CSE535 Asynchronous Systems
Hadoop HDFS

YoungMin Kwon

SUNY Korea
The State University of New York

The Design of HDFS

- Very large files
  - Hadoop clusters store petabytes of data
- Streaming data access
  - Write-once, read-many-times pattern
  - A dataset is created/copied from the source and then various analyses are performed over time
  - Time to read the whole dataset is more important than the latency in reading the first record

SUNY Korea

The Design of HDFS

- Commodity hardware
  - Run on clusters of commodity hardware
  - The chance of node failure is high for large clusters
  - HDFS is designed to carry on working without a noticeable interruption in the face of such failures.

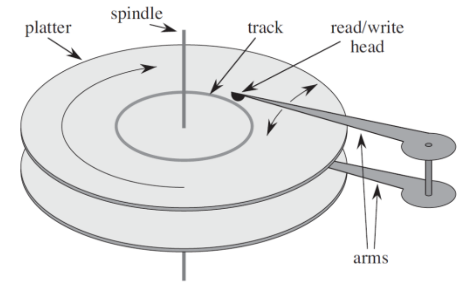
SUNY Korea

HDFS Concepts: Blocks

- Disk block size
  - The minimum amount of data a disk can read or write (typically 512 bytes)
- Filesystem block
  - Integral multiple of the disk block size (typically a few kilobytes)
- HDFS block
  - Much larger unit (64 MB).
  - A file smaller than a block doesn't occupy the whole block
    - 1 MB file does not take the whole 64 MB block

SUNY Korea

Why Large Block



- Minimize seek
  - Make data transfer time longer than seek time
  - Transferring large file with multiple blocks are dominated by the data transfer time
  - e.g.) Suppose that seek time is 10 ms, transfer rate is 100 MB/s. To make seek time 1 % of the transfer time.

    => The block size is around 100 MB.

  - Large blocks may result in fewer tasks: may reduce parallelism.

SUNY Korea

Block Abstraction

- A file can be larger than any single disk in the network
  - A file can be stored in multiple disks
- Simplify the storage subsystem
  - Fixed size block -> easy to calculate how many blocks can be stored on a disk
  - Metadata can be handled separately in a different system
- Fault tolerance and availability
  - Blocks can be easily replicated to other machines
  - Popular files can be replicated more

SUNY Korea

Namenodes

- Two types of nodes
  - Namenodes (master) and Datanode (worker)
- Namenodes: manage the filesystem namespace
  - Filesystem tree and metadata for all the files and directories
  - Namespace is stored in the namespace image and the edit log
  - Does not store block locations persistently
  - Client presents a filesystem interface similar to POSIX.

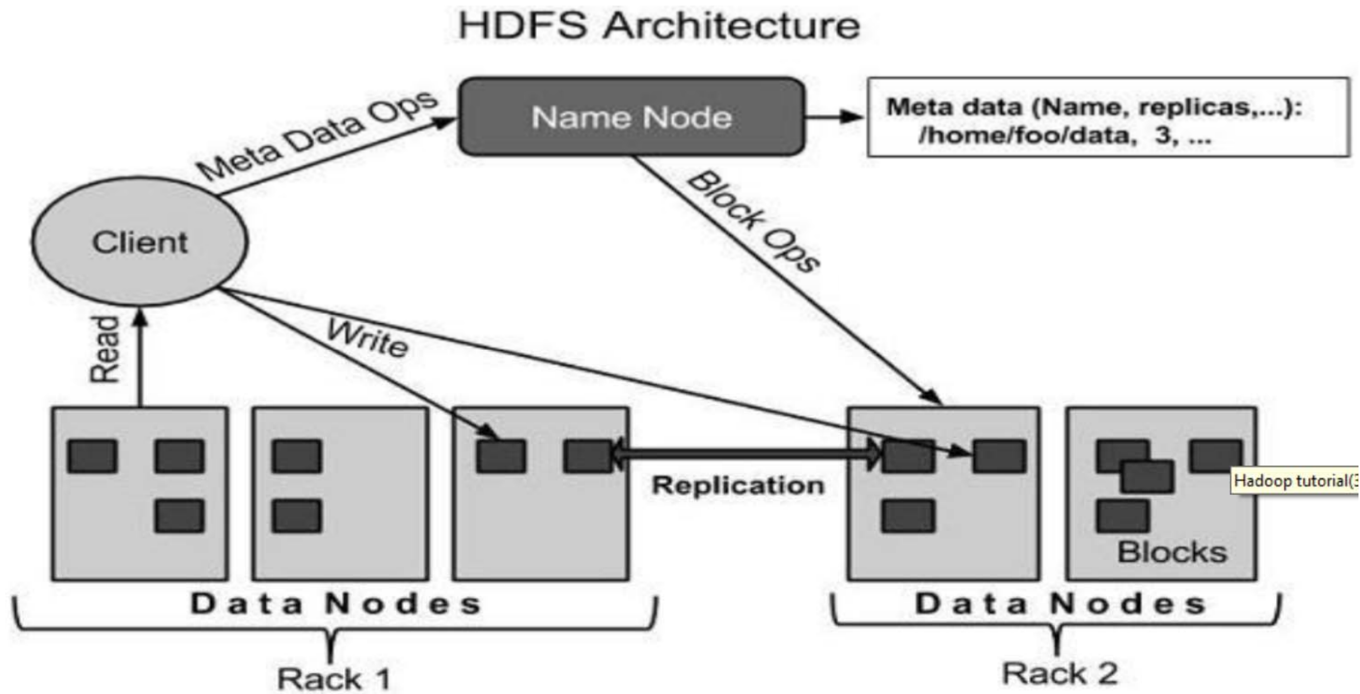
SUNY Korea

Datanodes

- Workhorse of the file system
- Store/retrieve blocks
- Periodically report back to the namenode with the list of blocks they are storing

SUNY Korea

HDFS Architecture



SUNY Korea

HDFS Federation

- Namenode keeps a reference to every file and block in the filesystem in memory
  - Can be a limiting factor for scaling

- HDFS Federation
  - Multiple namenodes
  - Each namenode manages a portion of the filesystem namespace
    - /user, /share, …

SUNY Korea

HDFS High-Availability

- Namenode can be a single point of failure
- To recover from a failure using replica
  - Load namespace image into memory
  - Replay its edit log
  - Receive enough block reports from datanodes
  - The recover process can take as long as 30 min

SUNY Korea

HDFS High-Availability

- A pair of namenodes in an active-standby configuration
- In the event of active node failure, the standby node take over the namenode roles
- Implementation
  - The namenodes share the edit log
  - Datanodes send block reports to both namenodes
  - Clients must be configured to handle namenode failover.

SUNY Korea

HDFS High-Availability

- Failover
  - Failover controller (ZooKeeper): manages the transition from the active namenode to the standby node
- Fencing
  - When it is impossible to be sure that the failed namenode has stopped running
    - Due to slow network, network partition, …
  - Ensure that the previously active namenode does not harm
    - Killing the namenode's process,
    - Revoking its access to the shared storage
    - Disabling its network port
    - STONITH (Shoot The Other Node In The Head): forcibly power down the host machine

SUNY Korea

HDFS Command-Line Interface

- hadoop fs –copyFromLocal foo.txt hdfs://localhost:900000/foo.txt
- hadoop fs -ls hdfs://localhost:900000
- hadoop fs –copyToLocal foo.txt hdfs://localhost:900000/bar.txt
- hadoop fs –rm hdfs://localhost:900000/foo.txt

- hadoop fs -ls /user/hadoop/

- hadoop fs -mkdir hdfs://localhost:9000/user/hadoop/aa
- hadoop fs -rmdir hdfs://localhost:9000/user/hadoop/aa

SUNY Korea

HDFS Java Interface

- Reading Data from a Hadoop URL
  - Use java.net.URL
  - Call `URL.setURLStreamHandlerFactory` with an instance of `FsUrlStreamHandlerFactory`
  - Issue: if other part of the code wants to set `URLStreamHandlerFactory` with a different HandlerFactory, this method won't work

SUNY Korea

```java
import java.net.URL;
import java.io.InputStream;
import org.apache.hadoop.fs.FsUrlStreamHandlerFactory;
import org.apache.hadoop.io.IOUtils;
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);
        }
    }
}
```

```
hadoop jar URLCat.jar URLCat hdfs://localhost:9000/input/sample2.txt
```

SUNY Korea

HDFS Java Interface

- Reading Data Using the FileSystem API
  - A file in a Hadoop filesystem is represented by a Hadoop Path object
  - A path example
    - hdfs://localhost/input/foo.txt

SUNY Korea

Java Interface

- To get a FileSystem instance
  - `public static FileSystem get(URI uri, Configuration conf, String user) throws IOException`
  - conf: client or server's configuration (core-site.xml)
  - uri: specifies the URI scheme
  - user: for security

- With a FileSystem instance, we can invoke open method to get the input stream
  - `public FSDataInputStream open(Path f) throws IOException`

SUNY Korea

```java
import java.net.URI;
import java.io.InputStream;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
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);
        }
    }
}
```

```
hadoop jar URLCat.jar FileSystemCat hdfs://localhost:9000/input/sample2.txt
```

SUNY Korea

Java Interface: Writing Data

- FSDataOutputStream
  - FSDataOutputStream does not support the seek operation (FSDataInputStream supports seek)

- APIs to get FSDataOutputStream

```
public FSDataOutputSream create(Path f)
throws IOException;
public FSDataOutputSream append(Path f)
throws IOException;
```

SUNY Korea

Java Interface: Writing Data

- create and append methods support Progressable interface

- Progressable interface

```
public interface Progressable {
    public void progress();
}
```

SUNY Korea

```java
import java.net.URI;
import java.io.InputStream;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.OutputStream;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.Progressable;

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);
    }
}
```

```
hadoop jar URLCat.jar FileCopyWithProgress ./tmp.txt hdfs://localhost:9000/input/sample3.txt
```

SUNY Korea

Java Interface: Directories

- FileSystem provides mkdirs method to create a directory

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

- Implicitly, create will create any parent directories as well

SUNY Korea

Java Interface: File Status

- FileStatus class encapsulates file system metadata for files and directories.
- `getFileStatus(Path f)` on FileSystem provides a way to get FileStatus object

SUNY Korea

```java
import java.net.URI;
import java.io.InputStream;
import java.util.Date;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileStatus;

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

        System.out.println("getPath:             " + stat.getPath());
        System.out.println("isDir:               " + stat.isDir());
        System.out.println("getLen:              " + stat.getLen());
        System.out.println("getModificationTime: " +
                              new Date(stat.getModificationTime()));
        System.out.println("getReplication:      " + stat.getReplication());
        System.out.println("getBlockSize:        " + stat.getBlockSize());
        System.out.println("getOwner:            " + stat.getOwner());
        System.out.println("getGroup:            " + stat.getGroup());
        System.out.println("getPermission:       " + stat.getPermission());
    }
}
```

SUNY Korea

```
hadoop jar URLCat.jar FileMetadata hdfs://localhost:9000/input/sample2.txt
getPath:              hdfs://localhost:9000/input/sample2.txt
isDir:                false
getLen:               19530
getModificationTime: Mon Apr 10 18:08:39 PDT 2017
getReplication:       1
getBlockSize:         134217728 (0x8000000)
getOwner:             hadoop
getGroup:             supergroup
getPermission:        rw-r--r—
```

```
hadoop jar URLCat.jar FileMetadata hdfs://localhost:9000/input
getPath:              hdfs://localhost:9000/input
isDir:                true
getLen:               0
getModificationTime: Mon Apr 17 01:21:08 PDT 2017
getReplication:       0
getBlockSize:         0
getOwner:             hadoop
getGroup:             supergroup
getPermission:        rwxr-xr-x
```

Listing Files

- How to list the contents of a directory

```
public FileStatus[] listStatus(Path f) throws IOException;
public FileStatus[] listStatus(Path f, PathFilter filter)
throws IOException;
public FileStatus[] listStatus(Path[] files) throws
IOException;
public FileStatus[] listStatus(Path[] files, PathFilter
filter) throws IOException;
```

  - When the argument is a file, it returns an array of length 1
  - When the argument is a directory, it returns zero or more FileStatus representing the files and the directories contained in the directory

SUNY Korea

```java
import java.net.URI;
import java.io.InputStream;
import java.util.Date;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileUtil;

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);
    }
}
```

SUNY Korea

```
hadoop jar URLCat.jar ListStatus hdfs://localhost:9000/input
hdfs://localhost:9000/input/sample.txt
hdfs://localhost:9000/input/sample2.txt
```

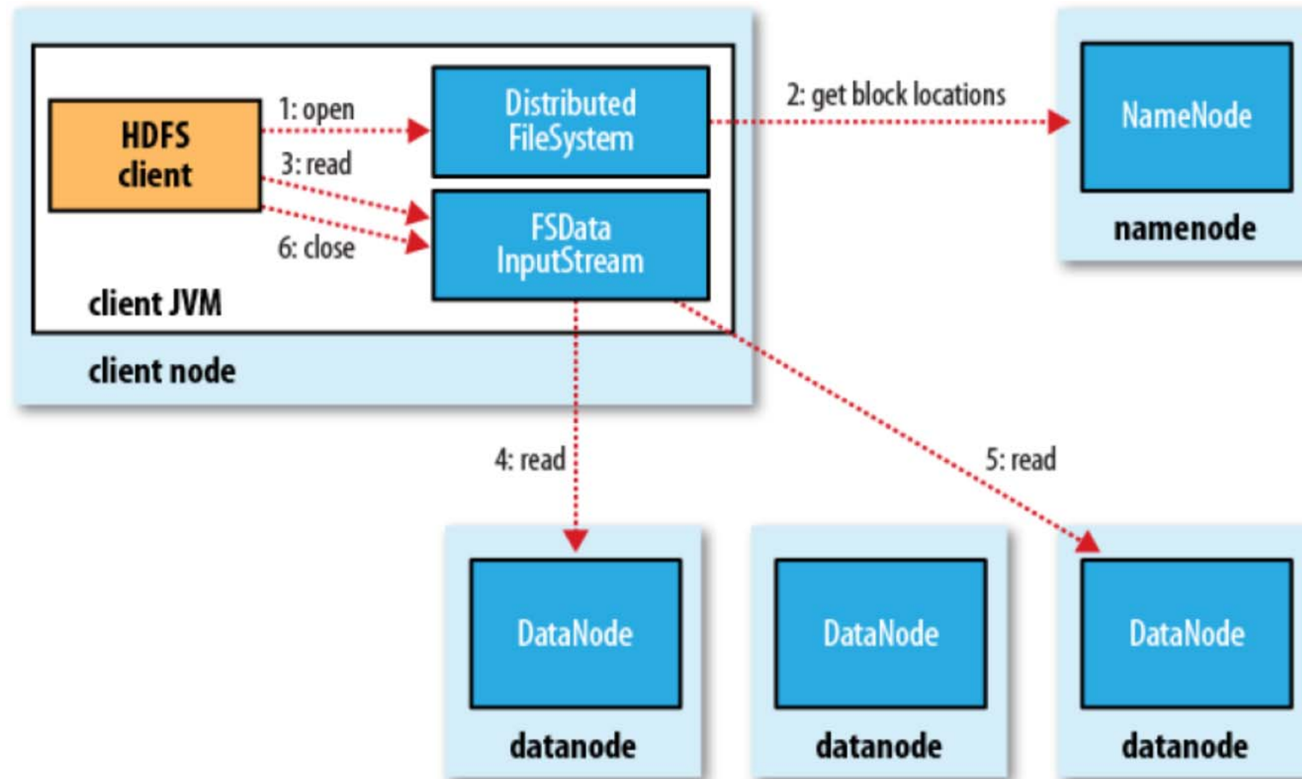
- Deleting Data

  ```
  public boolean delete(Path f, boolean recursive)
  throws IOException
  ```

  - If f is a file or an empty directory recursive is ignored
  - A nonempty directory is deleted along with its contents only if recursive is true

SUNY Korea

Data Flow: File Read



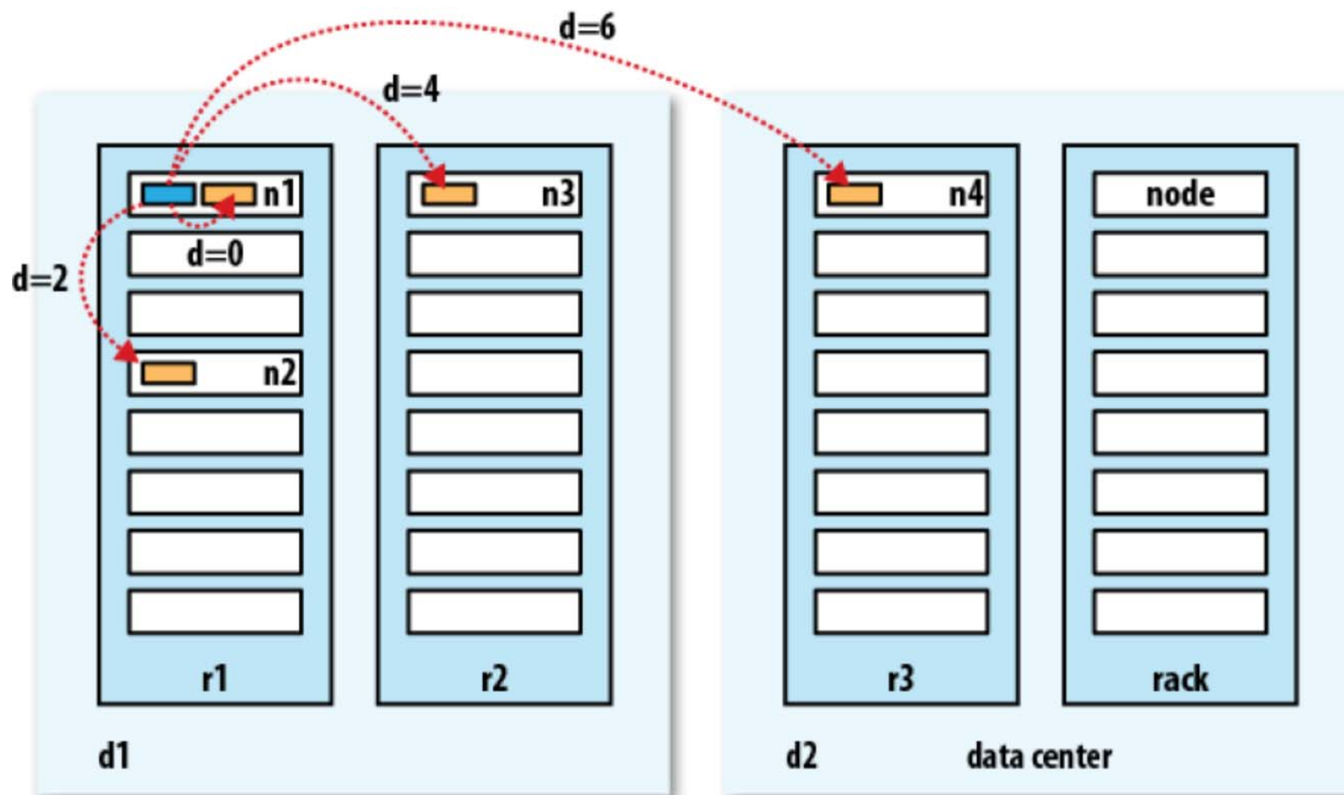
SUNY Korea

Network Topology and Hadoop

- The bandwidth available for each of the following scenarios becomes progressively less
  - Process on the same node
  - Different nodes on the same rack
  - Nodes on different racks in the same data center
  - Nodes in different data centers
- E.g.
  - distance(/d1/r1/n1, /d1/r1/n1) = 0
  - distance(/d1/r1/n1, /d1/r1/n2) = 2
  - distance(/d1/r1/n1, /d1/r2/n3) = 4
  - distance(/d1/r1/n1, /d2/r3/n4) = 6

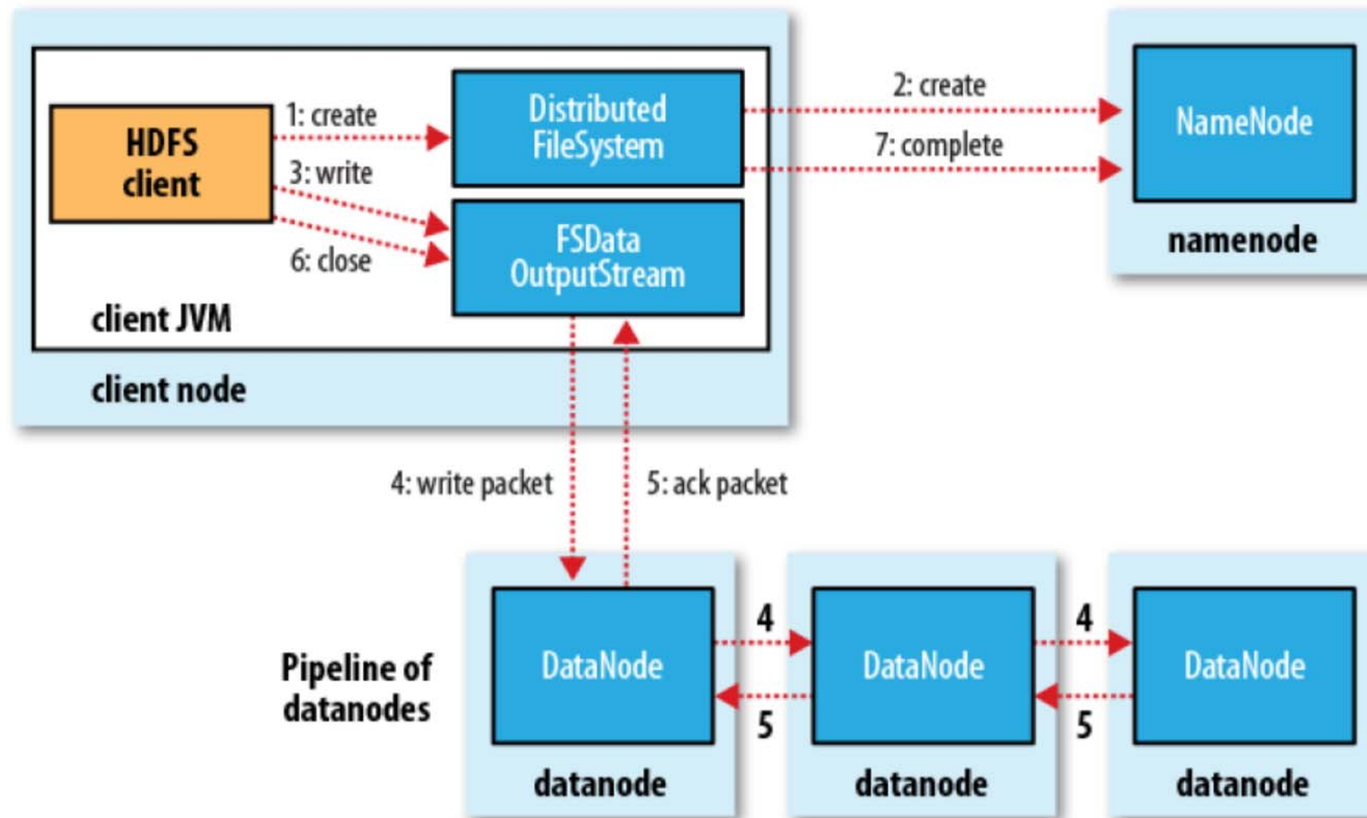
SUNY Korea

Network Topology and Hadoop



SUNY Korea

Data Flow: File Write



SUNY Korea