The Hadoop Distributed Filesystem (HDFS)

The Origin of HDFS

Google's File System (GFS)

HDFS

- **Distributed filesystems**: Filesystems that manage the storage across a network of machines. This is the only way to deal with the situation where a dataset outgrows the storage capacity of of a single physical machine.
- **Hadoop Distributed Filesystem (HDFS)**: A general purpose filesystem abstraction designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. It is Hadoop's flagship filesystem. It can deal with both unstructured and structured data.

Characteristics of HDFS

- **Highly Scalable:** can have a computer cluster from several to tens of thousands of nodes.
- **Reliable**: tolerant to fault due to built-in redundancy.
- **Massive parallel processing**: supports massive parallel processing with linearly scalable bandwidth.
- **Very easy to manage**: it is managed automatically.

Parallel Computing Paradigms

- Shared memory by partitioning work.
- Message passing by partitioning data.
- A hybrid of the two exists.
- What Hadoop does is to use a higher level of abstraction than pure shared memory or massage passing. It is a message-passing, data parallel, and pipelined work paradigm. This is a new way of parallel computing. The beauty of the system is that all that complicated work is handled automatically.

Areas where HDFS is a good fit:

- **Very large files:** Hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.
- **Streaming data access:** The most efficient data processing pattern: write once, read many times. Data analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.
- **Using commodity hardware:** Cheaper. Although the chance of node failure is higher (at least for large clusters), HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

Areas where HDFS is not a good fit today:

- **Low-latency data access**: A requirement of tens of milliseconds range will not work well with HDFS since it is optimized for delivering a high throughput of data (at the expense of latency).
- **Lots of small files**: The namenode holds filesystem metadata in memory. The amount of this memory limits the number of files in a filesystem. While storing millions of files is feasible, storing billions of files is beyond the capability of current hardware.
- **Multiple writers and arbitrary file modifications**: Files in HDFS may be written to by a single writer. Writes are always made at the end of the file. There is no support for multiple writers, or for modifications at arbitrary offsets in the file.

Some Basic HDFS Concepts

- Filesystem options
- Blocks
- Namenodes and Datanodes
- HDFS High-Availability

The Underlying Filesystem Options

- **ext3 (third extended file system)**
  - Used by Yahoo
  - Released in 2001
  - Up to 32 TB of file size
- **ext4 (fourth extended file system)**
  - Used by Google
  - Released in 2008
  - Fast like XFS
  - Up to 1 EB (exabytes)

Commonly used in HDFS

- **XFS**
  - Released in 1993 by Silicon Graphics
  - Fast (parallel I/O)
  - Some drawbacks (cannot be shrunk; slow metadata processing)

Blocks

- **Disk blocks**: The minimum amount of data that it can read or write, which is normally 512 bytes.
- **Filesystem blocks** (on a single disk): An integral multiple of the disk block size, typically a few kilobytes in size.
- **HDFS blocks** (abstraction): Its size is a much larger unit—64 MB or **128 MB** by default. Files in HDFS are broken into block-sized chunks, stored as independent units. Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage.

Why Is a Block in HDFS So Large?

- To minimize the cost of seeks. A large block size like this (128 or 64 MB) can make the time to transfer the data from the disk be significantly greater than the time to seek to the start of the block. Thus, the data transfer can operate at near the disk transfer rate.

  Example: If the seek time is 10 ms and the transfer rate is 100 MB/s, to make the seek time 1% of the transfer time, the block size should be around 100 MB.
- The map tasks in MapReduce operate on one block at a time, so if you have too few tasks (fewer than nodes in the cluster), your jobs will run slower than they could otherwise.

Benefits of block abstraction for an HDFS

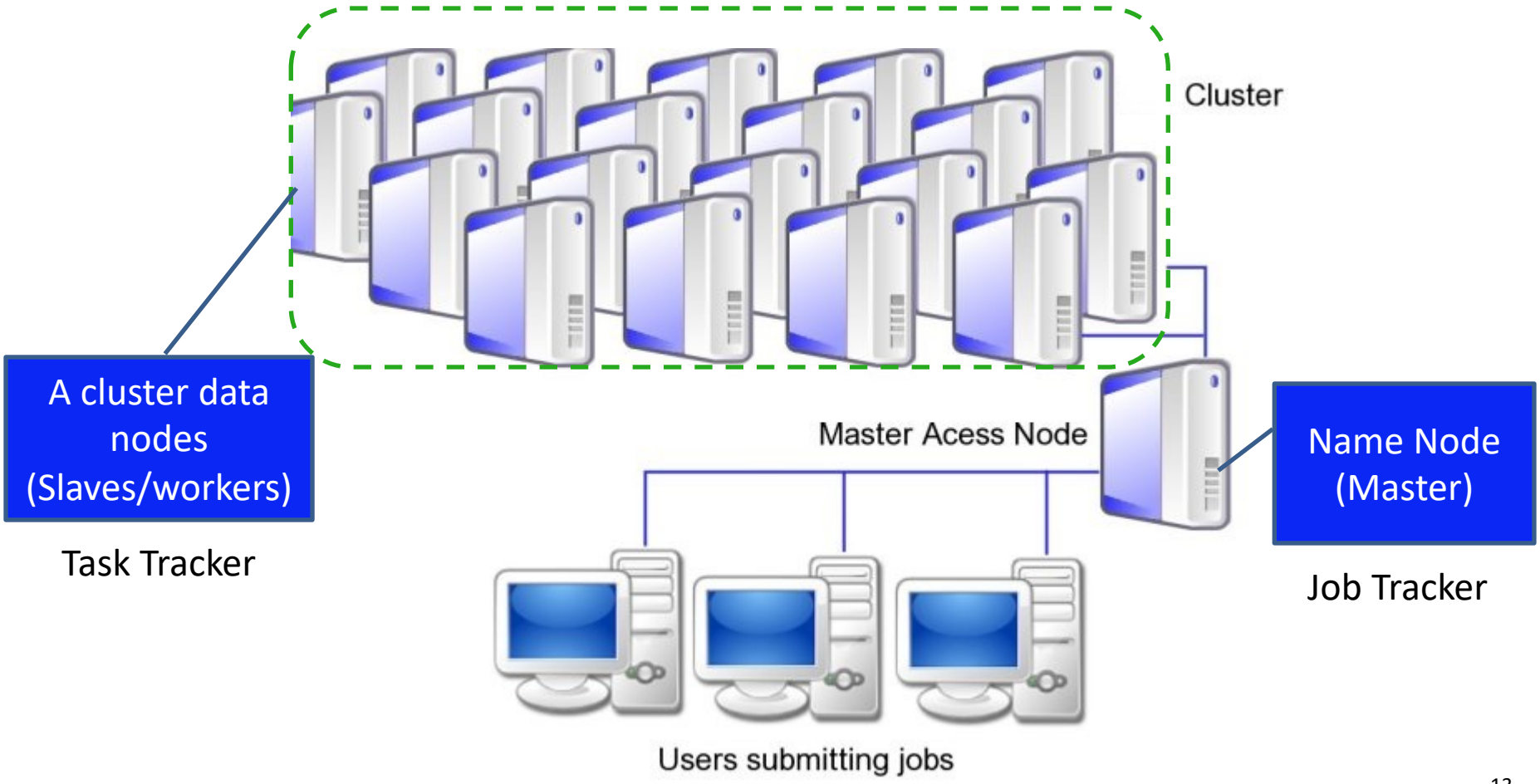
- A file can be larger than any single disk in the network.
- Making the unit of abstraction a block simplifies the storage subsystem (it is easy to calculate how many blocks can be stored on a given disk).
- Blocks fit well with replication for providing fault tolerance and availability. Each block is replicated to a small number (typically 3) of physically separate machines. If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client. A block that is no longer available due to failure can be replicated from its alternative locations to other live machines to bring the replication factor back to the normal level.
- The *fsck* command in HDFS understands blocks. For example, running:

  hdfs fsck / -files -blocks

  will list the blocks of all the files in the filesystem.

A Hadoop Cluster

A Hadoop cluster is composed of a Name Node and a cluster of Data Nodes.



Name Nodes and Data Nodes

- The **name node** manages the filesystem namespace. It maintains the filesystem tree and the metadata (held in memory). This information is stored persistently on the local disk: the namespace image and the edit log. The name node knows the locations of all the blocks of a given file. It is reconstructed from data nodes when the system starts.
- **Data nodes** are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the name node), and they report back to the name node periodically with lists of blocks that they are storing.

The Importance of the Name Node

Without the name node, the filesystem has no use (a single point of failure, or SPOF). It is critical to make this node resilient to failure. There are two mechanisms.

- Back up the files that make up the persistent state of the filesystem metadata. The name node writes its persistent state to multiple file systems, the local disk as well as a remote NFS mount.
- Run a secondary name node. Despite its name, in Hadoop 1 it did not act as a name node (just a checkpoint) but periodically merges the namespace image with the edit log, which can be used to rebuild a new name node if a total failure of name node happens.

HDFS High-Availability

- Hadoop 2 implements a pair of name nodes in an active-standby configuration as a standard operational procedure. If the active name node does fail, the standby can take over quickly (in a few tens of seconds).
- Data nodes send block reports to both name nodes since the block mapping is stored in the memory of name nodes.
- Only one name node is active at a time.
- Each name node runs a failover controller process to monitor its health by using the heartbeating mechanism.
- A fencing mechanism is used to prevent a failing primary name node from damaging the cluster by killing all its processes or even “shooting the other node in the head” (cut the power).

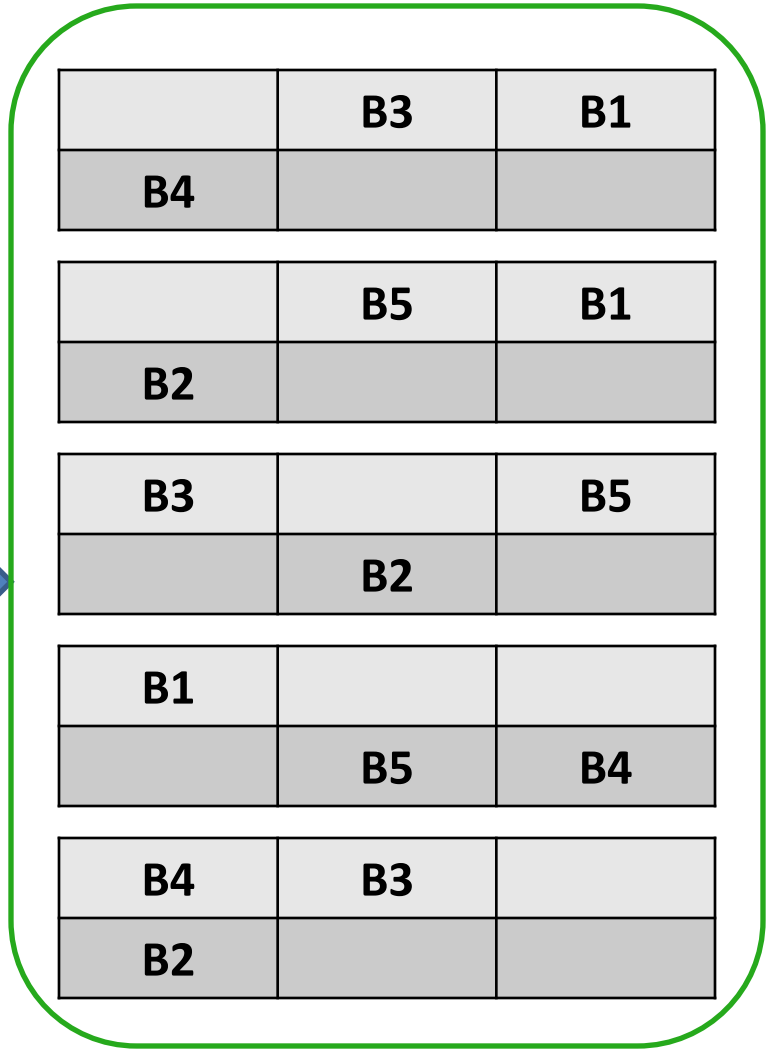
The Name Node Holds the Metadata

- Names of the files
- Data blocks of each file
- File system permissions
- Last access time
- Disk space quotas
- …

Name node file metadata:
File1.txt → B1, B2, B3
File2.txt → B4, B5

Default block replication:
dfs.replication = 3 (within the file called hdfs-site.xml)

Name Node



Data Nodes (5)

The Name Node Has an Embedded Web Server

It shows some statistics of the filesystem:

- Time the name node started, version, time of compilation
- Name node logs
- Cluster summary, such as
  - Number of files and directories blocks
  - Configured capacity
  - DFS used
  - DFS remaining
  - Live nodes
  - Dead nodes
- Name node storage
  - Storage directory
  - Type
  - State

An Example of Name Node Statistics (1)

Summary

Security is off.

Safemode is off.

1035 files and directories, 801 blocks = 1836 total filesystem object(s).

Heap Memory used 34.27 MB of 240 MB Heap Memory. Max Heap Memory is 240 MB.

Non Heap Memory used 79.37 MB of 81.48 MB Commited Non Heap Memory. Max Non Heap Memory is <unbonded>.

Configured Capacity:	41.64 GB
DFS Used:	1.9 GB (4.57%)
Non DFS Used:	13.03 GB
DFS Remaining:	24.27 GB (58.3%)
Block Pool Used:	1.9 GB (4.57%)
DataNodes usages% (Min/Median/Max/stdDev):	4.57% / 4.57% / 4.57% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	2/13/2018, 12:42:49 PM
Last Checkpoint Time	2/13/2018, 11:23:57 AM

An Example of Name Node Statistics (2)

NameNode Journal Status

Current transaction ID: 9511

Journal Manager	State
FileJournalManager(root=/hadoop/hdfs/namenode)	EditLogFileOutputStream(/hadoop/hdfs/namenode/current/edits_inprogress_0000000000000006552)

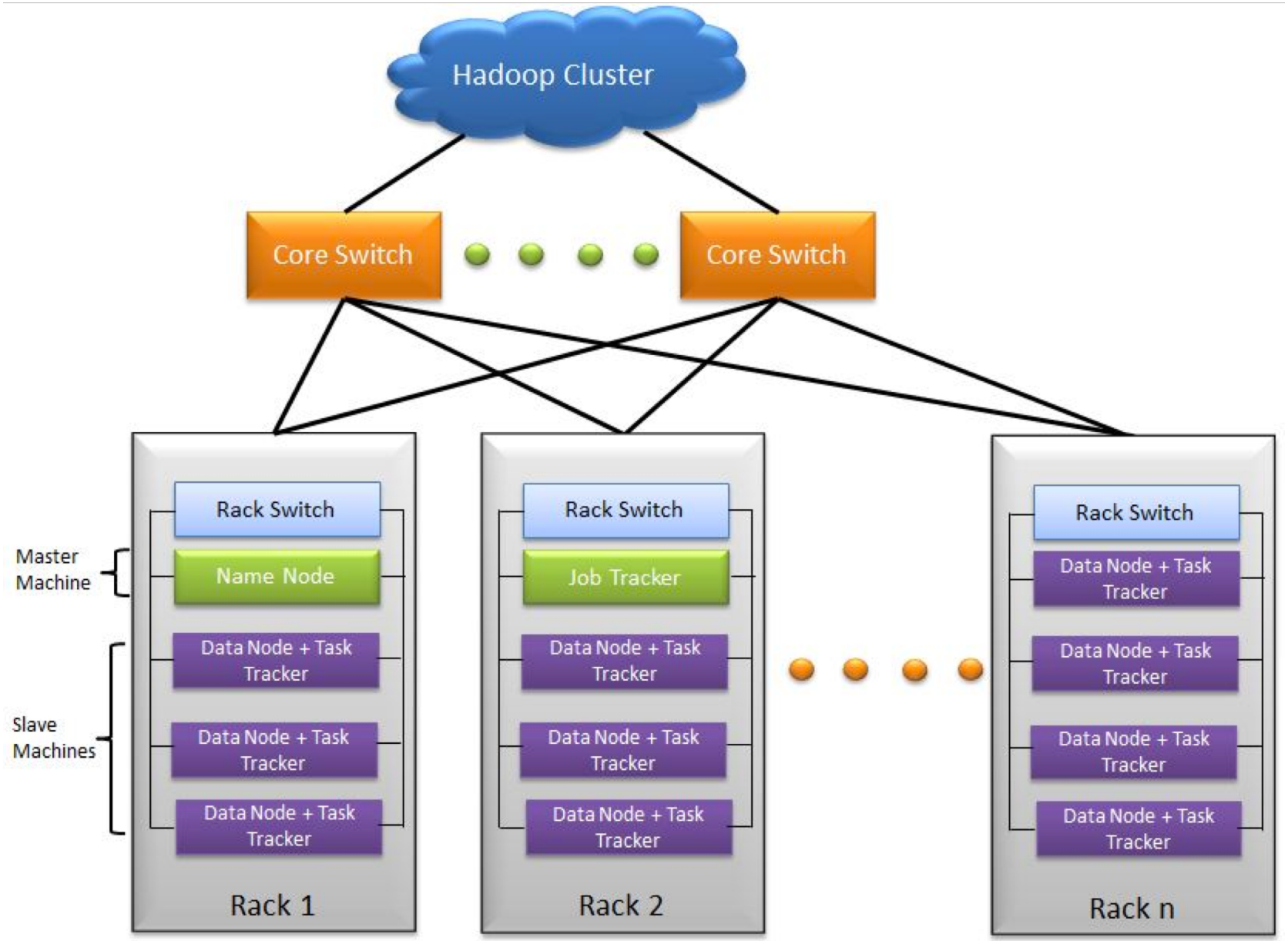
NameNode Storage

Storage Directory	Type	State
/hadoop/hdfs/namenode	IMAGE_AND_EDITS	Active

DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	41.64 GB	1.9 GB (4.57%)	24.27 GB (58.3%)	1.9 GB	1

A Diagram of a HDFS Cluster

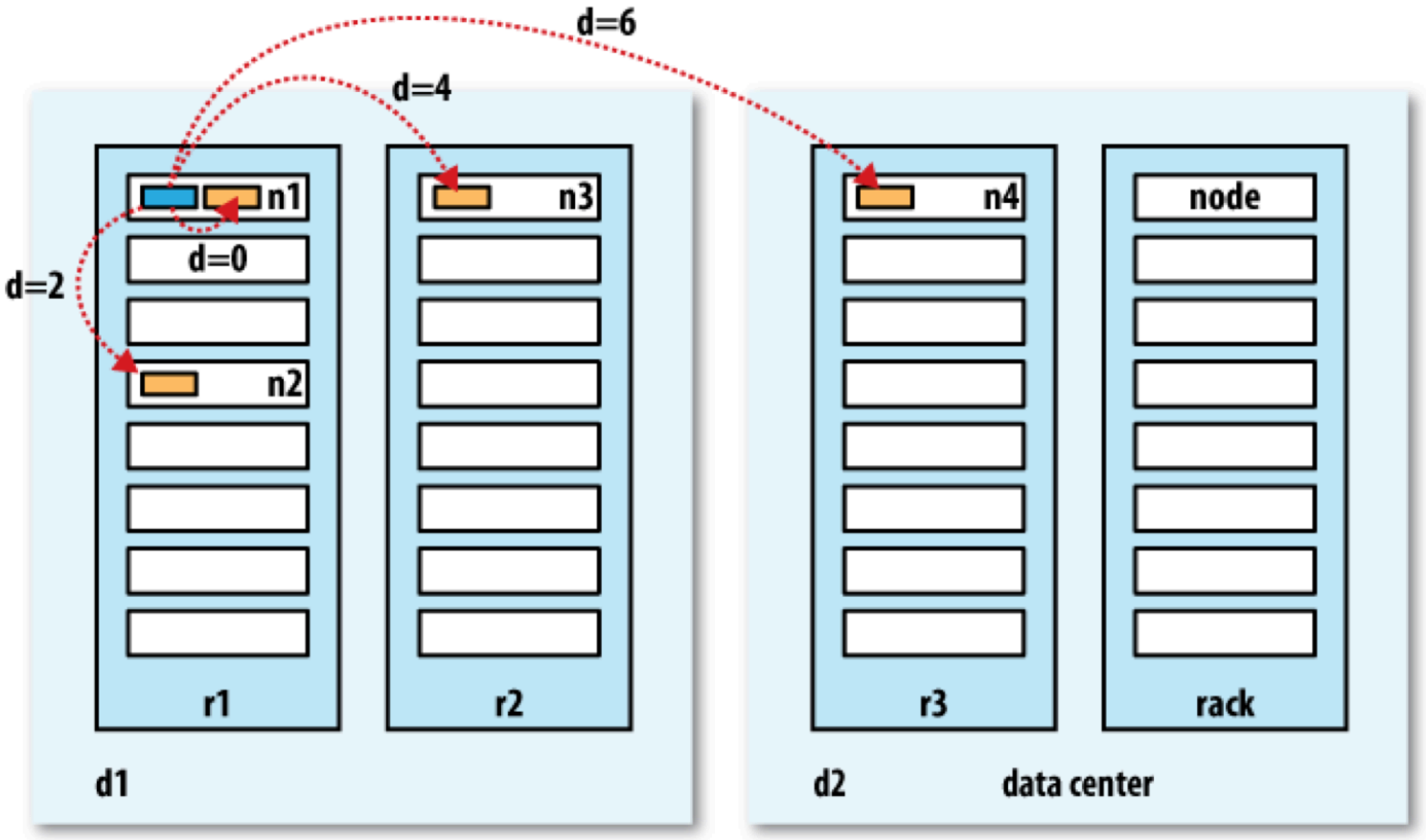


Hadoop at Yahoo

An Example: The Yahoo Hadoop Cluster

- 8 core switches
- 100 racks
- 40 slaves (data nodes) per rack
- 1 Gigabit in-rack network
- 10 Gigabit between-racks network (8 Gigabits for HDFS, 2 Gigabits for MR admin and user traffic)
- 11 PB in total

Network Distance in an HDFS Cluster



Use the bandwidth between two nodes as a measure of distance:

- distance(/d1/r1/n1, /d1/r1/n1) = 0 (processes on the same node)
- distance(/d1/r1/n1, /d1/r1/n2) = 2 (different nodes on the same rack)
- distance(/d1/r1/n1, /d1/r2/n3) = 4 (different racks in the same data center)
- distance(/d1/r1/n1, /d2/r3/n4) = 6 (in different data centers)

Name Node is Rack Aware

- Allows consideration of a node's physical location, when allocating storage and scheduling tasks.
- Can diagnose the health of the files system and can rebalance the data on different nodes.
- Handles different types of cluster that might otherwise require operator intervention. This design allows a single operator to maintain a cluster of 1000s of nodes.

Basic Filesystem Operations

- In the Filesystem, we can do all of the usual operations, such as reading files, creating directories, moving files, deleting data, and listing directories.
- This is through Hadoop's filesystem shell command fs, which supports many subcommands, e.g., -mkdir.
- You can type hadoop fs -help to get detailed help on every command.
- The files in HDFS are not visible with your normal Linux/Unix commands due to different URI schemes.

Some Common Hadoop Commands

- hadoop fs -ls

  List the contents that match the specified file pattern.
- hadoop fs -mkdir

  Create a directory in a specified location.
- hadoop fs -copyFromLocal (= hadoop fs -put)

  Copy a file(s) from the local system into HDFS
- hadoop fs -copyToLocal (= hadoop fs -get)

  Copy a file(s) from HDFS to the local file system.
- hadoop fs -rm

  Delete all files matching the specified pattern.
- hadoop fs -mv

  Move files from a source to destination.

Create an HDFS directory at System Root

- Create an HDFS directory, /stsci5065/data, at the CentOS root to hold your files.

```
[root@sandbox ~]# hadoop fs -mkdir /stsci5065
[root@sandbox ~]# hadoop fs -mkdir /stsci5065/data
```

- This directory is not visible to normal Linux/Unix, for example, the find command cannot find it.

```
[root@sandbox ~]# find /stsci5065/data
find: `/stsci5065/data': No such file or directory
```

- Neither does the ls command.

```
[root@sandbox ~]# ls /stsci5065
ls: cannot access /stsci5065: No such file or directory
```

The New Directory is Visible in HDFS

- The stsci5065 directory can be shown with the HDFS' hadoop fs -ls command.

```
[root@sandbox ~]# hadoop fs -ls /
Found 15 items
drwxrwxrwx   - yarn   hadoop          0 2017-07-28 14:20 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2017-07-28 14:06 /apps
drwxr-xr-x   - yarn   hadoop          0 2017-07-28 14:00 /ats
drwxr-xr-x   - hdfs   hdfs            0 2017-07-28 14:14 /demo
drwxr-xr-x   - hdfs   hdfs            0 2017-07-28 14:00 /hdp
drwx------   - livy   hdfs            0 2017-07-28 14:02 /livy-recovery
drwx------   - livy   hdfs            0 2017-07-28 14:01 /livy2-recovery
drwxr-xr-x   - mapred hdfs            0 2017-07-28 14:00 /mapred
drwxrwxrwx   - mapred hadoop          0 2017-07-28 14:00 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2017-07-28 14:00 /ranger
drwxrwxrwx   - spark  hadoop          0 2017-07-28 14:27 /spark-history
drwxrwxrwx   - spark  hadoop          0 2018-02-15 23:32 /spark2-history
drwxr-xr-x   - root   hdfs            0 2018-02-15 23:28 /stsci5065
drwxrwxrwx   - hdfs   hdfs            0 2017-07-28 14:21 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2018-02-15 23:16 /user
```

Download Some Data Files from Internet

wget http://www.gutenberg.org/files/100/100-0.txt
wget ftp://ftp.ncbi.nih.gov/genomes/INFLUENZA/influenza.cds
wget http://www.gutenberg.org/files/74/74-0.txt

Note:

- 100-0.txt (pg100.txt, shakespeare.txt)
- 74-0.txt (tom_sawyer.txt)

File Permissions in HDFS

- HDFS has a permissions model for files and directories that is much like POSIX (a family of standards for compatibility b/n OS').
- Permissions are managed in three distinct scopes or classes, which are known as owner, group, and others.
- Three type of permissions: read (r), write (w), and execute (x)

Three Permission Triads (modes)	
1st triad	what the owner can do
2nd triad	what the group members can do
3rd triad	what others can do
Each Triad	
1st character	r: readable
2nd character	w: writable
3rd character	x: executable

- The name node's processes are super-users and permissions checks are not performed for them.

rwxr-xr--

owner group others

Load a Data File into HDFS with Commands

- Load a local file, shakespeare.txt, into Hadoop's HDFS using the -copyFromLocal option of the Hadoop fs command.

```
hadoop fs -copyFromLocal pg100.txt /stsci5065/data
```

- Load more files to HDFS:
  - tom_sawyer.txt
  - influenza.cds

```
hadoop fs -put tom_sawyer.txt /stsci5065/data

hadoop fs -put influenza.cds /stsci5065/data
```

Note: You can load multiple files at once with one command.

List the Files Just Loaded

[root@sandbox /]# hadoop fs -ls /stsci5065/data

```
[root@sandbox ~]# hadoop fs -ls /stsci5065/data
Found 3 items
-rw-r--r--   1 root hdfs    5856576 2018-02-15 23:20 /stsci5065/data/100-0.txt
-rw-r--r--   1 root hdfs 1082513347 2018-02-15 23:21 /stsci5065/data/influenza.cds
-rw-r--r--   1 root hdfs     433025 2018-02-15 23:24 /stsci5065/data/tom_sawyer.txt
```

Check the Name Node Browser at port 50070

Browse Directory

/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxrwxrwx	yarn	hadoop	0 B	7/28/2017, 10:20:55 AM	0	0 B	app-logs
drwxr-xr-x	hdfs	hdfs	0 B	7/28/2017, 10:06:42 AM	0	0 B	apps
drwxr-xr-x	yarn	hadoop	0 B	7/28/2017, 10:00:13 AM	0	0 B	ats
drwxr-xr-x	hdfs	hdfs	0 B	7/28/2017, 10:14:33 AM	0	0 B	demo
drwxr-xr-x	hdfs	hdfs	0 B	7/28/2017, 10:00:22 AM	0	0 B	hdp
drwx------	livy	hdfs	0 B	7/28/2017, 10:02:03 AM	0	0 B	livy-recovery
drwx------	livy	hdfs	0 B	7/28/2017, 10:01:55 AM	0	0 B	livy2-recovery
drwxr-xr-x	mapred	hdfs	0 B	7/28/2017, 10:00:21 AM	0	0 B	mapred
drwxrwxrwx	mapred	hadoop	0 B	7/28/2017, 10:00:29 AM	0	0 B	mr-history
drwxr-xr-x	hdfs	hdfs	0 B	7/28/2017, 10:00:08 AM	0	0 B	ranger
drwxrwxrwx	spark	hadoop	0 B	7/28/2017, 10:27:20 AM	0	0 B	spark-history
drwxrwxrwx	spark	hadoop	0 B	2/15/2018, 6:44:08 PM	0	0 B	spark2-history
drwxr-xr-x	root	hdfs	0 B	2/15/2018, 6:28:33 PM	0	0 B	stsci5065
drwxrwxrwx	hdfs	hdfs	0 B	7/28/2017, 10:21:33 AM	0	0 B	tmp
drwxr-xr-x	hdfs	hdfs	0 B	2/15/2018, 6:16:29 PM	0	0 B	user

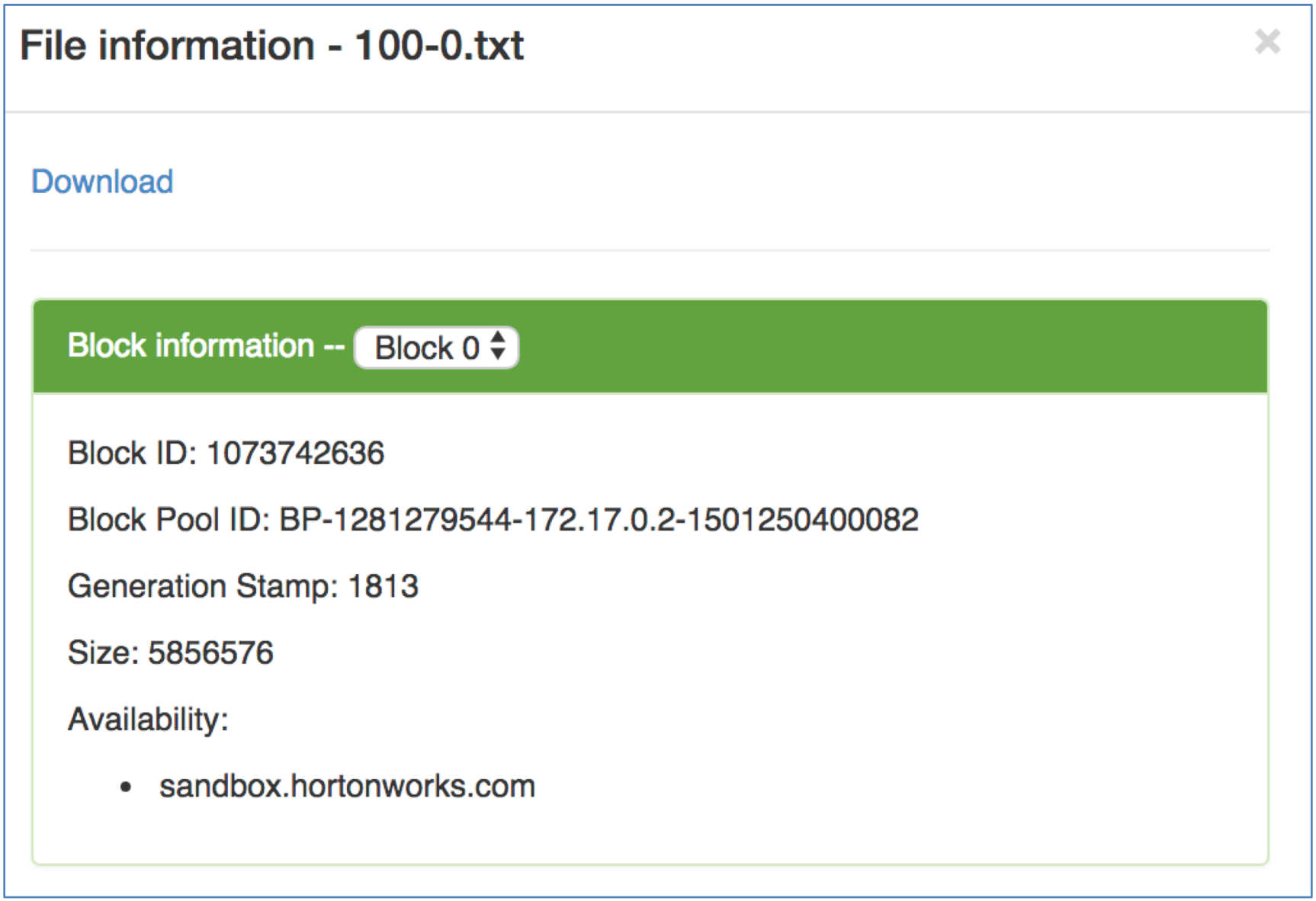
The Contents of /stsci5065/data in Data Node Browser

Browse Directory

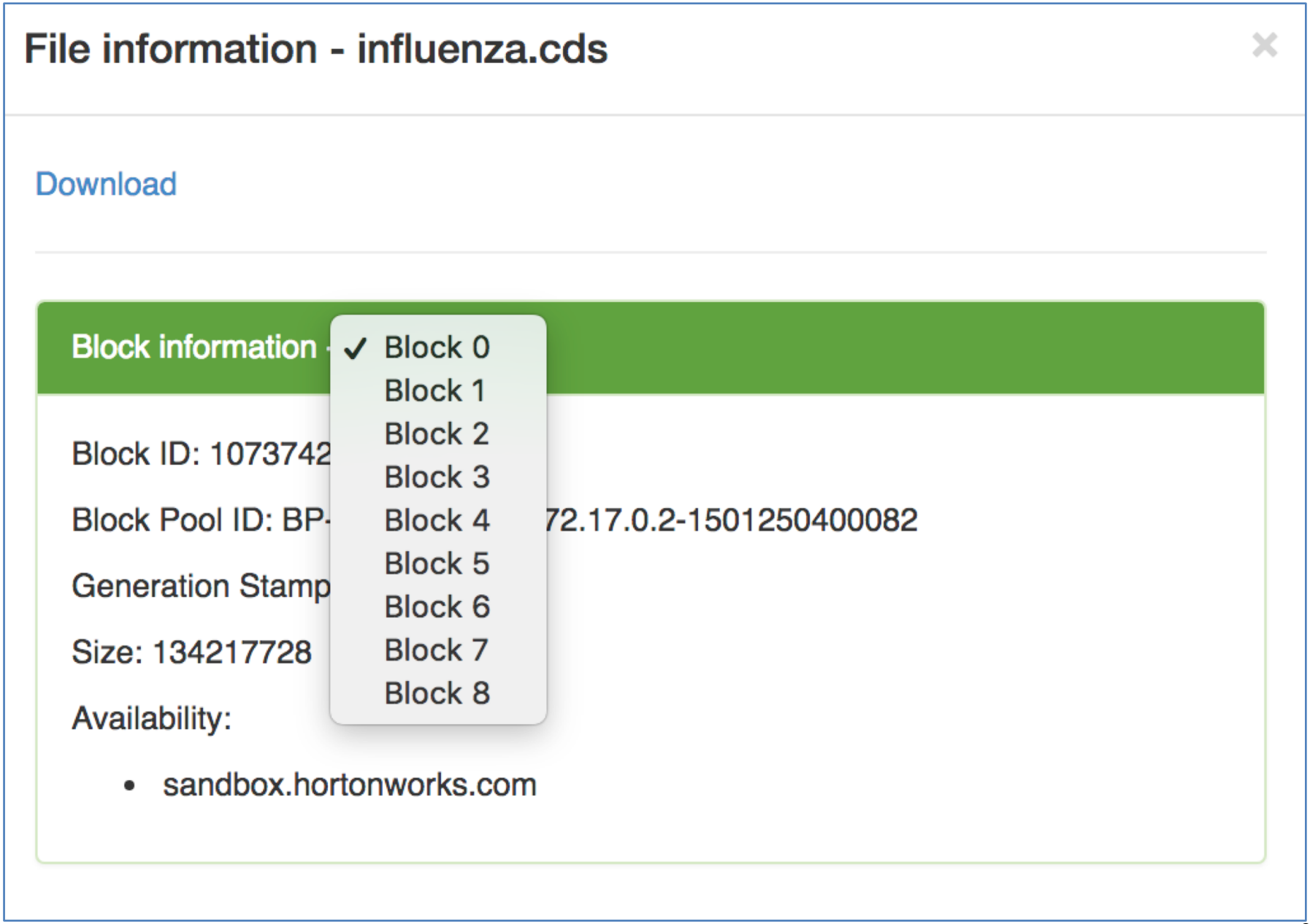
/stsci5065/data Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	hdfs	5.59 MB	2/15/2018, 6:20:56 PM	1	128 MB	100-0.txt
-rw-r--r--	root	hdfs	1.01 GB	2/15/2018, 6:21:25 PM	1	128 MB	influenza.cds
-rw-r--r--	root	hdfs	422.88 KB	2/15/2018, 6:24:15 PM	1	128 MB	tom_sawyer.txt

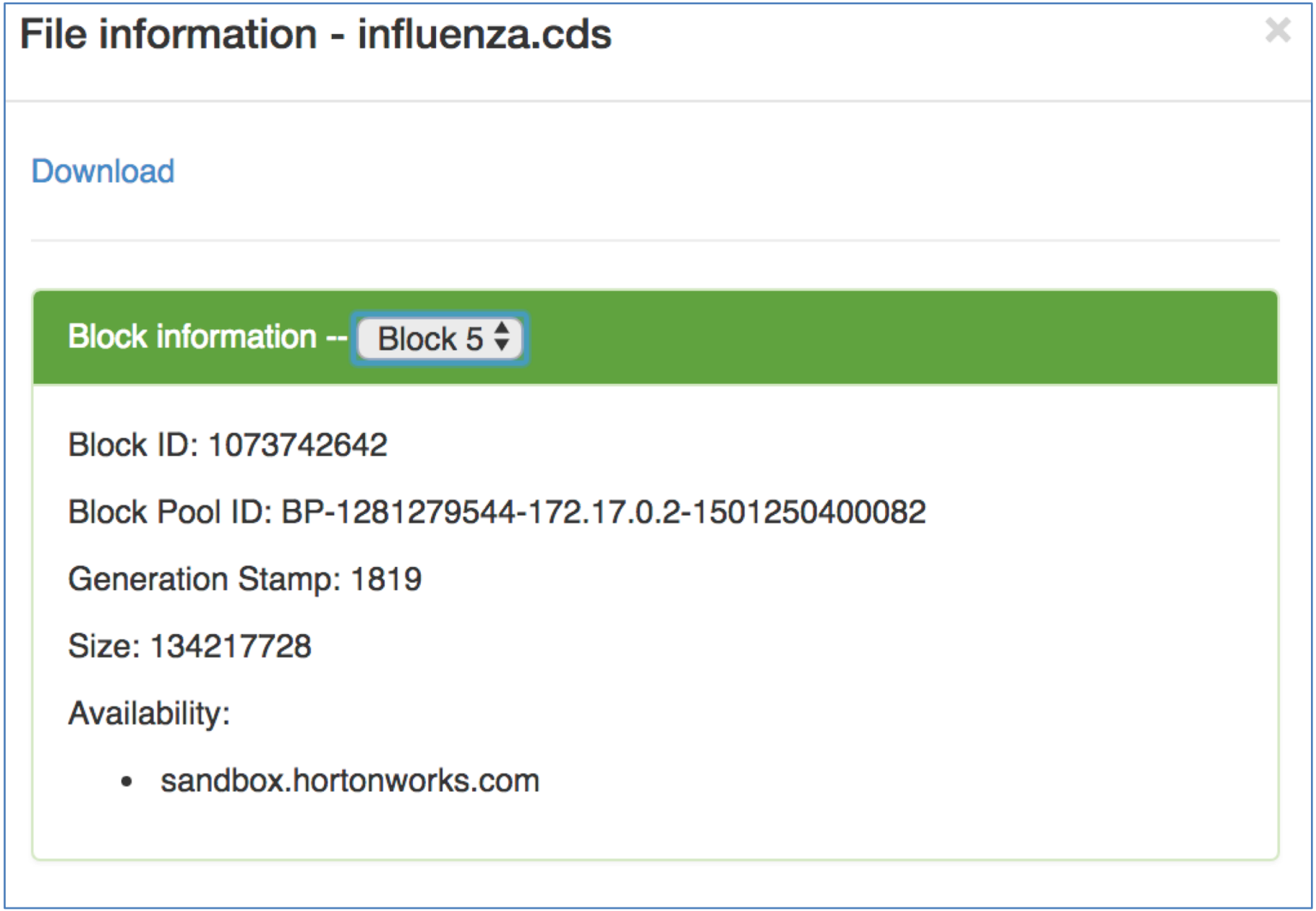
File information of pg100.txt



File information of influenza.cds



File information of influenza.cds, the 5th Block



The Sizes of the File Blocks

- Go with the default block size 128 MB/block.
- File size = 1.01 GB (influenza.cds).
  - Block 1, 128 MB
  - Block 2, 128 MB
  - Block 3, 128 MB
  - Block 4, 128 MB
  - Block 5, 128 MB
  - Block 6, 128 MB
  - Block 7, 128 MB
  - Block 8, 8 MB
- The last HDFS block only takes the storage of the actual size (8 MB), not the default block size.

Can Also run a Command to See the # of Blocks

hadoop fsck /stsci5065/data -files -blocks | less

```
[root@sandbox ~]# hdfs fsck /stsci5065/data -files -blocks
Connecting to namenode via http://sandbox.hortonworks.com:50070/fsck?ugi=root&files=1&block
s=1&path=%2Fstsci5065%2Fdata
FSCK started by root (auth:SIMPLE) from /172.17.0.2 for path /stsci5065/data at Fri Feb 16
00:05:56 UTC 2018
/stsci5065/data <dir>
/stsci5065/data/100-0.txt 5856576 bytes, 1 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742636_1813 len=5856576 repl=1

/stsci5065/data/influenza.cds 1082513347 bytes, 9 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742637_1814 len=134217728 repl=1
1. BP-1281279544-172.17.0.2-1501250400082:blk_1073742638_1815 len=134217728 repl=1
2. BP-1281279544-172.17.0.2-1501250400082:blk_1073742639_1816 len=134217728 repl=1
3. BP-1281279544-172.17.0.2-1501250400082:blk_1073742640_1817 len=134217728 repl=1
4. BP-1281279544-172.17.0.2-1501250400082:blk_1073742641_1818 len=134217728 repl=1
5. BP-1281279544-172.17.0.2-1501250400082:blk_1073742642_1819 len=134217728 repl=1
6. BP-1281279544-172.17.0.2-1501250400082:blk_1073742643_1820 len=134217728 repl=1
7. BP-1281279544-172.17.0.2-1501250400082:blk_1073742644_1821 len=134217728 repl=1
8. BP-1281279544-172.17.0.2-1501250400082:blk_1073742645_1822 len=8771523 repl=1

/stsci5065/data/tom_sawyer.txt 433025 bytes, 1 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742646_1823 len=433025 repl=1
```

Second Page of Output

```
Status: HEALTHY
 Total size:    1088802948 B
 Total dirs:    1
 Total files:   3
 Total symlinks:                0
 Total blocks (validated):      11 (avg. block size 98982086 B)
 Minimally replicated blocks:   11 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1
FSCK ended at Fri Feb 16 00:05:56 UTC 2018 in 1 milliseconds


The filesystem under path '/stsci5065/data' is HEALTHY
```

Copy a File from HDFS to Local Filesystem

- hadoop fs **-copyToLocal** /stsci5065/data/100-0.txt 100-0.copy.txt

  (copy it to the **current** directory, in this case /root)

```
[root@sandbox ~]# hadoop fs -copyToLocal /stsci5065/data/100-0.txt 100-0.copy.txt
[root@sandbox ~]# ls
100-0.copy.txt  anaconda-ks.cfg  hdp              install.log.syslog  start_hbase.sh
100-0.txt       blueprint.json   influenza.cds    sandbox.info        tom_sawyer.txt
74-0.txt        build.out        install.log      start_ambari.sh
```

- hadoop fs **-copyToLocal** /stsci5065/data/tom_sawyer / t_s.copy.txt

  (copy it to the system root)

```
[root@sandbox ~]# hadoop fs -copyToLocal /stsci5065/data/tom_sawyer.txt /t_s.copy.txt
[root@sandbox ~]# ls /
bin                                  hadoop      opt             sys
boot                                 home        packer-files    tmp
cgroups_test                         kafka-logs  proc            t_s.copy.txt
dev                                  lib         root            usr
doSet_version1501251577745249.json   lib64       sandbox-flavor  vagrant
doSet_version1501251578123737.json   lost+found  sbin            var
doSet_version1501251578334395.json   media       selinux
etc                                  mnt         srv
[root@sandbox ~]#
```

Compare t_s.copy.txt with tom_sawyer.txt

- **openssl dgst -md5** /t_s.copy.txt ~/tom_sawyer.txt

```
[root@sandbox ~]# openssl dgst -md5 /t_s.copy.txt  ~/tom_sawyer.txt
MD5(/t_s.copy.txt)= 035f3501a3502554264bf0171f3af8fe
MD5(/root/tom_sawyer.txt)= 035f3501a3502554264bf0171f3af8fe
[root@sandbox ~]#
```

(the same output values of the command indicate these two files are the same, i.e., the file survived to HDFS and is back intact)

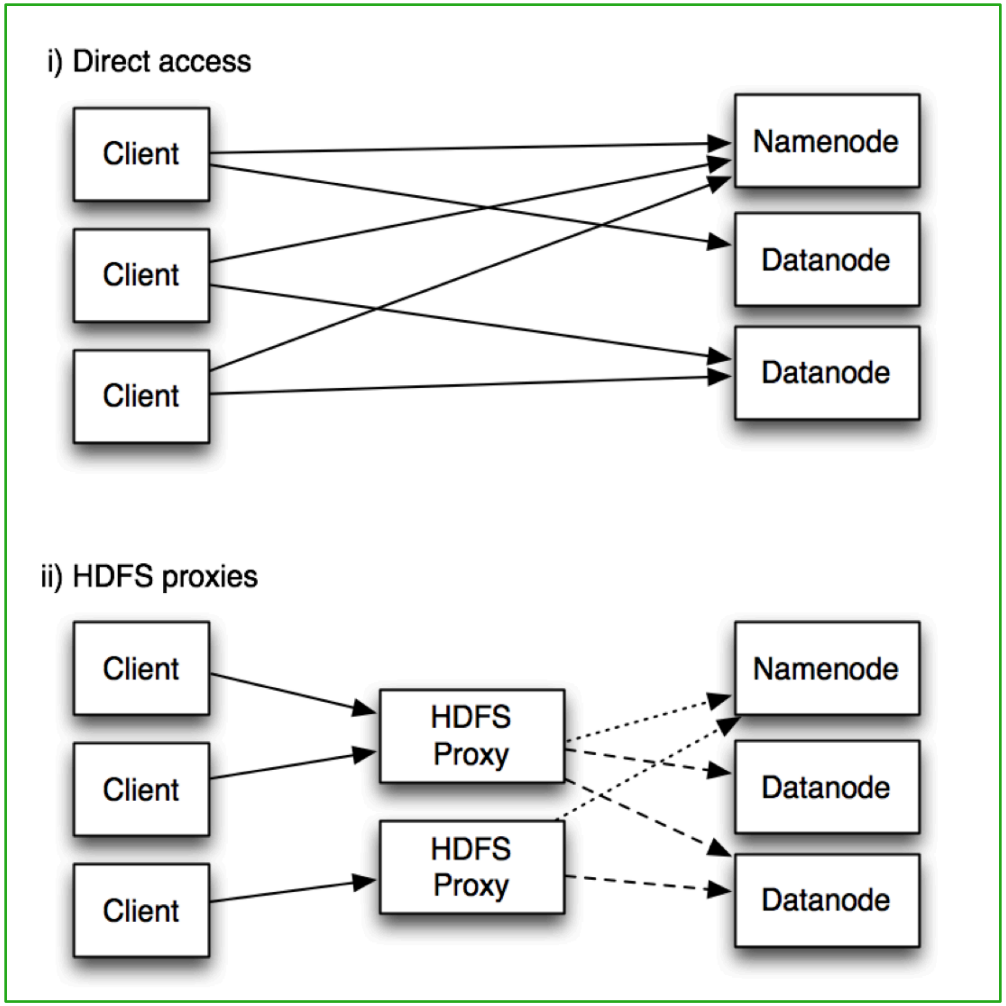
Hadoop Filesystems

- Hadoop has an abstract notion of filesystem, and HDFS is just one implementation. Some are listed below.
  - Local — for a locally connected disk with client-side checksums
  - HDFS — Hadoop's distributed filesystem
  - HFTP — providing read-only access to HDFS over HTTP
  - HSFTP — providing read-only access to HDFS over HTTPS
  - FTP — backed by an FTP server
  - S3(native) — backed by Amazon S3
  - S3(block-based) — backed by Amazon S3

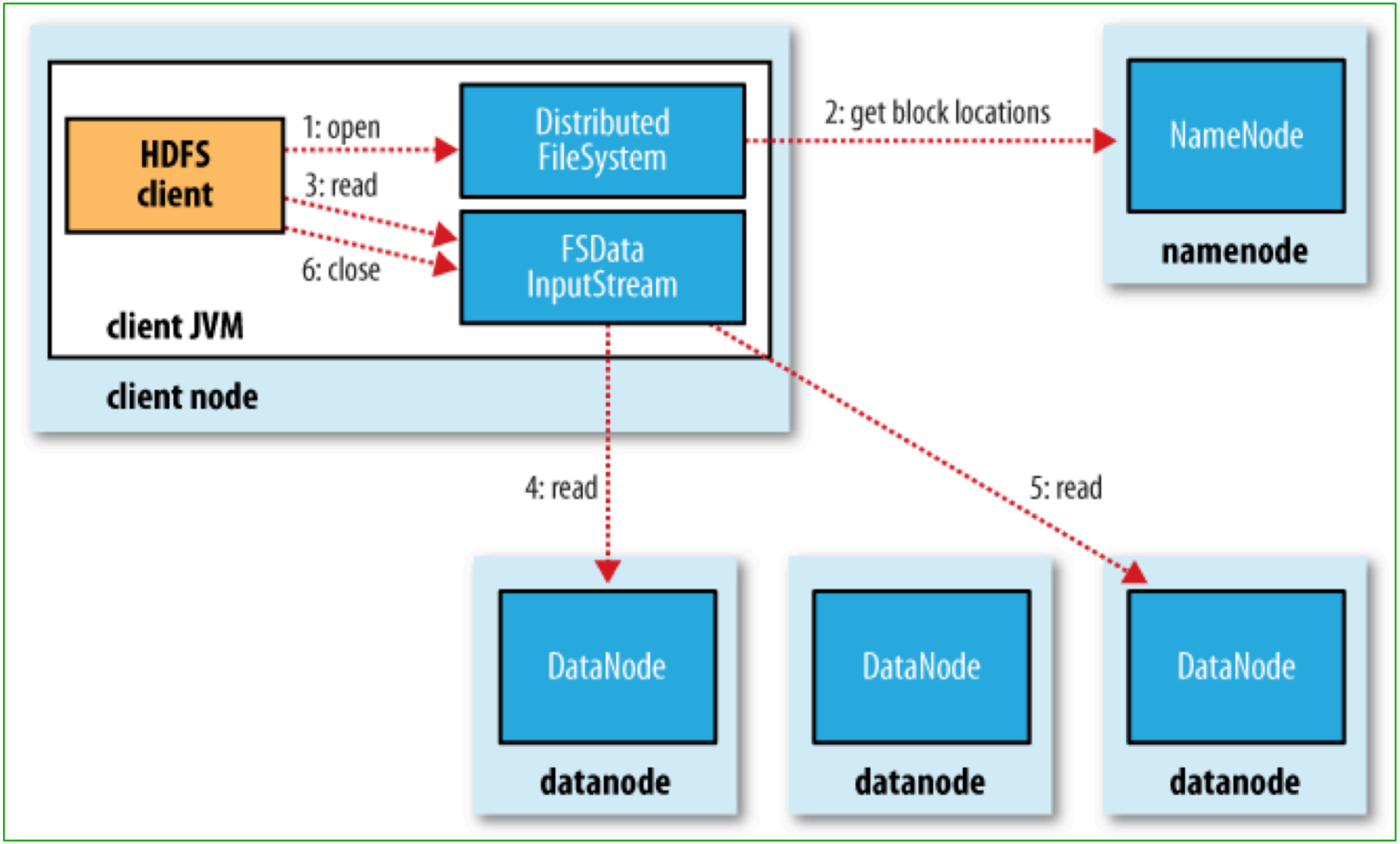
Interfaces to Hadoop's Filesystems: HTTP

All Hadoop filesystem interactions are mediated through the Java API.

- HTTP: There are two ways of accessing HDFS over HTTP:
  - directly, where the HDFS daemons serve HTTP requests to clients through ports **50070** (namenode) and **50075** (datanodes);
  - via a proxy (or proxies), which accesses HDFS on the client's behalf using the DistributedFileSystem API.



A Client Reading Data from HDFS



FUSE: Filesystem in Userspace

- FUSE allows filesystems that are implemented in user space to be integrated as a Unix filesystem. Hadoop's Fuse-DFS contrib module allows Hadoop filesystem (typically HDFS) to be mounted as a standard filesystem. You can then use Unix utilities (such as ls and cat) to interact with the filesystem, as well as POSIX libraries to access the filesystem from any programming language.

HDFS 2.x Features

- NameNode High-Availability (HA).
  - Two redundant NameNodes in active/passive configuration.
  - Manual or automated failover (through Zookeeper).
- NameNode Federation.
  - Multiple independent NameNodes using the same collection of DataNodes.