Big Data Analytics

3. Distributed File Systems

Lars Schmidt-Thieme

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

original slides by Lucas Rego Drumond, ISMLL

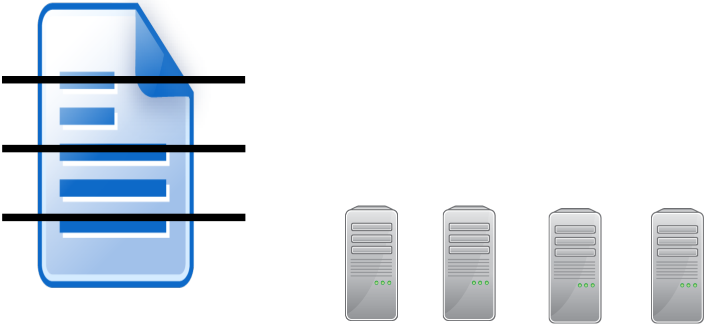
Outline

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)

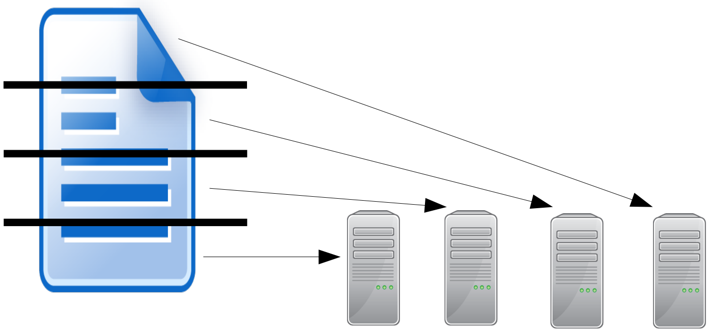
Outline

Why do we need a Distributed File System?

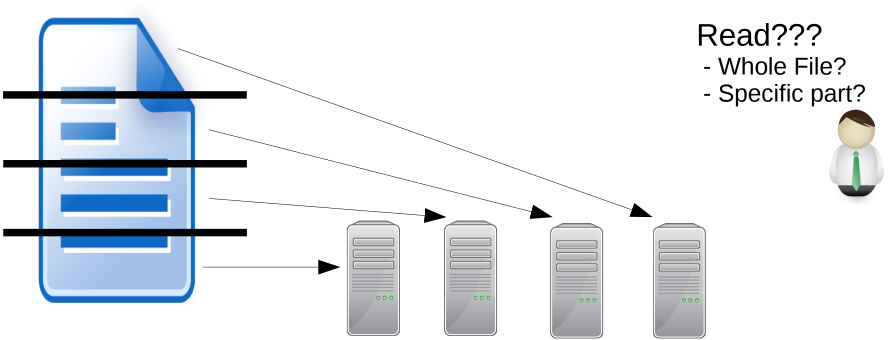
Why do we need a Distributed File System?



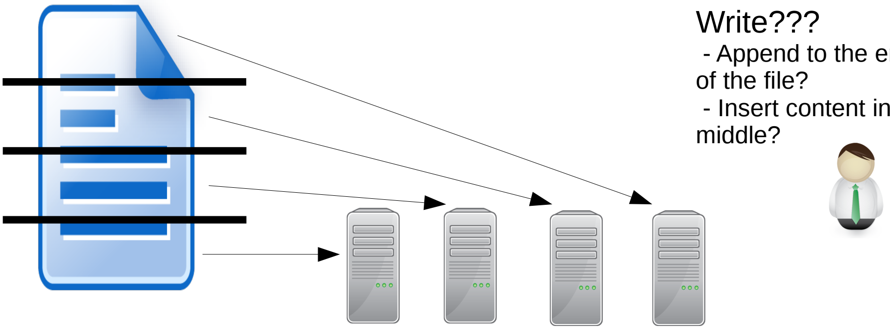
Why do we need a Distributed File System?



Why do we need a Distributed File System?



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**

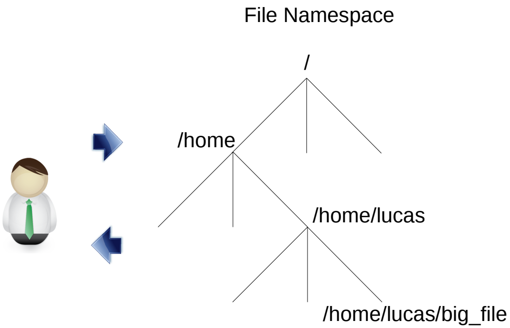
Why do we need a Distributed File System?

We want to:

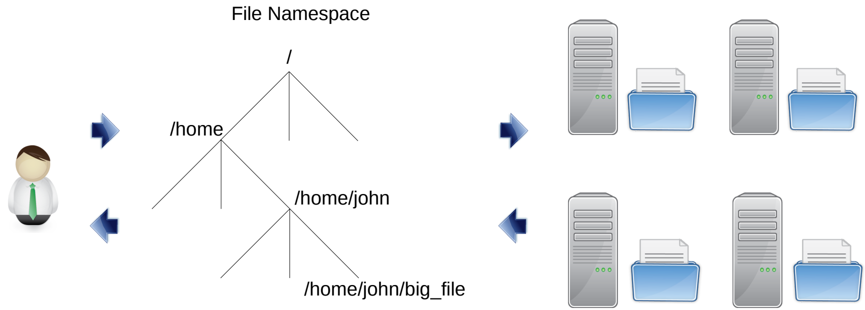
- Read large data fast
  - **scalability**: perform multiple **parallel reads and writes**
- Have the files available even if one computer crashes
  - **fault tolerance**: **replication**
- Hide parallelization and distribution details
  - **transparency**: clients can access it like a local filesystem

Outline

What is a Distributed File System?



What is a Distributed File System?



Examples

- GFS (Google Inc.)
- HDFS (Apache Software Foundation)
- Ceph (Inktank, Red Hat)
- MooseFS (Core Technology / Gemius)
- Windows Distributed File System (DFS) (Microsoft)
- FhGFS (Fraunhofer)
- GlusterFS (Red Hat)
- Lustre
- Ibrix

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

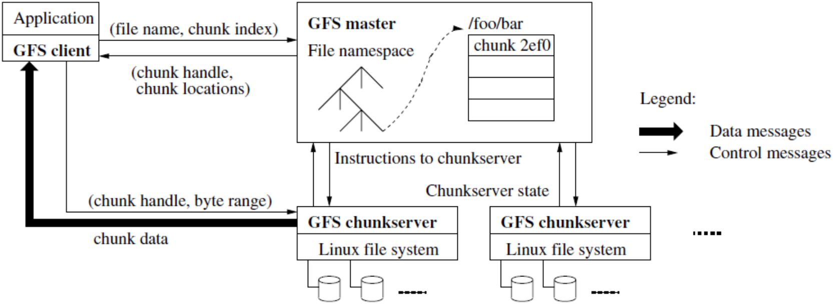
Components

A typical distributed filesystem contains the following components

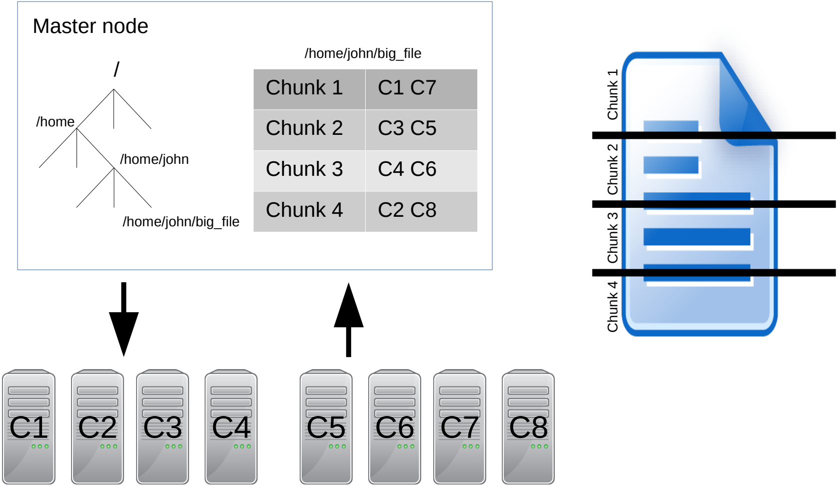
- Clients - they interface with the user
- Chunk nodes - stores chunks of files
- Master node - stores which parts of each file are on which chunk node

Distributed File Systems

The Google File System Architecture

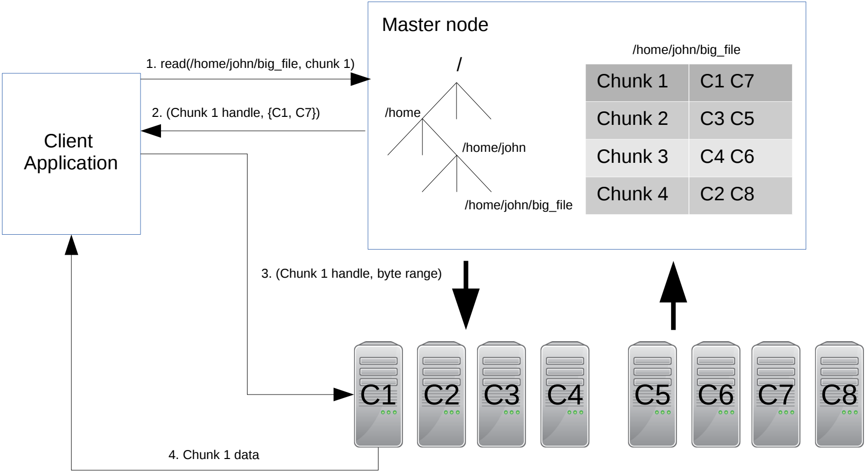


Distributed File Systems - Storing files



Chunk 1	C1 C7
Chunk 2	C3 C5
Chunk 3	C4 C6
Chunk 4	C2 C8

Read Example



Write Example

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

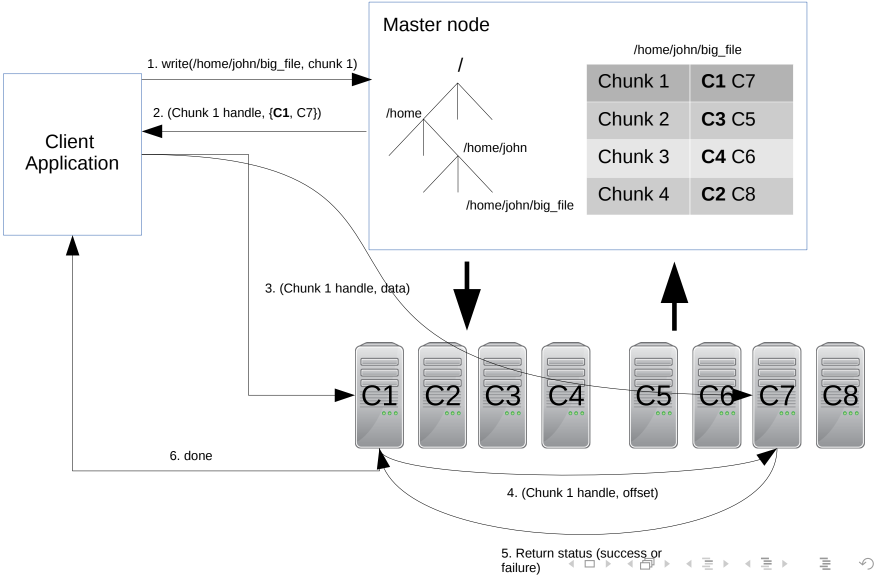
Write Example

- 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 are appends to the end of the file

Considerations

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

Considerations

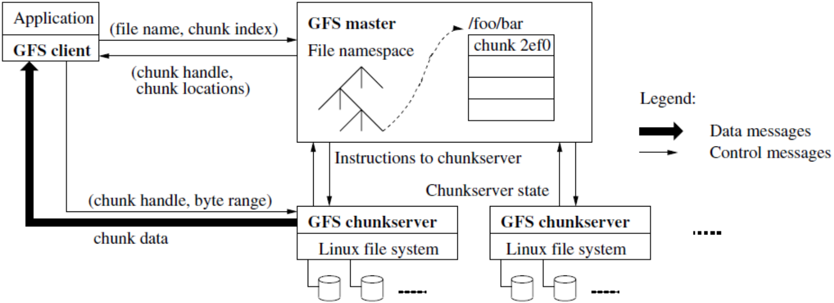
- Reads are very efficient operations
- Writes are efficient if they are appends to the end of the file
- Write in the middle of a file can be problematic
- Primary replica decides the order in which to make writes:
    - Data is always consistent in all replicas

Outline

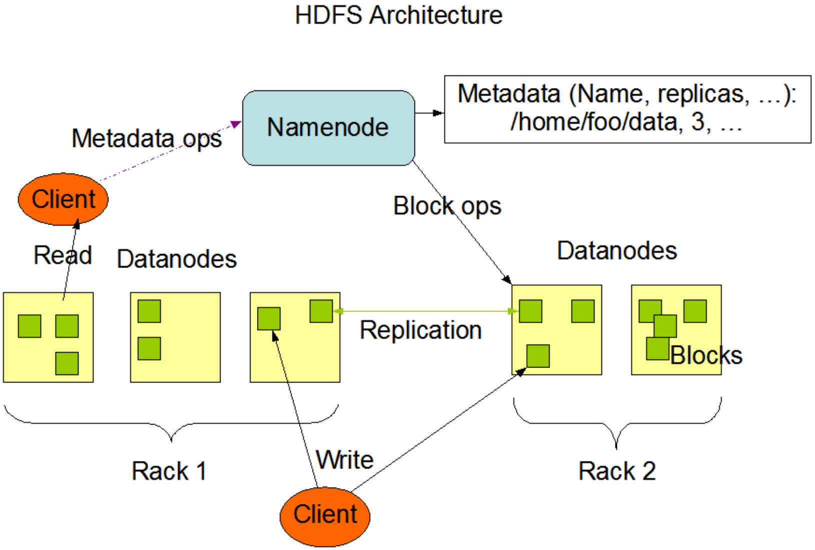
GFS vs. HDFS

	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



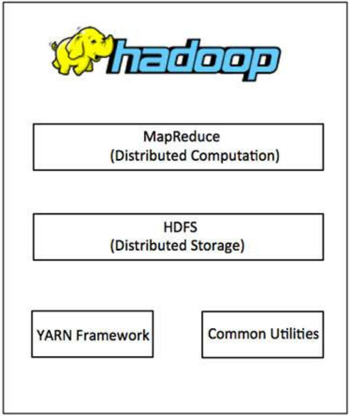
Hadoop Distributed File System



Outline

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)

Hadoop Overall Architecture



source: `http://www.tutorialspoint.com/hadoop/hadoop_introduction.htm`

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

Hadoop, 2015.

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=%2Fm
FSCK started by lst (auth:SIMPLE) from /147.172.223.14 for path /mydata/rcv1_test.binary at Tue May 03 19:2
/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 [DatanodeInfoWithStor
1. BP-282002004-147.172.223.14-1462282706590:blk_1073741843_1019 len=134217728 repl=2 [DatanodeInfoWithStor
2. BP-282002004-147.172.223.14-1462282706590:blk_1073741844_1020 len=134217728 repl=2 [DatanodeInfoWithStor
3. BP-282002004-147.172.223.14-1462282706590:blk_1073741845_1021 len=134217728 repl=2 [DatanodeInfoWithStor
4. BP-282002004-147.172.223.14-1462282706590:blk_1073741846_1022 len=134217728 repl=2 [DatanodeInfoWithStor
5. BP-282002004-147.172.223.14-1462282706590:blk_1073741847_1023 len=134217728 repl=2 [DatanodeInfoWithStor
6. BP-282002004-147.172.223.14-1462282706590:blk_1073741848_1024 len=134217728 repl=2 [DatanodeInfoWithStor
7. BP-282002004-147.172.223.14-1462282706590:blk_1073741849_1025 len=134217728 repl=2 [DatanodeInfoWithStor
8. BP-282002004-147.172.223.14-1462282706590:blk_1073741850_1026 len=134123014 repl=2 [DatanodeInfoWithStor

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 %)
 Number of data-nodes:          3
```

hdfs Inspect File Health

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

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

```
test.binary


3.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.14:50010,DS-e3b3aadb-
3.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.14:50010,DS-e3b3aadb-
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58eb5
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58eb5
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.14:50011,DS-783f2c65-
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58eb5
3.14:50011,DS-783f2c65-69ea-46ff-88ed-deebabf73158,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58eb5
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.14:50011,DS-783f2c65-
3.14:50010,DS-e3b3aadb-4f1c-49d1-872b-1879362f35c1,DISK], DatanodeInfoWithStorage[147.172.223.225:50010,DS-8aa58eb5
```