Distributed Filesystems

Amir H. Payberah
Swedish Institute of Computer Science

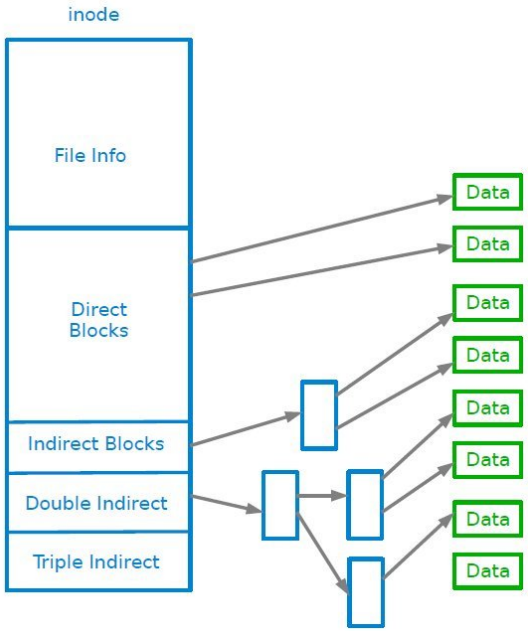
amir@sics.se
April 8, 2014

What is Filesystem?

- Controls how data is stored in and retrieved from disk.

What is Filesystem?

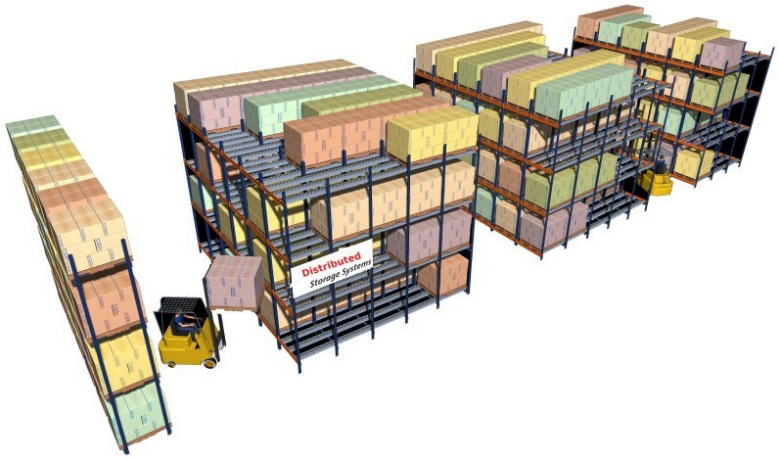
- Controls how data is stored in and retrieved from disk.



Distributed Filesystems

- When data outgrows the storage capacity of a single machine: partition it across a number of separate machines.

- Distributed filesystems: manage the storage across a network of machines.




hadoop
HDFS

HDFS

- Hadoop Distributed FileSystem
- Appears as a single disk
- Runs on top of a native filesystem, e.g., ext3
- Fault tolerant: can handle disk crashes, machine crashes, ...
- Based on Google's filesystem GFS

HDFS is Good for ...

- Storing large files
  - Terabytes, Petabytes, etc...
  - 100MB or more per file.
- Streaming data access
  - Data is written once and read many times.
  - Optimized for batch reads rather than random reads.
- Cheap commodity hardware
  - No need for super-computers, use less reliable commodity hardware.

HDFS is Not Good for ...

- Low-latency reads
  - High-throughput rather than low latency for small chunks of data.
  - HBase addresses this issue.
- Large amount of small files
  - Better for millions of large files instead of billions of small files.
- Multiple writers
  - Single writer per file.
  - Writes only at the end of file, no-support for arbitrary offset.

HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.

- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.

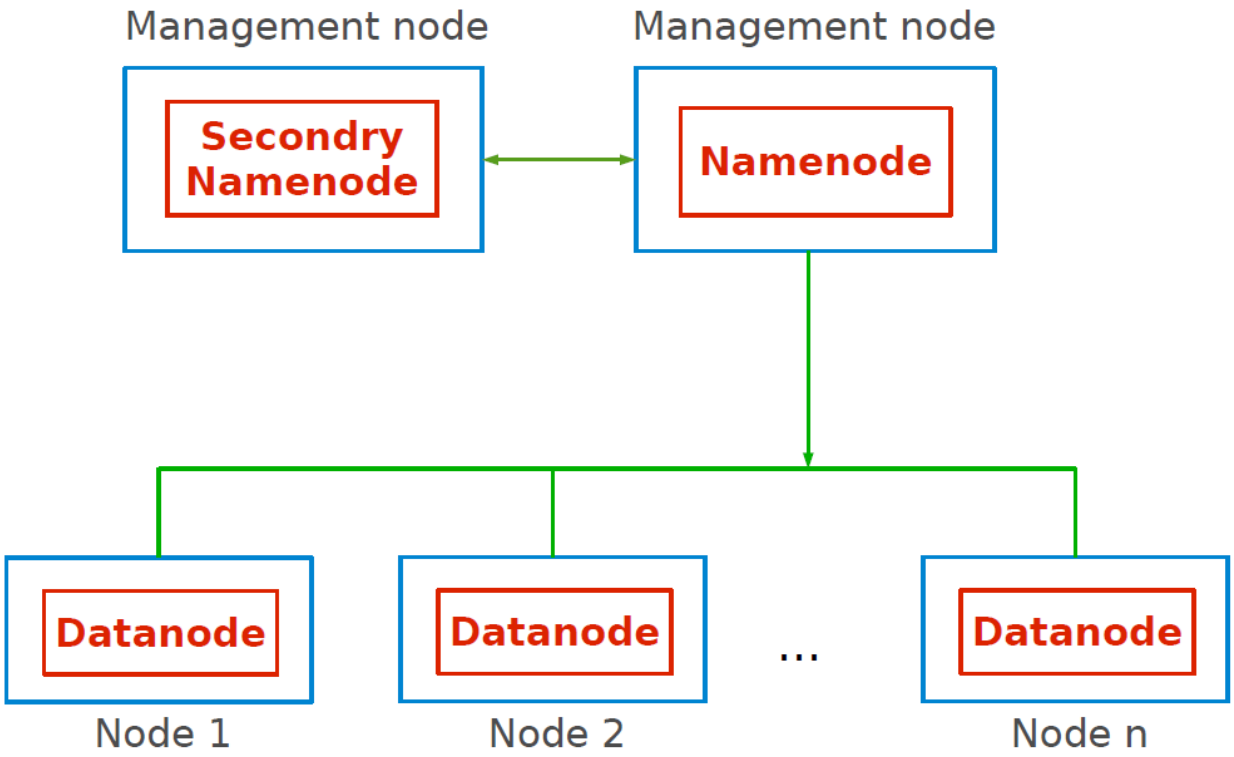
HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.
- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.
- Datanode
  - Stores and retrieves data blocks
  - Reports to Namenode
  - Runs on many machines

HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.
- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.
- Datanode
  - Stores and retrieves data blocks
  - Reports to Namenode
  - Runs on many machines
- Secondary Namenode
  - Only for checkpointing.
  - Not a backup for Namenode

HDFS Daemons (2/2)



Files and Blocks (1/3)

- Files are split into blocks.

- Blocks
  - Single unit of storage: a contiguous piece of information on a disk.
  - Transparent to user.
  - Managed by Namenode, stored by Datanode.
  - Blocks are traditionally either 64MB or 128MB: default is 64MB.

Files and Blocks (2/3)

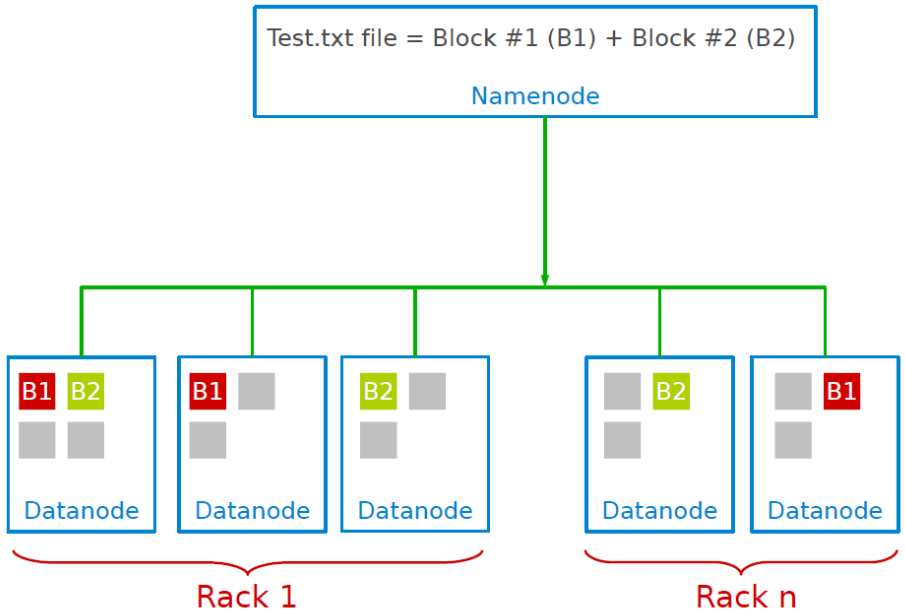
- Why is a block in HDFS so large?

Files and Blocks (2/3)

- Why is a block in HDFS so large?
  - To minimize the cost of seeks.
- Time to read a block = seek time + transfer time
- Keeping the ratio $\frac{seektime}{transfertime}$ small: we are reading data from the disk almost as fast as the physical limit imposed by the disk.
- Example: if seek time is 10ms and the transfer rate is 100MB/s, to make the seek time 1% of the transfer time, we need to make the block size around 100MB.

Files and Blocks (3/3)

- Same block is replicated on multiple machines: default is 3
  - Replica placements are rack aware.
  - 1st replica on the local rack.
  - 2nd replica on the local rack but different machine.
  - 3rd replica on the different rack.
- Namenode determines replica placement.

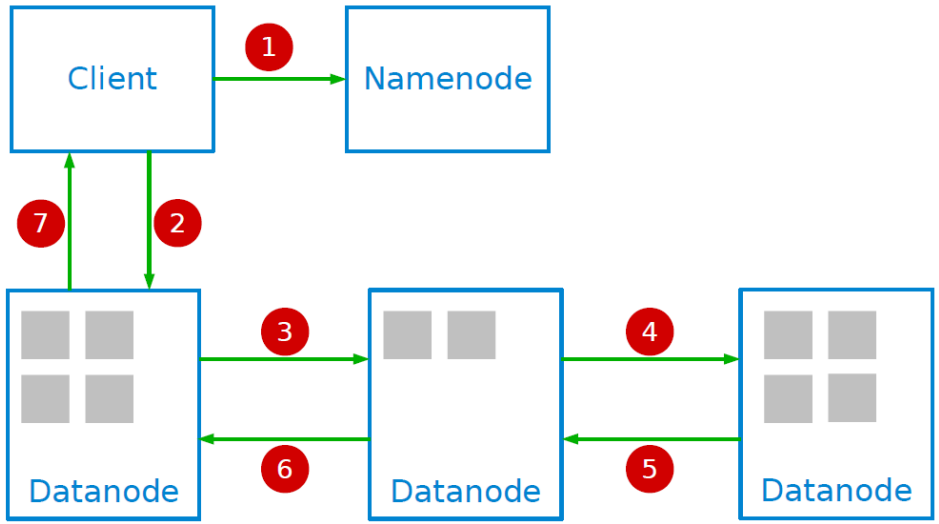


HDFS Client

- Client interacts with Namenode
  - To update the Namenode namespace.
  - To retrieve block locations for writing and reading.
- Client interacts directly with Datanode
  - To read and write data.
- Namenode does not directly write or read data.

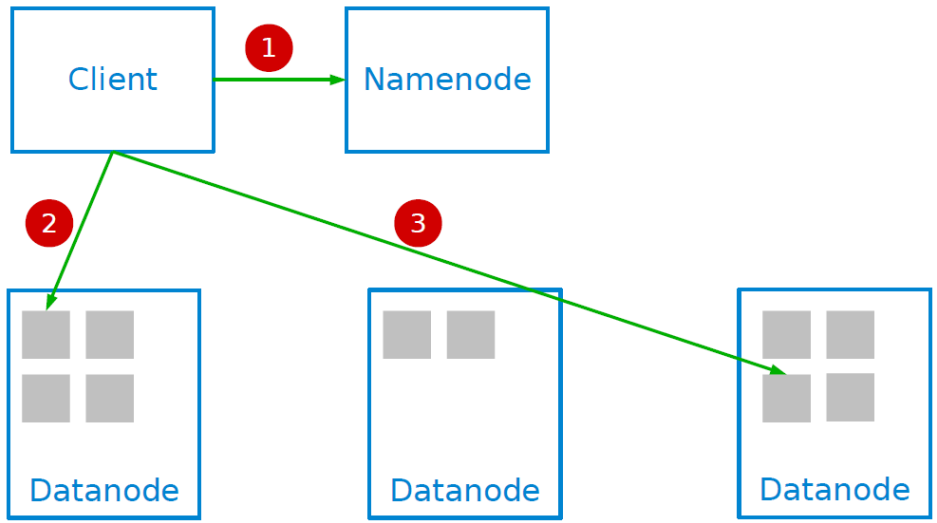
HDFS Write

- 1. Create a new file in the Namenode's Namespace; calculate block topology.
- 2, 3, 4. Stream data to the first, second and third node.
- 5, 6, 7. Success/failure acknowledgment.



HDFS Read

- 1. Retrieve block locations.
- 2, 3. Read blocks to re-assemble the file.

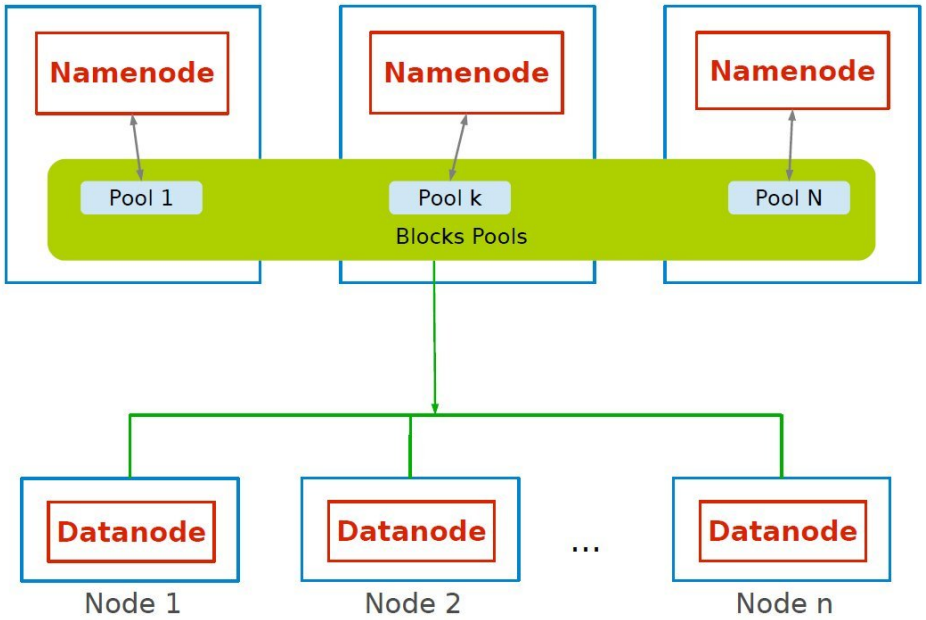


Namenode Memory Concerns

- For fast access Namenode keeps all block metadata in-memory.
  - Will work well for clusters of 100 machines.
- Changing block size will affect how much space a cluster can host.
  - 64MB to 128MB will reduce the number of blocks and increase the space that Namenode can support.

HDFS Federation

- Hadoop 2+
- Each Namenode will host part of the blocks.
- A Block Pool is a set of blocks that belong to a single namespace.
- Support for 1000+ machine clusters.



Namenode Fault-Tolerance (1/2)

- Namenode is a single point of failure.
- If Namenode crashes then cluster is down.

Namenode Fault-Tolerance (1/2)

- Namenode is a single point of failure.
- If Namenode crashes then cluster is down.
- Secondary Namenode periodically merges the namespace image and log and a persistent record of it written to disk (checkpointing).
- But, the state of the secondary Namenode lags that of the primary: does not provide high-availability of the filesystem

Namenode Fault-Tolerance (2/2)

- High availability Namenode.
  - Hadoop 2+
  - Active standby is always running and takes over in case main Namenode fails.

Summary

- Good for large files.
- Streaming access rather than random access.
- Daemons: Namenode, Secondary Namenode, and Datanode

HDFS Installation and Shell

HDFS Installation

- Three options
  - Local (Standalone) Mode
  - Pseudo-Distributed Mode
  - Fully-Distributed Mode

Installation - Local

- Default configuration after the download.
- Executes as a single Java process.
- Works directly with local filesystem.
- Useful for debugging.

Installation - Pseudo-Distributed (1/6)

- Still runs on a single node.
- Each daemon runs in its own Java process.
  - Namenode
  - Secondary Namenode
  - Datanode
- Configuration files:
  - `hadoop-env.sh`
  - `core-site.xml`
  - `hdfs-site.xml`

Installation - Pseudo-Distributed (2/6)

- Specify environment variables in `hadoop-env.sh`

```
export JAVA_HOME=/opt/jdk1.7.0_51
```

Installation - Pseudo-Distributed (3/6)

- Specify location of Namenode in `core-site.sh`

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020</value>
  <description>NameNode URI</description>
</property>
```

Installation - Pseudo-Distributed (4/6)

- Configurations of Namenode in `hdfs-site.sh`
- Path on the local filesystem where the Namenode stores the namespace and transaction logs persistently.

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/namenode</value>
  <description>description...</description>
</property>
```

Installation - Pseudo-Distributed (5/6)

- Configurations of Secondary Namenode in `hdfs-site.sh`
- Path on the local filesystem where the Secondary Namenode stores the temporary images to merge.

```
<property>
  <name>dfs.namenode.checkpoint.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/secondary</value>
  <description>description...</description>
</property>
```

Installation - Pseudo-Distributed (6/6)

- Configurations of Datanode in `hdfs-site.sh`
- Comma separated list of paths on the local filesystem of a Datanode where it should store its blocks.

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/datanode</value>
  <description>description...</description>
</property>
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

- Start the Namenode, Secondary namenode and Datanode daemons.

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start secondarynamenode
hadoop-daemon.sh start datanode
jps
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

- Start the Namenode, Secondary namenode and Datanode daemons.

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start secondarynamenode
hadoop-daemon.sh start datanode
jps
```

- Verify the deamons are running:
  - Namenode: `http://localhost:50070`
  - Secondary Namenode: `http://localhost:50090`
  - Datanode: `http://localhost:50075`

HDFS Shell

```
hdfs dfs -<command> -<option> <path>
```

HDFS Shell

```
hdfs dfs -<command> -<option> <path>
```

```
hdfs dfs -ls /
hdfs dfs -ls file:///home/big
hdfs dfs -ls hdfs://localhost/
hdfs dfs -cat /dir/file.txt
hdfs dfs -cp /dir/file1 /otherDir/file2
hdfs dfs -mv /dir/file1 /dir2/file2
hdfs dfs -mkdir /newDir
hdfs dfs -put file.txt /dir/file.txt # can also use copyFromLocal
hdfs dfs -get /dir/file.txt file.txt # can also use copyToLocal
hdfs dfs -rm /dir/fileToDelete
hdfs dfs -help
```

Questions?