Topic 5: HDFS Interfaces

1. Hadoop Deployment Modes

Hadoop Cluster Mode

- Deployed in a **cluster of computer nodes**
- Multiple physical or virtual machines
- Production environment
- Full distributed processing

Pseudo-Distributed Mode

- **All services run on a single machine**
- All Java Virtual Machines for Hadoop services on one node
- **Highly simulates full cluster** behavior
- **Lab setting**: Ubuntu 14.04 Virtual Machine

Advantages of Pseudo-Distributed:

- Easy for beginner's practice
- Easy for testing and debugging
- Simulates cluster environment
- Single machine setup

2. Types of HDFS Interfaces

Four Main Interfaces

1. **File System Shell** (Command-Line Interface)
    - Commands: `hadoop fs` or `hdfs dfs`
2. **Hadoop Filesystem Java API**
    - Programmatic access to HDFS
3. **Hadoop Web User Interface**
    - Browser-based access
4. **RESTful Proxy Interfaces**
    - HTTP-based access (e.g., HttpFS)

3. Shell Interface to HDFS

Prerequisites

Bash Shell

```
$ which bash
/bin/bash
```

Hadoop Home Directory

```
$ cd $HADOOP_HOME
$ ls
bin  include  libexec  logs  README.txt  share
etc  lib  LICENSE.txt  NOTICE.txt  sbin
```

Key Directories:

- `bin/`: User-facing scripts and commands
- `sbin/`: System administration scripts
- `share/`: JAR files and libraries

Checking Hadoop Daemons

```
$ jps
28530 SecondaryNameNode
11188 NodeManager
28133 NameNode
28311 DataNode
10845 ResourceManager
3542 Jps
```

Required Daemons:

- NameNode (HDFS master)
- DataNode (HDFS slave)
- SecondaryNameNode (backup)
- ResourceManager (YARN master)
- NodeManager (YARN slave)

Common HDFS Shell Commands

Directory Operations

Create Directory

```
$ hadoop fs -mkdir -p /user/bigdata
$ hadoop fs -mkdir input
```

List Directory Contents

```
$ hadoop fs -ls
Found 1 item
drwxr-xr-x - bigdata supergroup 0 2017-07-17 16:33 input

$ hadoop fs -ls input
-rw-r--r-- 1 bigdata supergroup 1494 2017-07-12 17:53 input/README.txt
```

File Operations

Upload File to HDFS

```
$ hadoop fs -put README.txt input
$ hadoop fs -copyFromLocal README.txt input  # Alternative
```

Read File from HDFS

```
$ hadoop fs -cat input/README.txt
<contents of README.txt displayed>
```

Download File from HDFS

```
$ hadoop fs -get input/README.txt local_file.txt
$ hadoop fs -copyToLocal input/README.txt local_file.txt  # Alternative
```

Delete File

```
$ hadoop fs -rm input/README.txt
```

Delete Directory

```
$ hadoop fs -rm -r input
```

HDFS Path Representation

Full URI Format

```
hdfs://<hostname>:<port>/user/bigdata/input/README.txt
```

Components:

- `hdfs://`: Protocol scheme
- `<hostname>`: NameNode hostname
- `<port>`: NameNode port (default 8020 or 9000)
- `/user/bigdata/`: User directory
- `input/README.txt`: File path

Short Form (Default Settings)

When using default configuration, you can omit hostname, port, and user:

```
# Full form
hadoop fs -ls hdfs://localhost:8020/user/bigdata/input

# Short form (equivalent)
hadoop fs -ls input
```

Frequently Used Commands Reference

Command	Description
`-put`	Upload file(s) from local filesystem to HDFS
`-mkdir`	Create directory in HDFS
`-ls`	List files in HDFS directory
`-cat`	Display content of file(s) in HDFS
`-copyFromLocal`	Copy file from local to HDFS (similar to put)
`-copyToLocal`	Copy file(s) from HDFS to local filesystem
`-get`	Download file(s) from HDFS to local (similar to copyToLocal)
`-rm`	Delete file(s) in HDFS
`-rm -r`	Delete directory in HDFS (recursive)
`-mv`	Move/rename file or directory in HDFS
`-cp`	Copy file or directory in HDFS
`-du`	Display disk usage of files
`-count`	Count number of directories, files, and bytes
`-chmod`	Change file permissions
`-chown`	Change file owner

Command	Description
`-tail`	Display last kilobyte of file
`-touchz`	Create zero-length file
`-setrep`	Change replication factor

4. Web Interface to HDFS

NameNode Web UI

Default URL: `http://localhost:50070` (Hadoop 2.x) or `http://localhost:9870` (Hadoop 3.x)

Features:

- **Overview Tab**
  - Cluster summary
  - NameNode status
  - Storage capacity
  - Live/dead nodes
- **Datanodes Tab**
  - List of all DataNodes
  - Individual node status
  - Storage capacity per node
  - Node configuration
- **Browse Filesystem**
  - Navigate HDFS directory structure
  - View file details
  - Download files
  - View file permissions
- **Logs**
  - NameNode logs
  - System events
  - Error messages
- **Utilities**
  - Snapshot management
  - Block reports
  - Decommission nodes

DataNode Web UI

Default URL: `http://localhost:50075` (Hadoop 2.x) or `http://localhost:9864` (Hadoop 3.x)

Features:

- DataNode information
- Block scanner reports
- Local logs
- Node metrics

5. Java Interface to HDFS

Core Classes and Objects

1. Path Object

```java
// Represents a file or directory in HDFS
Path path = new Path(uri);
Path path = new Path("/user/bigdata/input/file.txt");
```

2. Configuration Object

```java
// Holds Hadoop configuration
Configuration conf = new Configuration();
// Uses default configuration files (core-site.xml, hdfs-site.xml)
```

3. FileSystem Object

Factory Methods:

```java
// Get FileSystem using default configuration
public static FileSystem get(Configuration conf) throws IOException

// Get FileSystem with specific URI
public static FileSystem get(URI uri, Configuration conf) throws IOException

// Get FileSystem with specific user
public static FileSystem get(URI uri, Configuration conf, String user)
    throws IOException

// Get local filesystem
public static FileSystem getLocal(Configuration conf) throws IOException
```

Example:

```java
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(URI.create(uri), conf);
```

Reading Files from HDFS

Open File for Reading

```
// Open with default buffer size
public FSDataInputStream open(Path f) throws IOException

// Open with specific buffer size
public abstract FSDataInputStream open(Path f, int bufferSize)
    throws IOException
```

Complete File Reading Example

```
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);
        FSDataInputStream in = null;
        try {
            Path path = new Path(uri);
            in = fs.open(path);
            IOUtils.copyBytes(in, System.out, 4096, false);
        } finally {
            IOUtils.closeStream(in);
        }
    }
}
```

Writing Files to HDFS

Create File for Writing

```
// Create new file
public FSDataOutputStream create(Path f) throws IOException

// Create with overwrite option
public FSDataOutputStream create(Path f, boolean overwrite)
    throws IOException
```

Simple Write Example

```java
public class FileSystemPut {
    public static void main(String[] args) throws Exception {
        String localStr = args[0];
        String hdfsStr = args[1];

        Configuration conf = new Configuration();
        FileSystem local = FileSystem.getLocal(conf);
        FileSystem hdfs = FileSystem.get(URI.create(hdfsStr), conf);

        Path localFile = new Path(localStr);
        Path hdfsFile = new Path(hdfsStr);

        FSDataInputStream in = local.open(localFile);
        FSDataOutputStream out = hdfs.create(hdfsFile);

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

Buffered Write Example

```java
public class FileSystemPutAlt {
    public static void main(String[] args) throws Exception {
        String localStr = args[0];
        String hdfsStr = args[1];

        Configuration conf = new Configuration();
        FileSystem local = FileSystem.getLocal(conf);
        FileSystem hdfs = FileSystem.get(URI.create(hdfsStr), conf);

        Path localFile = new Path(localStr);
        Path hdfsFile = new Path(hdfsStr);

        FSDataInputStream in = local.open(localFile);
        FSDataOutputStream out = hdfs.create(hdfsFile);

        byte[] buffer = new byte[256];
        int bytesRead = 0;
        while ((bytesRead = in.read(buffer)) > 0) {
            out.write(buffer, 0, bytesRead);
        }

        in.close();
        out.close();
    }
}
```

Compiling and Running Java Programs

Set Classpath

```
export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
javac -cp $HADOOP_CLASSPATH FileSystemCat.java
```

Create JAR File

```
jar cvf FileSystemCat.jar FileSystemCat*.class
```

Run Java Program

```
hadoop jar FileSystemCat.jar FileSystemCat input/README.txt
```

Other FileSystem API Methods

Directory and File Management

```
// Create directories
boolean mkdirs(Path f) throws IOException

// Check if file/directory exists
boolean exists(Path f) throws IOException

// Delete file/directory
boolean delete(Path f, boolean recursive) throws IOException

// Get file status (metadata)
FileStatus getFileStatus(Path f) throws IOException

// List directory contents
FileStatus[] listStatus(Path f) throws IOException

// Rename/move file
boolean rename(Path src, Path dst) throws IOException
```

File Metadata (FileStatus Object)

```
FileStatus status = fs.getFileStatus(path);

status.getPath()            // File path
status.isDirectory()        // Is directory?
status.getLen()             // File size in bytes
status.getModificationTime() // Last modified time
status.getReplication()    // Replication factor
status.getBlockSize()      // Block size
status.getOwner()          // File owner
status.getGroup()          // File group
status.getPermission()     // File permissions
```

Key Points for Exam

Three Main Interface Types

1. **Shell Interface**: Command-line (`hadoop fs`, `hdfs dfs`)
2. **Java API**: Programmatic access (FileSystem, Path, Configuration)
3. **Web UI**: Browser-based (NameNode:50070/9870)

Essential Shell Commands

Must Know:

- `-put` / `-copyFromLocal`: Upload to HDFS
- `-get` / `-copyToLocal`: Download from HDFS
- `-ls`: List files
- `-cat`: Display file
- `-mkdir`: Create directory
- `-rm` / `-rm -r`: Delete file/directory

Java API Core Classes

1. **Configuration**: Holds Hadoop settings
2. **FileSystem**: Main interface for HDFS operations
3. **Path**: Represents file/directory location
4. **FSDataInputStream**: Read from HDFS
5. **FSDataOutputStream**: Write to HDFS

Java API Patterns

Reading:

```
Configuration → FileSystem → Path → open() → FSDataInputStream → read() → close()
```

Writing:

```
Configuration → FileSystem → Path → create() → FSDataOutputStream → write() →
close()
```

URI Format

Full: `hdfs://<hostname>:<port>/user/bigdata/file.txt` **Short**: `file.txt` (uses defaults from configuration)

Web UI Ports

- **NameNode**: 50070 (Hadoop 2.x), 9870 (Hadoop 3.x)
- **DataNode**: 50075 (Hadoop 2.x), 9864 (Hadoop 3.x)
- **ResourceManager**: 8088
- **NodeManager**: 8042

Required Daemons (use `jps`)

1. NameNode
2. DataNode
3. SecondaryNameNode
4. ResourceManager
5. NodeManager

Compilation and Execution

1. Set HADOOP_CLASSPATH
2. Compile with `javac -cp $HADOOP_CLASSPATH`
3. Create JAR with `jar cvf`
4. Run with `hadoop jar`

Key FileSystem Methods

- `get()`: Obtain FileSystem instance
- `open()`: Open file for reading
- `create()`: Create file for writing
- `mkdirs()`: Create directories
- `exists()`: Check existence
- `delete()`: Remove file/directory
- `listStatus()`: List directory contents
- `getFileStatus()`: Get file metadata