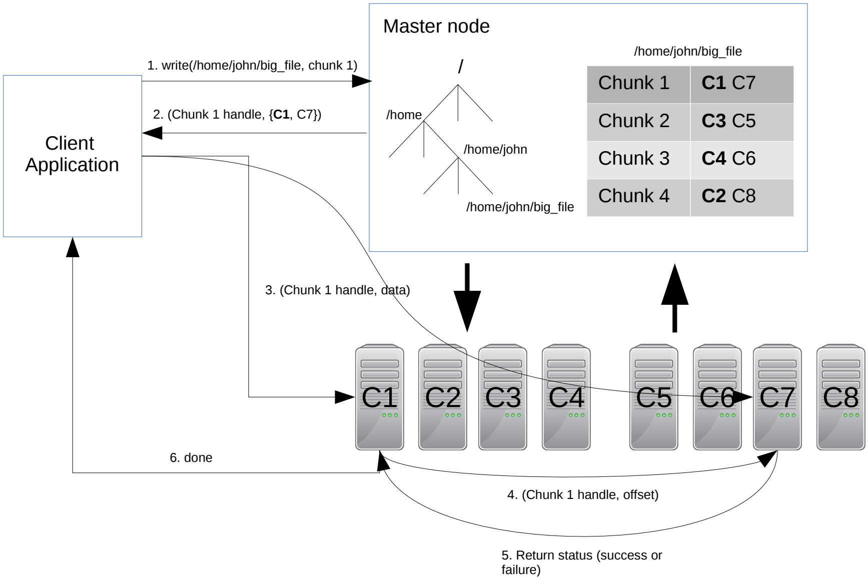
Write Example

- Make sure each replica contains the same data all the time
- One replica is designated to be the primary replica

Write Example

- Make sure each replica contains the same data all the time
- One replica is designated to be the primary replica
- Master pings the nodes to make sure they are alive

Write Example



Considerations

- Reads are very efficient operations

Considerations

- Reads are very efficient operations
- Writes are efficient if they append to the end of the file

Considerations

- Reads are very efficient operations
- Writes are efficient if they append to the end of the file
- Write in the middle of a file can be problematic

Considerations

- Reads are very efficient operations
- Writes are efficient if they append to the end of the file
- Write in the middle of a file can be problematic
- Primary replica decides the order in which to make writes:
  - Data is always consistent in all replicas

Replication Management

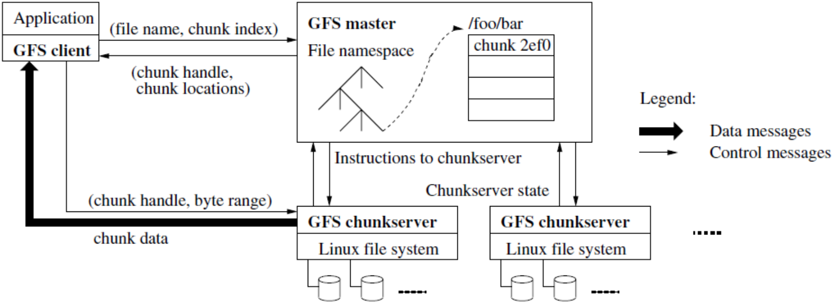
- Distributed file systems are usually hosted on large clusters
  - many nodes $\rightsquigarrow$ risk that one of them fails increases
  - commodity hardware: risk to fail is increased anyway
- Each chunk is stored redundantly on several chunk nodes (**replication**)
  - by defaut: 3
- Chunk node regularly send an I-am-alive-message to the master (**heartbeat**)
  - default: every 3s
- a chunk node without heartbeat for a longer period is considered to be **offline**/down/dead
  - default: after 10 minutes
- if a chunk node is found to be offline, the name node creates new replicas of its chunks spread over other chunk nodes.
  - until every chunk is replicated 3 times again

Outline

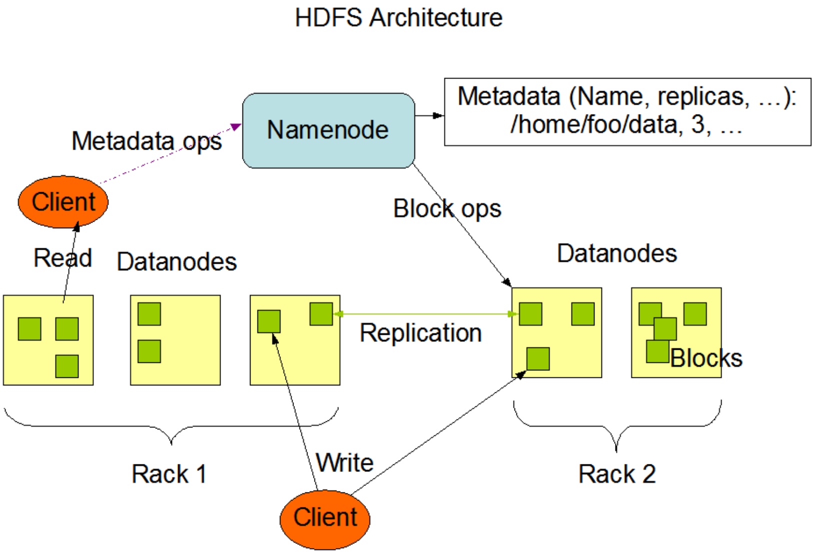
GFS vs. HDFS

	HDFS	**GFS**
Chunk Size	128Mb	64Mb
Default replicas	2 Files (data and generation stamp)	3 Chunknodes
Master	NameNode	GFS Master
Chunk Nodes	DataNode	Chunk Server

Google File System

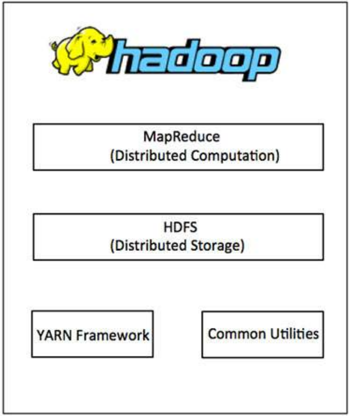


Hadoop Distributed File System



Outline

Hadoop Overall Architecture



source: http://www.tuto

Hadoop hdfs Setup (1/3)

1. Prerequisites:
   - several machines (≥ 1) with password-less ssh login
     - here: h0, h1, h2
     - test: on h0: **ssh h1** brings up a shell on h1
   - Java installed on all machines
     - test: on h0: **java -version** and **ssh h1 java -version** shows version
   - hadoop downloaded and unpacked on all machines (`http://hadoop.apache.org/releases.html`; here for v2.7.2)
     - put hadoop-2.7.2/bin and hadoop-2.7.2/sbin in the path
     - or always use full path names to hadoop binaries
     - test: on h0: **hadoop version** and **ssh h1 hadoop version** shows version

Hadoop hdfs Setup (2/3)

2. Configure Hadoop hdfs (identical on all machines):
   - create a configuration directory somewhere, say in /tmp/hadoop-conf
   - set environment variable **HADOOP_CONF_DIR** accordingly
   - put there two files, **core-site.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://h0:54310</value>
  </property>
</configuration>
```

   - and **hdfs-site.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
</configuration>
```

   - test: on h0: **hdfs getconf -namenodes** and **ssh h1 hdfs getconf -namenodes** yields h0.

Hadoop hdfs Setup (3/3)

3. Start hdfs:
   - on h0:
     - **hdfs namenode -format**: format disk / create data structures
     - **hdfs namenode**: start namenode daemon
     - **hdfs datanode**: start datanode daemon
   - on h1 and h2:
     - **hdfs datanode**: start datanode daemon
   - test: on h0: **hdfs dfsadmin -report** shows h0, h1 and h2.
     alternatively, visit the web interface at http://h0:50070

Hadoop hdfs Setup / Web Interface

Hadoop Overview **Datanodes** Datanode Volume Failures Snapshot Startup Progress Utilities

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
s1.ismll.de:50010 (147.172.223.225:50010)	2	In Service	449.78 GB	4 KB	135.81 GB	313.97 GB	0	4 KB (0%)	0	2.7.2
147.172.223.14:50010 (147.172.223.14:50010)	0	In Service	49.97 GB	4 KB	10.67 GB	39.31 GB	0	4 KB (0%)	0	2.7.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **df ⟨*path*⟩**, e.g., df /
  show free disk space

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory
- **mkdir ⟨*path*⟩**, e.g., mkdir /mydata
  create directory

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory
- **mkdir ⟨*path*⟩**, e.g., mkdir /mydata
  create directory
- **put ⟨*files*⟩. . . ⟨*path*⟩**, e.g., put abc.csv /mydata
  upload files to hdfs

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . .:

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory
- **mkdir ⟨*path*⟩**, e.g., mkdir /mydata
  create directory
- **put ⟨*files*⟩. . . ⟨*path*⟩**, e.g., put abc.csv /mydata
  upload files to hdfs
- **get ⟨*paths*⟩. . . ⟨*dir*⟩**, e.g., get /mydata/abc.csv abc-copy.csv
  download files from hdfs

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . .:

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory
- **mkdir ⟨*path*⟩**, e.g., mkdir /mydata
  create directory
- **put ⟨*files*⟩. . . ⟨*path*⟩**, e.g., put abc.csv /mydata
  upload files to hdfs
- **get ⟨*paths*⟩. . . ⟨*dir*⟩**, e.g., get /mydata/abc.csv abc-copy.csv
  download files from hdfs
- **cat ⟨*paths*⟩. . .** , e.g., cat /mydata/abc.csv
  pipe files from hdfs to stdout

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **df ⟨*path*⟩**, e.g., df /
  show free disk space
- **ls ⟨*path*⟩**, e.g., ls /
  list directory
- **mkdir ⟨*path*⟩**, e.g., mkdir /mydata
  create directory
- **put ⟨*files*⟩. . . ⟨*path*⟩**, e.g., put abc.csv /mydata
  upload files to hdfs
- **get ⟨*paths*⟩. . . ⟨*dir*⟩**, e.g., get /mydata/abc.csv abc-copy.csv
  download files from hdfs
- **cat ⟨*paths*⟩. . .** , e.g., cat /mydata/abc.csv
  pipe files from hdfs to stdout
- **mv ⟨*src*⟩. . .** ⟨*dest*⟩, e.g., mv /mydata/abc.csv /mydata/abc.txt
  move or rename files on hdfs

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **cp ⟨*src*⟩. . .** ⟨*dest*⟩, e.g., cp /mydata/abc.csv /mydata/abc-copy.txt
  copy files on hdfs

hdfs Filesystem Interface

hdfs dfs -⟨*command*⟩ . . . :

- **cp** ⟨*src*⟩. . . ⟨*dest*⟩, e.g., cp /mydata/abc.csv /mydata/abc-copy.txt
  copy files on hdfs

URLs can be used as path names:

- / denotes the hdfs root.
- **file:///** denotes the root of the local filesystem

hdfs Inspect File Health

hdfs fsck ⟨*path*⟩ -files -blocks -locations

shows information about where (datanode) which parts (blocks) of a file are stored.

```
Connecting to namenode via http://lst-uni.ismll.de:50070/fsck?ugi=lst&files=1&blocks=1&locations=1&path=%2Fmydata
FSCK started by lst (auth:SIMPLE) from /147.172.223.14 for path /mydata/rcv1_test.binary at Tue May 03 19:26:28 C
/mydata/rcv1_test.binary 1207864838 bytes, 9 block(s):  OK
0. BP-282002004-147.172.223.14-1462282706590:blk_1073741842_1018 len=134217728 repl=2 [DatanodeInfoWithStorage[14
1. BP-282002004-147.172.223.14-1462282706590:blk_1073741843_1019 len=134217728 repl=2 [DatanodeInfoWithStorage[14
2. BP-282002004-147.172.223.14-1462282706590:blk_1073741844_1020 len=134217728 repl=2 [DatanodeInfoWithStorage[14
3. BP-282002004-147.172.223.14-1462282706590:blk_1073741845_1021 len=134217728 repl=2 [DatanodeInfoWithStorage[14
4. BP-282002004-147.172.223.14-1462282706590:blk_1073741846_1022 len=134217728 repl=2 [DatanodeInfoWithStorage[14
5. BP-282002004-147.172.223.14-1462282706590:blk_1073741847_1023 len=134217728 repl=2 [DatanodeInfoWithStorage[14
6. BP-282002004-147.172.223.14-1462282706590:blk_1073741848_1024 len=134217728 repl=2 [DatanodeInfoWithStorage[14
7. BP-282002004-147.172.223.14-1462282706590:blk_1073741849_1025 len=134217728 repl=2 [DatanodeInfoWithStorage[14
8. BP-282002004-147.172.223.14-1462282706590:blk_1073741850_1026 len=134123014 repl=2 [DatanodeInfoWithStorage[14

Status: HEALTHY
 Total size:    1207864838 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      9 (avg. block size 134207204 B)
 Minimally replicated blocks:   9 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     2.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
```

hdfs Inspect File Health

hdfs fsck ⟨*path*⟩ -files -blocks -locations

shows information about where (datanode) which parts (blocks) of a file are stored.

```
v1_test.binary
016

.223.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.14:50010,DS-e3b3aa
.223.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.14:50010,DS-e3b3aa
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.14:50011,DS-783f2c
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58
.223.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.14:50011,DS-783f2c
.223.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58
```

Summary (1/2)

- Basic requirements for distributed filesystem are
  - **scalability**: perform multiple parallel reads and writes
  - **fault tolerance**: replicate files on several nodes
  - **transparency**: clients can access files like on a local filesystem
- Distributed filesystems partition files into **chunks** / **blocks**
  - **chunk/data nodes** store individual chunks/blocks of a file.
  - a **master/name node** stores the index
    - for every file and chunk, on which chunk nodes it is stored
- reading can be done from any chunk node storing a chunk
  - master is queried to find out which chunks nodes this are
- writing needs to be synchronized over chunk nodes storing a chunk
  - for every chunk there is a **primary** chunk node
  - the primary chunk node stores a chunk first,
    then replicates it to other chunk nodes
    and only after all have been written confirms successful write.

Summary (2/2)

- Reading and write-appending is efficient,
  write-in-the-middle is not possible (as it changes the chunk structure)
- The **Google File System** (GFS) is an early distributed filesystem
  - deployed large scale in Googles data centers.
  - **Hadoop File System** (HFS) is an open-source implementation very similar to GFS.

Further Readings

- Google File System, the original paper: Ghemawat et al. [2003]
- Brief tutorial on HDFS architecture: Gupta [2015]
- Hadoop File System: [White, 2015, ch. 3]

References

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

Lokesh Gupta. Hdfs – hadoop distributed file system architecture tutorial, 2015. URL http://howtodoinjava.com/big-data/hadoop/hdfs-hadoop-distributed-file-system-architecture-tutorial/.

Tom White. *Hadoop: The Definitive Guide*. O'Reilly, 4 edition, 2015.