Chapter 3

The Hadoop Distributed Filesystem

Learning Objectives

- HDFS Concepts
- Command-Line Interface
- Java Interface
- Data Flow
- Distcp and Hadoop Archives

Design Criteria of HDFS

Very large files
- Petabyte-scale data

Lots of small files
- Growing of filesystem metadata

Streaming data access
- Write-once, read-many-times
- High throughput

Random data access
- Multiple writers, arbitrary file update
- Low latency

Commodity hardware
- Node failure handling

HDFS Blocks

- Block: the minimum unit of data to read/write/replicate
- Large block size: 64MB by default, 128MB in practice
  - Small metadata volumes, low seek time
- A small file does not occupy a full block
  - HDFS runs on top of underlying filesystem
- Filesystem check (fsck)

```
% hadoop fsck -files -blocks
```

Namenode – The Master

Task	Information Stored
Manage filesystem namespace (i.e., filesystem tree & metadata)	Namespace image, edit log (stored persistently)
Keeping track of all the blocks	Block locations (stored just in memory)

- Single point of failure
  - Backup the persistent metadata files
  - Run a secondary namenode (standby)

Datanodes - Workers

- Datanodes are the workhorse of the filesystem.
- Store and retrieve blocks when they are told to.
- Report the lists of storing blocks to the namenode periodically.

HDFS high availability

- Use a pair of namenodes in an active-standby configuration.
- The standby has both the latest edit log entries and an up-to-date block mapping in memory.
- Failover controller manages the transition from the active namenode to the standby.
- Failover may also be initiated manually by an administrator.

HDFS Configuration

- Pseudo-distributed configuration
  - fs.default.name = hdfs://localhost/
  - dfs.replication = 1
- hdfs://localhost/ is used to set a default filesystem for Hadoop.
- Set dfs.replication = 1 so that HDFS doesn't replicate filesystem blocks by the default factor of 3.

Basic Filesystem Operations – 1

- Copying a file from the local filesystem to HDFS

```
% hadoop fs -copyFromLocal input/docs/quangle.txt
hdfs://localhost/user/tom/quangle.txt
```

- Copying the file back to the local filesystem

```
% hadoop fs -copyToLocal hdfs://localhost/user/tom/quangle.txt
quangle.copy.txt
```

Basic Filesystem Operations – 2

- Check if two files are the same using MD5 message digest

```
% hadoop fs -copyToLocal quangle.txt quangle.copy.txt
% md5 input/docs/quangle.txt quangle.copy.txt
MD5 (input/docs/quangle.txt) = a16f231da6b05e2ba7a339320e7dacd9
MD5 (quangle.copy.txt) = a16f231da6b05e2ba7a339320e7dacd9
```

- Creating a directory

```
% hadoop fs -mkdir books
% hadoop fs -ls .
Found 2 items
drwxr-xr-x - tom supergroup   0 2009-04-02 22:41 /user/tom/books
-rw-r--r-- 1 tom supergroup 118 2009-04-02 22:29 /user/tom/quangle.txt
```

Hadoop Filesystems – 1

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)
Local	*file*	fs.LocalFileSystem
HDFS	*hdfs*	hdfs.DistributedFileSystem
HFTP	*hftp*	hdfs.HftpFileSystem
HSFTP	*hsftp*	hdfs.HsftpFileSystem
HAR	*har*	fs.HarFileSystem
KFS (Cloud-Store)	*kfs*	fs.kfs.KosmosFileSystem
FTP	*ftp*	fs.ftp.FTPFileSystem
S3 (native)	*s3n*	fs.s3native.NativeS3FileSystem
S3 (block-based)	*s3*	fs.s3.S3FileSystem

Hadoop Filesystems – 2

- Listing the files in the root directory of the local filesystem.

```
% hadoop fs -ls file:///
```

- HDFS is just one implementation of Hadoop filesystem.
- You can run MapReduce programs on any of these filesystems, but HDFS is better to process large volumes of data.

Open a Data Stream

- Using java.net.URL object to open a tream to read the data from

```
InputStream in = null;
try {
  in = new URL("hdfs://host/path").openStream();
  // process in
} finally {
  IOUtils.closeStream(in);
}
```

Read Data

- Displaying a file like UNIX cat command

```
public class URLCat {

  static {
    URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());
  }

  public static void main(String[] args) throws Exception {
    InputStream in = null;
    try {
      in = new URL(args[0]).openStream();
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

This method can only be called just once per JVM

A Sample Run

- Output of A Sample Run

```
% hadoop URLCat hdfs://localhost/user/tom/quangle.txt
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

Reading Data Using the FileSystem

```
public class FileSystemCat {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    InputStream in = null;
    try {
      in = fs.open(new Path(uri));
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

FSDataInputStream

- A specialization of java.io.DataInputStream

```
package org.apache.hadoop.fs;

public class FSDataInputStream extends DataInputStream
    implements Seekable, PositionedReadable {
  // implementation elided
}

public interface Seekable {
  void seek(long pos) throws IOException;
  long getPos() throws IOException;
  boolean seekToNewSource(long targetPos) throws IOException;
}
```

Seek – p.58

- Display files on standard output twice by using seek

```
public class FileSystemDoubleCat {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    FSDataInputStream in = null;
    try {
      in = fs.open(new Path(uri));
      IOUtils.copyBytes(in, System.out, 4096, false);
      in.seek(0); // go back to the start of the file
      IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
  }
}
```

Show File Status Test – p.62

- Example 3-5: ShowFileStatusTest

Writing Data

- Copy a local file to a Hadoop filesystem

```
public class FileCopyWithProgress {
  public static void main(String[] args) throws Exception {
    String localSrc = args[0];
    String dst = args[1];

    InputStream in = new BufferedInputStream(new FileInputStream(localSrc));

    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(dst), conf);
    OutputStream out = fs.create(new Path(dst), new Progressable() {
      public void progress() {
        System.out.print(".");
      }
    });

    IOUtils.copyBytes(in, out, 4096, true);
  }
}
```

FSDataOutputStream

- The create() method on FileSystem returns an FSDataOutputStream

```
package org.apache.hadoop.fs;

public class FSDataOutputStream extends DataOutputStream implements Syncable {

  public long getPos() throws IOException {
    // implementation elided
  }

  // implementation elided

}
```

Directories

- Creating a directory

```
public boolean mkdirs(Path f) throws IOException
```

- Often, you don’t need to explicitly create a directory.
- Writing a file will automatically creates any parent directories.

File Metadata: FileStatus

```
@Test
public void fileStatusForFile() throws IOException {
  Path file = new Path("/dir/file");
  FileStatus stat = fs.getFileStatus(file);
  assertThat(stat.getPath().toUri().getPath(), is("/dir/file"));
  assertThat(stat.isDir(), is(false));
  assertThat(stat.getLen(), is(7L));
  assertThat(stat.getModificationTime(),
      is(lessThanOrEqualTo(System.currentTimeMillis())));
  assertThat(stat.getReplication(), is((short) 1));
  assertThat(stat.getBlockSize(), is(64 * 1024 * 1024L));
  assertThat(stat.getOwner(), is("tom"));
  assertThat(stat.getGroup(), is("supergroup"));
  assertThat(stat.getPermission().toString(), is("rw-r--r--"));
}
```

Listing Files

```
public class ListStatus {

  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);

    Path[] paths = new Path[args.length];
    for (int i = 0; i < paths.length; i++) {
      paths[i] = new Path(args[i]);
    }

    FileStatus[] status = fs.listStatus(paths);
    Path[] listedPaths = FileUtil.stat2Paths(status);
    for (Path p : listedPaths) {
      System.out.println(p);
    }
  }
}
```

File Patterns

- Globbing: to use wildcard characters to match multiple files.

```
public FileStatus[] globStatus(Path pathPattern) throws IOException
public FileStatus[] globStatus(Path pathPattern, PathFilter filter) throws IOException
```

- Glob characters and their meanings

Glob	Name	Matches
*	*asterisk*	Matches zero or more characters
?	*question mark*	Matches a single character
[ab]	*character class*	Matches a single character in the set {a, b}
[^ab]	*negated character class*	Matches a single character that is not in the set {a, b}
[a-b]	*character range*	Matches a single character in the (closed) range [a, b], where a is lexicographically less than or equal to b
[^a-b]	*negated character range*	Matches a single character that is not in the (closed) range [a, b], where a is lexicographically less than or equal to b
{a,b}	*alternation*	Matches either expression a or b
\c	*escaped character*	Matches character c when it is a metacharacter

PathFilter

- Allows programmatic control over matching

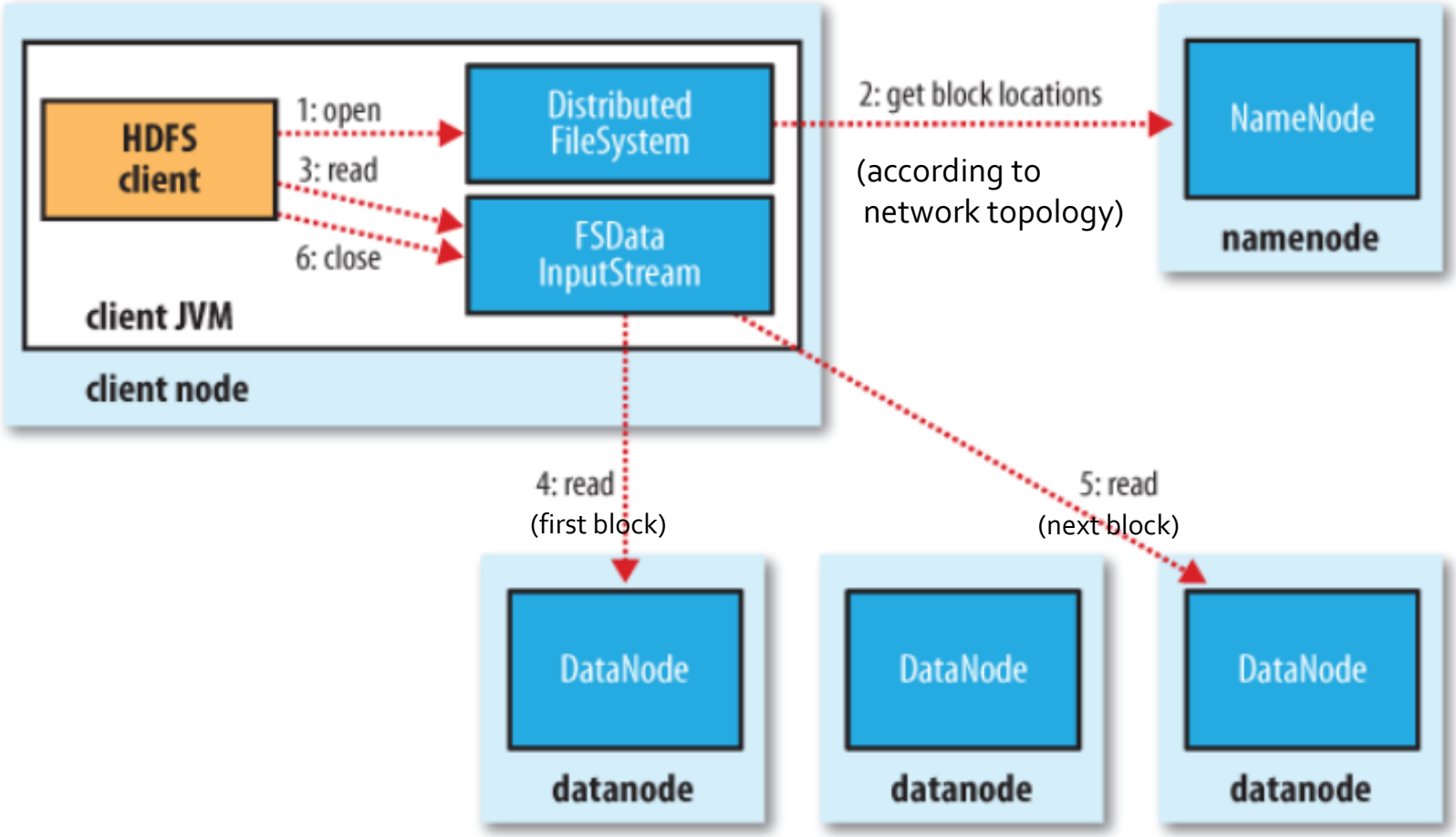
```
package org.apache.hadoop.fs;

public interface PathFilter {
  boolean accept(Path path);
}
```

- PathFilter for excluding paths that match a regex

```
public class RegexExcludePathFilter implements PathFilter {

  private final String regex;

  public RegexExcludePathFilter(String regex) {
    this.regex = regex;
  }

  public boolean accept(Path path) {
    return !path.toString().matches(regex);
  }
}
```

```
fs.globStatus(new Path("/2007/*/*"), new RegexExcludeFilter("^.*/2007/12/31$"))
```

A Client Reading Data From HDFS



Error Handling – 1

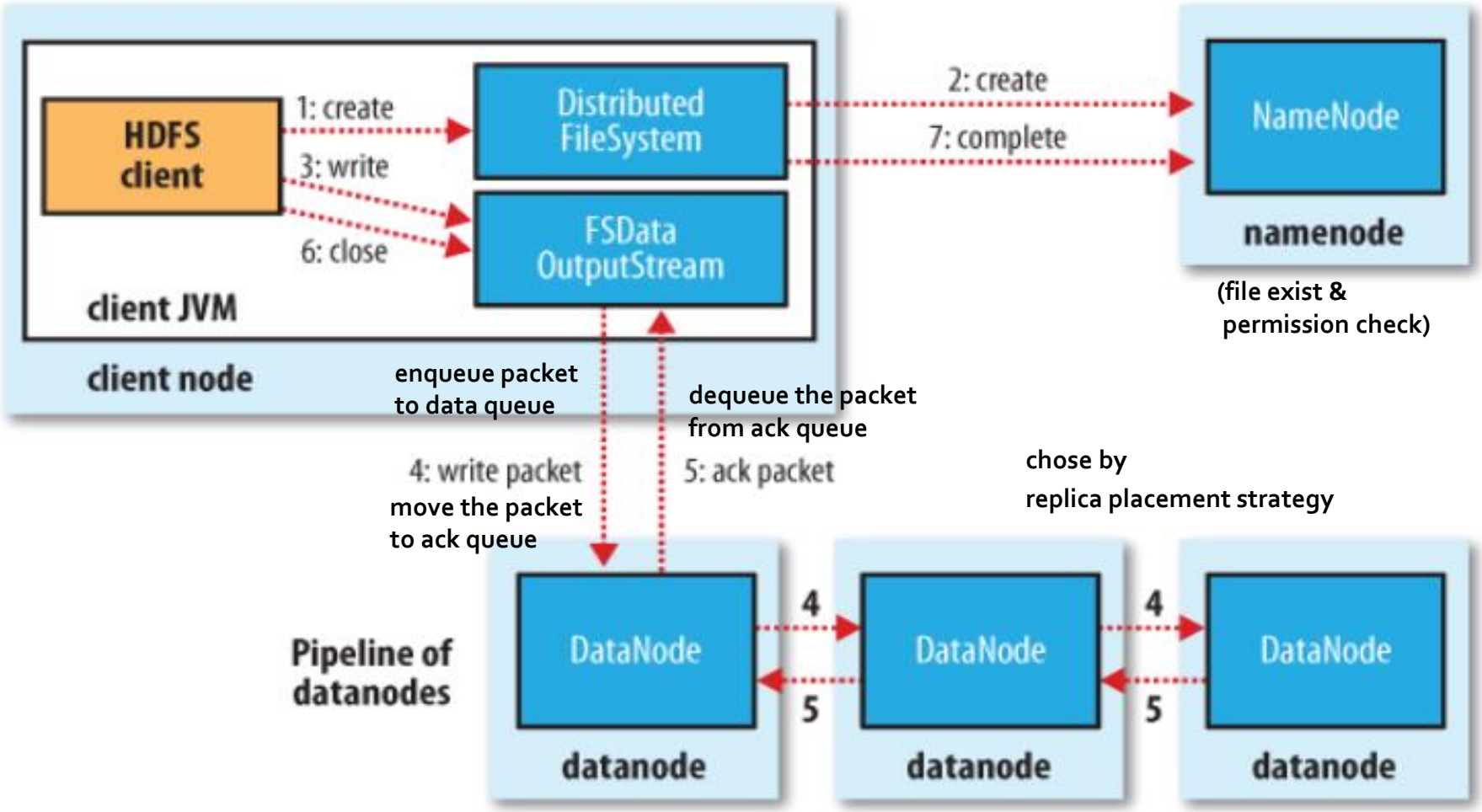
Key point

Client contacts datanodes directly to retrieve data.

Error handling

- Error in client-datanode communication
  - Try next closest datanode for the block
  - Remember failed datanode for later blocks
- Block checksum error
  - Report to the namenode

A Client Writing Data To HDFS



Error Handling – 2

Datanode error while data is being written:

- Client
  - Adds any packets in the *ack* queue to data queue.
  - Removes the failed datanode from the pipeline.
  - Writes the remainder of the data.
- Namenode
  - Arranges under-replicated blocks for further replicas.
- Failed datanode
  - Deletes the partial block when the node recovers later on.

Coherency Model

- A coherency model for a filesystem describes the data visibility of reads and writes for a file.

Key point

The current block being written that is not guaranteed to be visible (even if the stream is flushed).

- HDFS provides sync() method
  - Forcing all buffers to be synchronized to the datanodes.
  - Applications should call sync() at suitable points.

Distcp – Parallel Coping

- Hadoop comes with *distcp* for copying large amounts of data to and from Hadoop filesystems in parallel.

If the clusters are running identical versions of Hadoop, the *hdfs* scheme is appropriate:

```
% hadoop distcp hdfs://namenode1/foo hdfs://namenode2/bar
```

RPC versions are compatible. To repeat the previous example using HFTP:

```
% hadoop distcp hftp://namenode1:50070/foo hdfs://namenode2/bar
```

Hadoop Archives – New Commands

- Hadoop Archives, or HAR files, are a file archiving facility that packs files into HDFS blocks more efficiently.
- It reduces namenode memory usage while still allowing transparent access to files.

```
hadoop fs -ls -R /home/jwu

hadoop archive -archiveName files.har -p /home/jwu input1 input2 /home/jwu

hadoop fs -ls /home/jwu/files.har

hadoop fs -ls -R har:///home/jwu/files.har
```

Hadoop Archives – Old Commands

```
% hadoop fs -lsr /my/files
-rw-r--r--   1 tom supergroup          1 2009-04-09 19:13 /my/files/a
drwxr-xr-x   - tom supergroup          0 2009-04-09 19:13 /my/files/dir
-rw-r--r--   1 tom supergroup          1 2009-04-09 19:13 /my/files/dir/b

% hadoop archive -archiveName files.har /my/files /my

% hadoop fs -ls /my
Found 2 items
drwxr-xr-x   - tom supergroup          0 2009-04-09 19:13 /my/files
drwxr-xr-x   - tom supergroup          0 2009-04-09 19:13 /my/files.har
% hadoop fs -ls /my/files.har
Found 3 items
-rw-r--r--  10 tom supergroup        165 2009-04-09 19:13 /my/files.har/_index
-rw-r--r--  10 tom supergroup         23 2009-04-09 19:13 /my/files.har/_masterindex
-rw-r--r--   1 tom supergroup          2 2009-04-09 19:13 /my/files.har/part-0
```