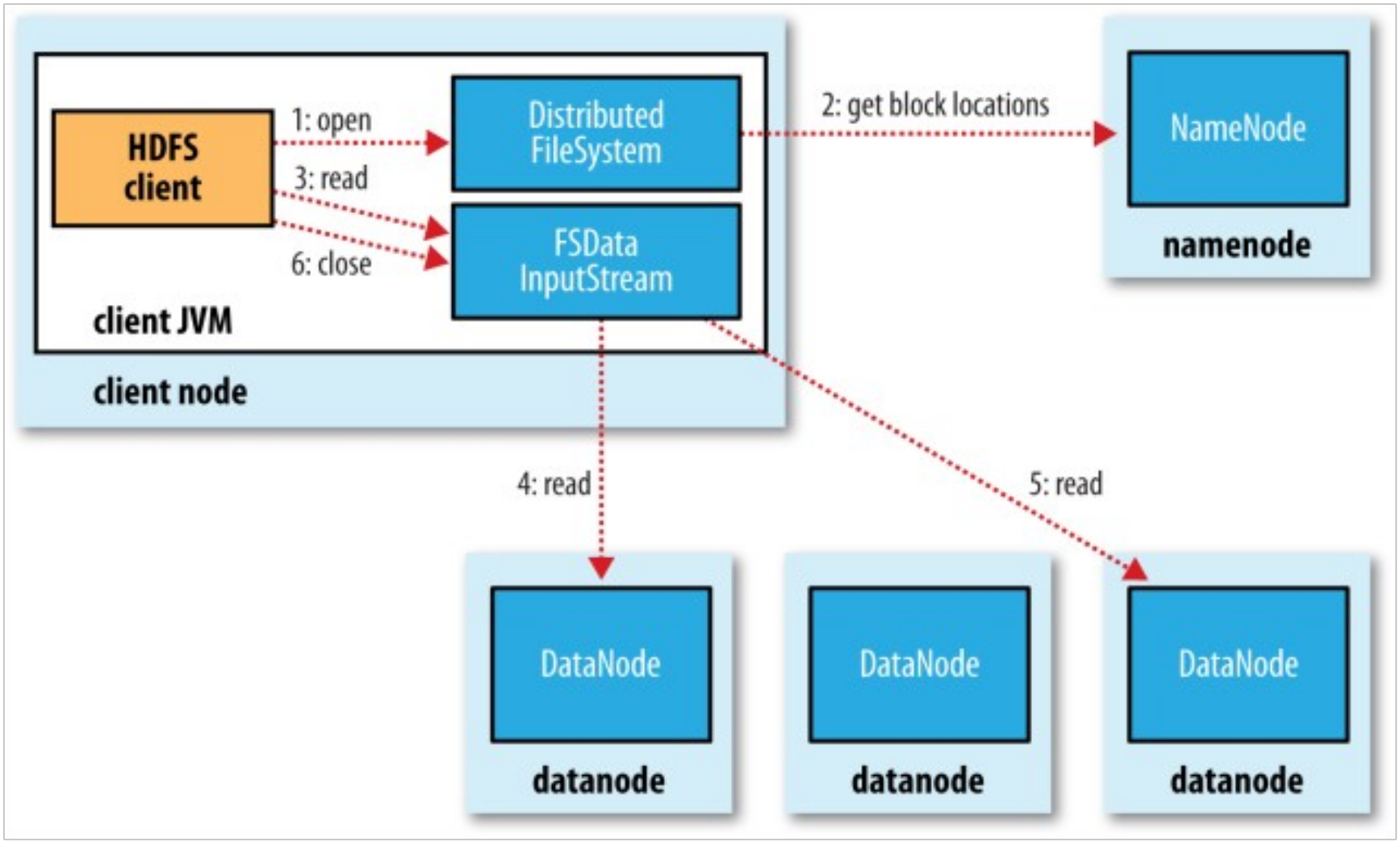
Java API for HDFS

Agenda

- File Read in Hadoop
- Reading using Hadoop URI
- File System API

Data Flow of a File Read in Hadoop



File Read in Hadoop

- **Step-1**
  - The client opens the file it wishes to read by calling open() on the FileSystem object.
- **Step-2**
  - DistributedFileSystem calls the namenode, using(RPCs), to determine the locations of the first few blocks in the file.
  - For each block, the namenode returns the addresses of the datanodes that have a copy of that block..
  - The DistributedFileSystem returns an FSDataInputStream to the client for it to read data from.
  - FSDataInputStream wraps a DFSInputStream, which manages the datanode and namenode I/O

File Read in Hadoop

- **Step-3**
  - The client then calls read() on the stream.
  - DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first(closest) datanode for the first block in the file.

File Read in Hadoop

- **Step-4**
  - Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream.
- **Step-5**
  - When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block.

File Read in Hadoop

- **Step-6**
  - When the client has finished reading, it calls close() on the FSDataInputStream.

Reading Data from a Hadoop URL

- **Using java.net.URL object to open a stream to read the data from.**
  - InputStream in = null;
  - try {
  - in = new URL("hdfs://host/path").openStream();
  - // process in
  - } finally {
  - IOUtils.closeStream(in);
  - }
- **setURLStreamHandlerFactory() method on URL helps Java to recognize Hadoop's hdfs URL with an instance of FsUrlStreamHandlerFactory.**
- **Can be called only once per JVM, so it is typically executed in a static block.**
- **If some other part of your program sets a URLStreamHandlerFactory, we won't be able to use this for reading data from Hadoop.**

Reading Data from a Hadoop URL

```
public class HadoopURLCat {
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);
}
}
}
```

Reading Data from a Hadoop URL

- **IOUtils class which comes with Hadoop is used**
  - For copying bytes between the input stream and the output stream (System.out)
  - For closing the stream in the finally clause.
- **CopyBytes() method arguments**
  - InputStream
  - OutputStream
  - buffer size used for copying
  - whether to close the streams when the copy is complete.

Reading Data Using the FileSystem API

- It is not possible to set a URLStreamHandlerFactory always.
- FileSystem API is used to open an input stream for a file.
- A file in a Hadoop filesystem is represented by a Hadoop Path object.

Reading Data Using the FileSystem API

- **First step is to retrieve an instance for the filesystem—HDFS, in this case.**
  - Default
    - public static FileSystem get(Configuration conf) throws IOException
  - Using URI
    - public static FileSystem get(URI uri, Configuration conf) throws IOException
  - Given by user(good for security)
    - public static FileSystem get(URI uri, Configuration conf, String user) throws IOException

Reading Data Using the FileSystem API

- **A Configuration object encapsulates a client or server's configuration,it read configuration files as etc/hadoop/core-site.xml.**
- **For local filesystem instance**
  - public static LocalFileSystem getLocal(Configuration conf) throws IOException

Reading Data Using the FileSystem API

- **Second step is to invoke an open() method to get the input stream for a file:**
  - Using defalt buffer(4 kb)
    - public FSDataInputStream open(Path f) throws IOException
  - User will supply buffer
    - public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException

Reading Data Using the FileSystem API

```
public class HadoopFileSystemCat {
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);
}
}
}
```

Resources

- **Hadoop: The Definitive Guide**
  - Tom White (Author)
  - O'Reilly Media; 4th Edition.