HDFS

Protocols and Applications

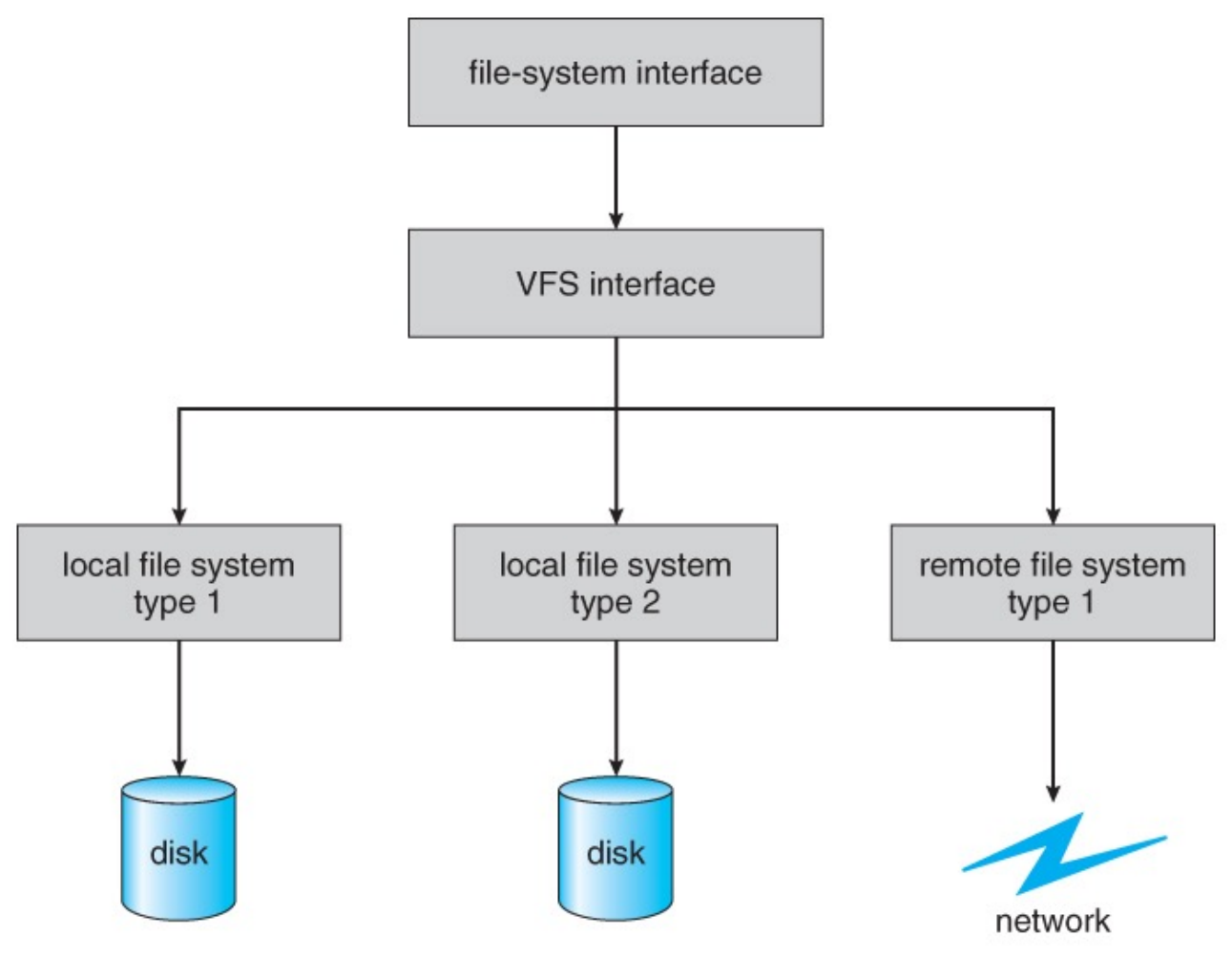
Agenda

- DFS
- Data Splitting
- Map Reduce with Cluster Configuration Demo

Source: Books, Source Code, Blogs

FS

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html

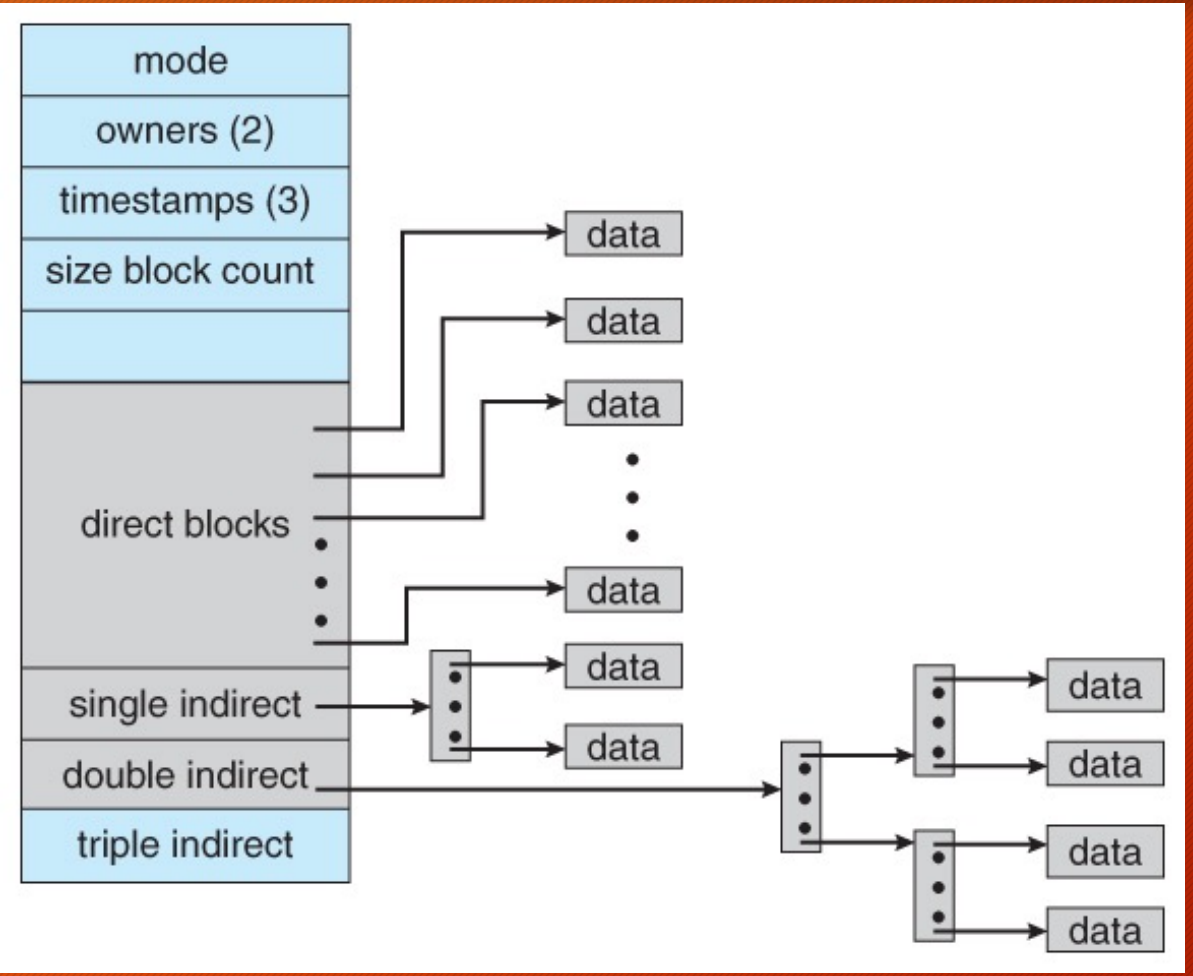


The VFS in Linux is based upon four key object types:

- The ***inode*** object, representing an individual file
- The ***file*** object, representing an open file.
- The ***superblock*** object, representing a filesystem.
- The ***dentry*** object, representing a directory entry.

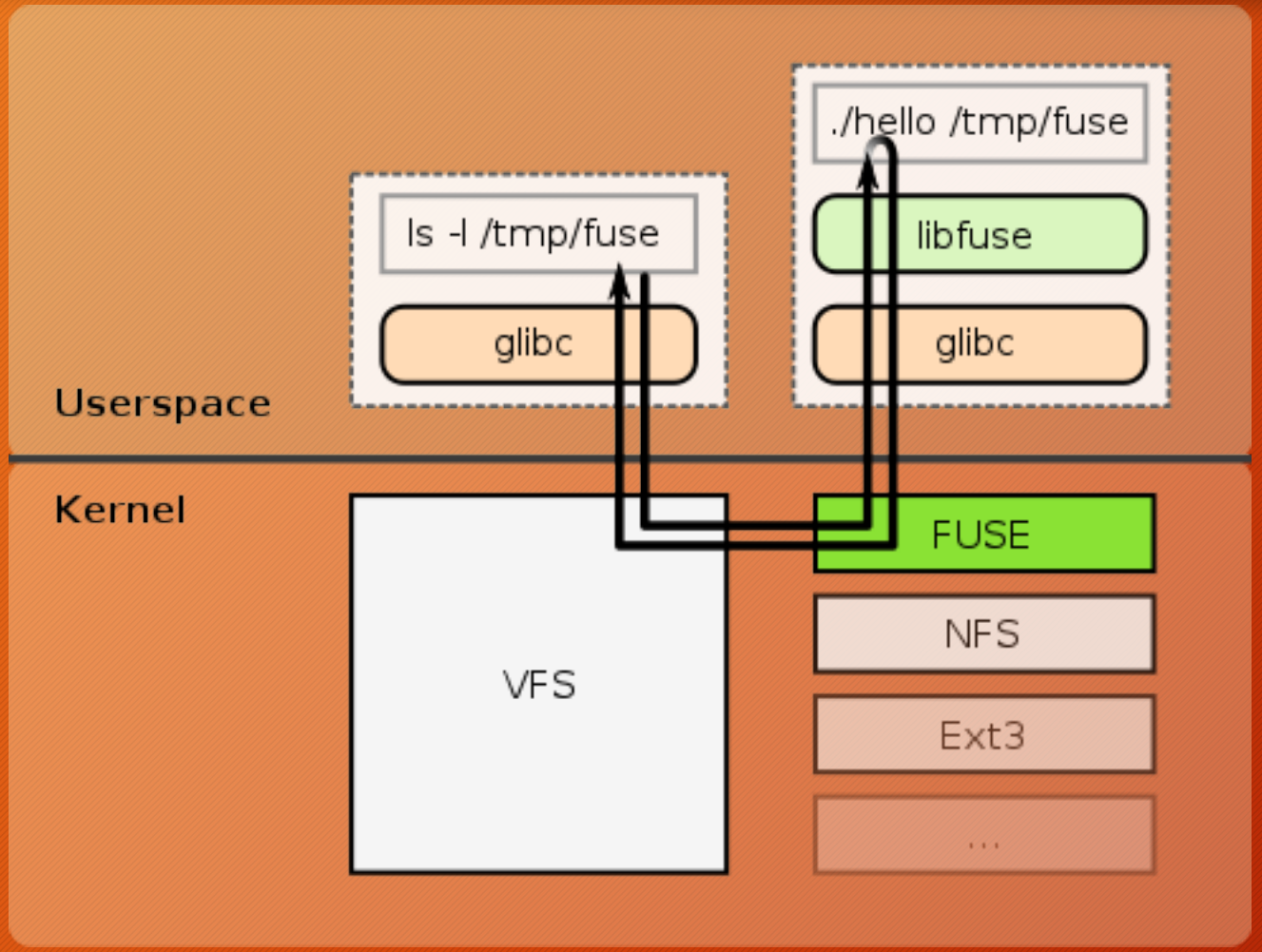
FS

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html



FUSE

https://tuxintersect.files.wordpress.com/2012/11/490px-fuse_structure-svg.png



Filesystem in User space (FUSE) allows filesystems that are implemented in user space to be integrated as Unix filesystems. Hadoop's Fuse-DFS contrib module allows HDFS (or any Hadoop filesystem) to be mounted as a standard local filesystem. Fuse-DFS is implemented in C using libhdfs as the interface to HDFS. -

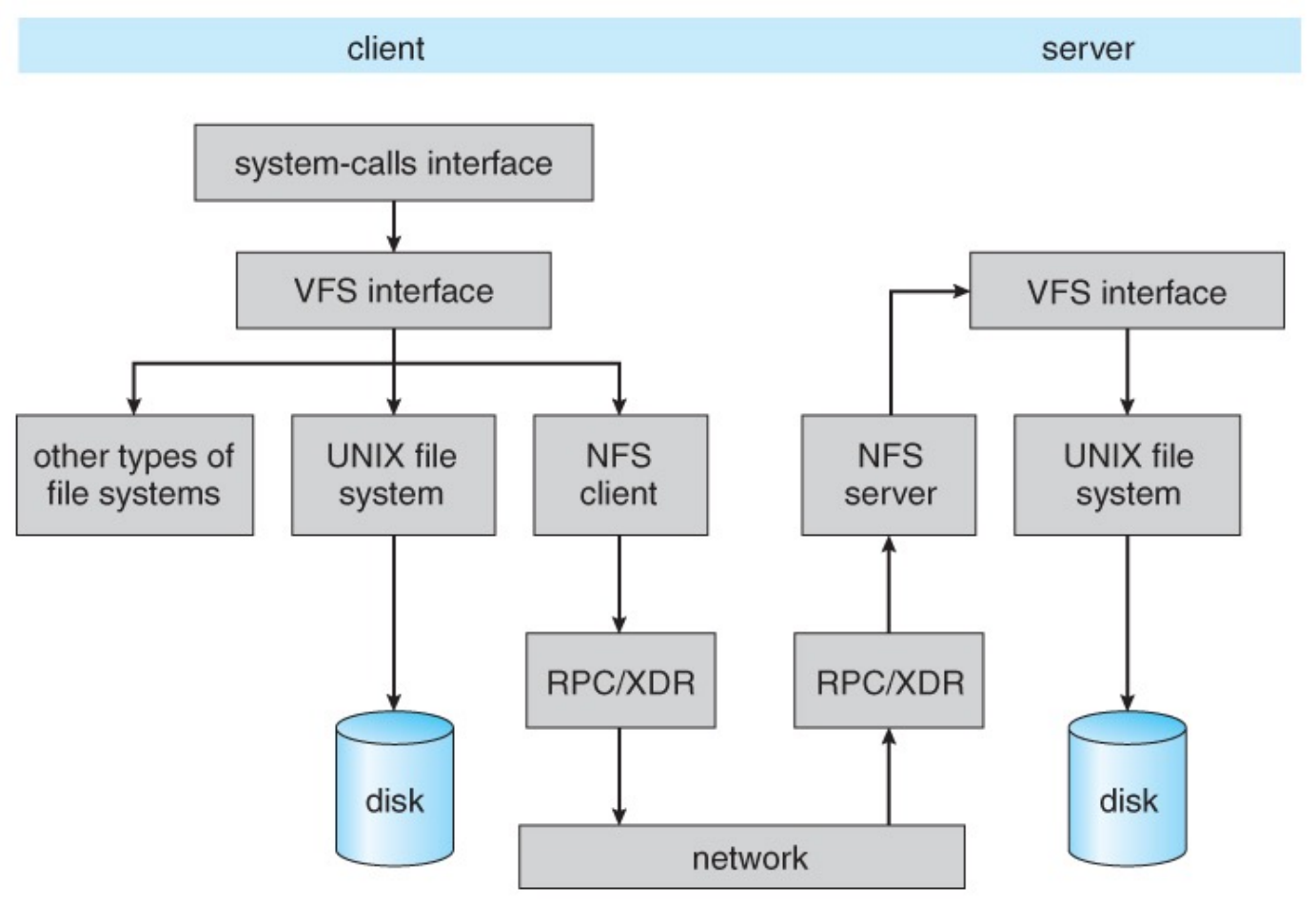
P56 of Hadoop the Definitive Guide 4th edition

DFS Considerations

- Considerations
  - Caching
  - Lock (Read-Write)
  - Fault Tolerance
  - Performance
- Examples
  - HPC : Lustre (uses RDMA)
  - VM: Gluster (uses RDMA)
  - File: NFS, CIFS
  - Big Data: **Hadoop Distributed File System (HDFS).**

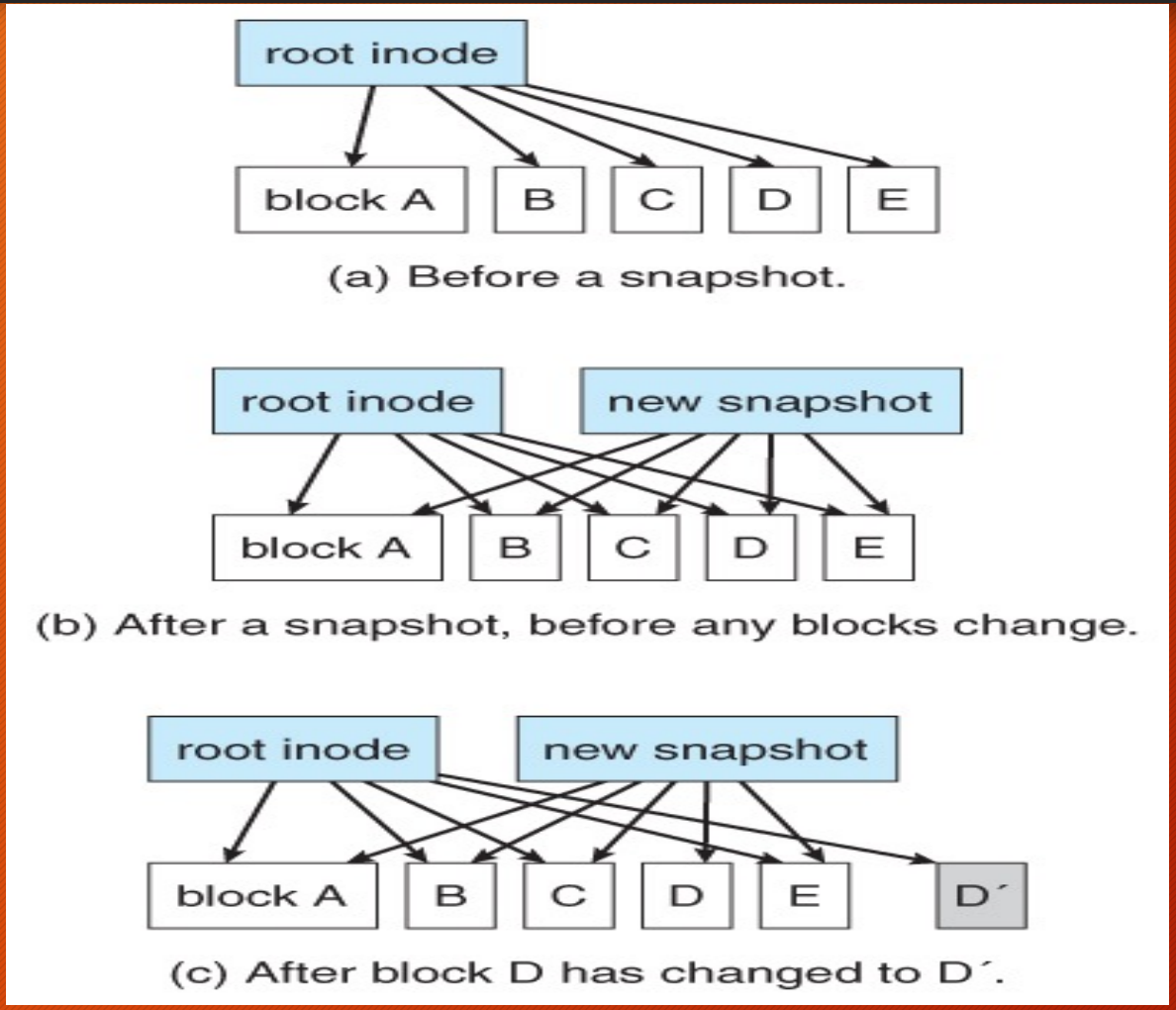
DFS Arch Sample

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html



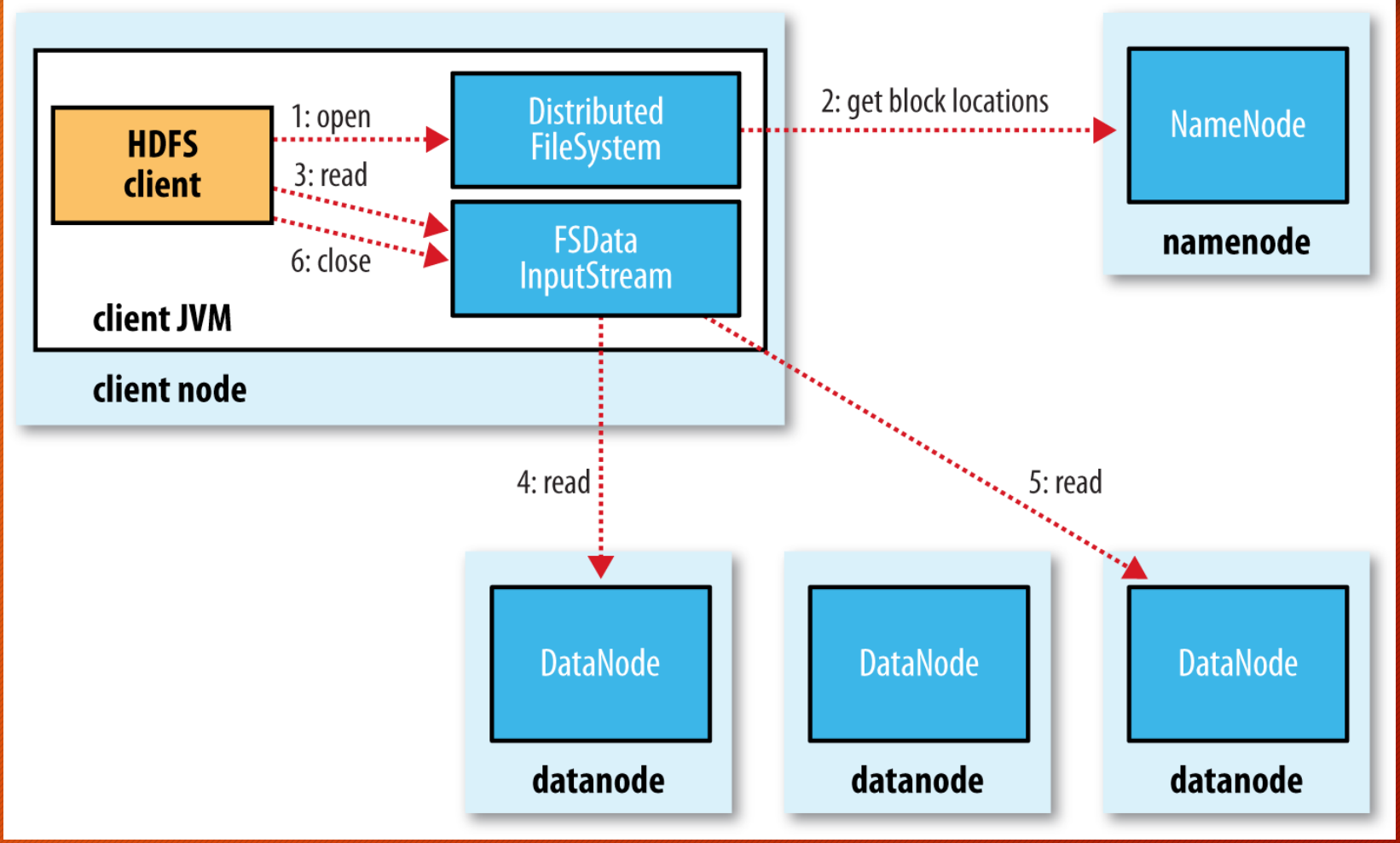
DFS Snapshot

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html



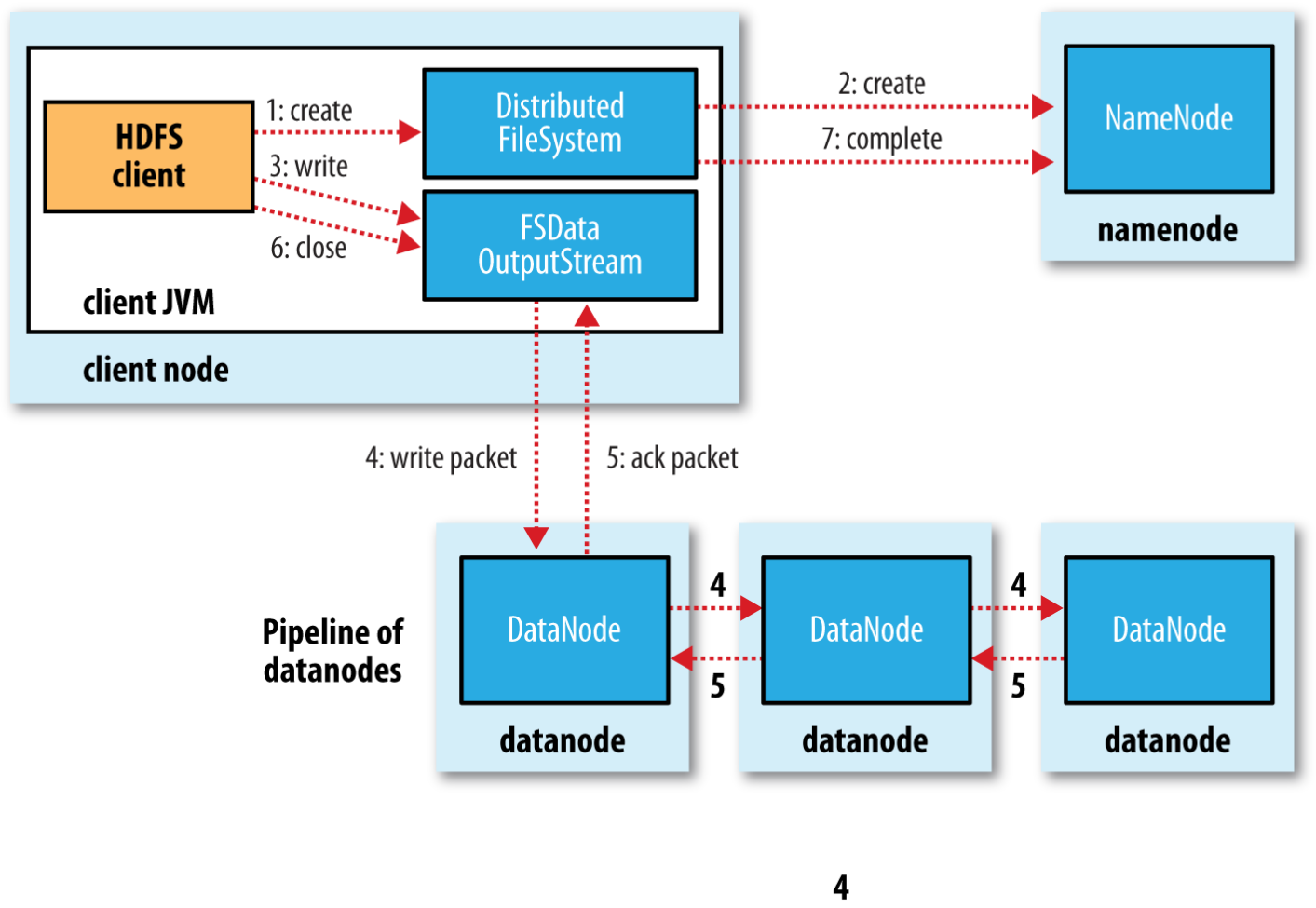
HDFS Read

Hadoop: The Definitive Guide, 4th Edition, P 69



HDFS Write

Hadoop: The Definitive Guide, 4th Edition, P 73



HDFS: INode

1. **File/directory name**: The name of the file or directory.
2. **File ID**: A unique identifier for the file or directory.
3. **Permissions**: Access control information, specifying who can read, write, or execute the file.
4. **Timestamps**: Information about when the file was created, last accessed, and last modified.
5. **Block locations**: For files, the list of blocks that make up the file and their corresponding DataNode locations.

HDFS: IDirectory

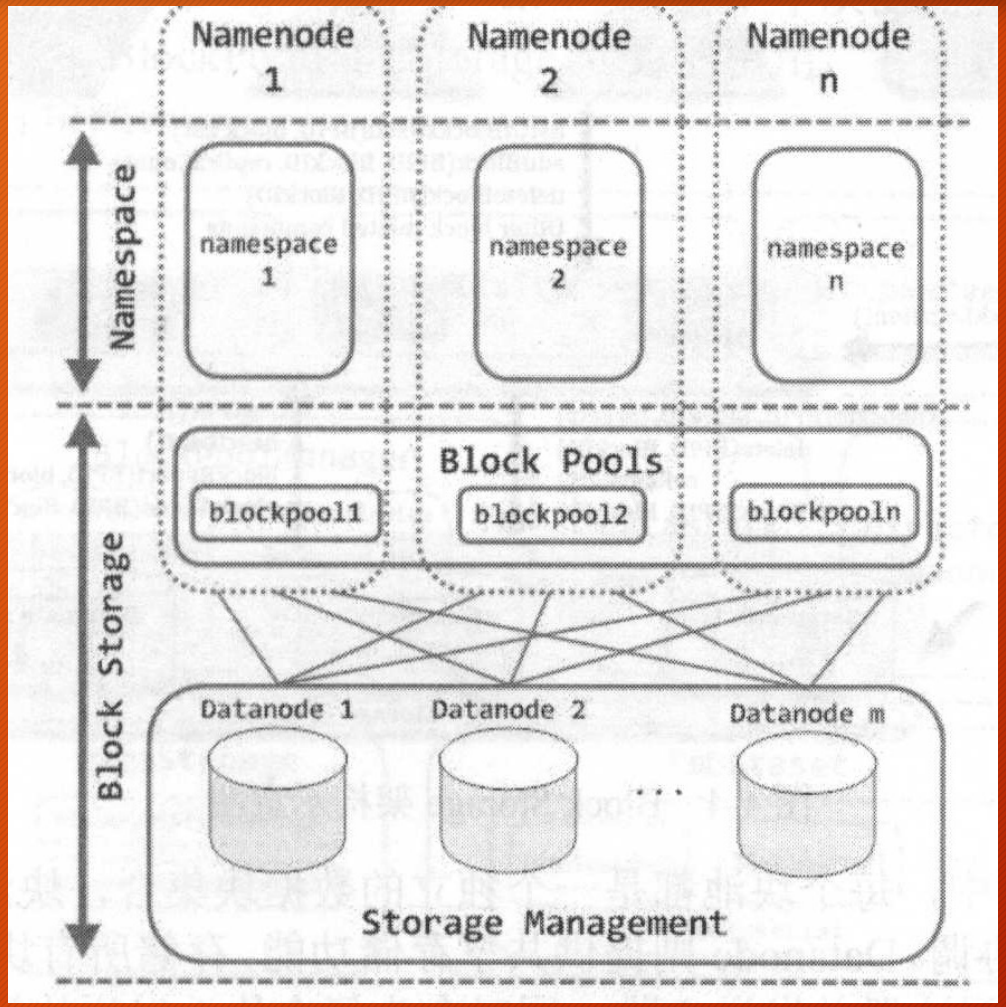
1. **Name**: The name of the directory.
2. **Parent**: A reference to the parent directory's INodeDirectory.
3. **Children**: A collection of references to the child entries (files or subdirectories) within the directory.
4. **Permissions**: Access control information specifying who can read, write, or execute the directory.
5. **Modification Timestamp**: The timestamp indicating when the directory was last modified.
6. **Access Timestamp**: The timestamp indicating when the directory was last accessed.
7. **Owner and Group**: Information about the owner and group associated with the directory.
8. **Quota Information**: If applicable, information about directory quotas.
9. **ID**: A unique identifier for the directory.

Features https://storageconference.us/2010/Papers/MSST/Shvachko.pdf

- Lease while writing (FileUnderConstructionFeature)
- Balancer
- Block Scanner
- Checkpoint and Backup
- Block Placement Strategy
- Heartbeat

Blockpools in Federation

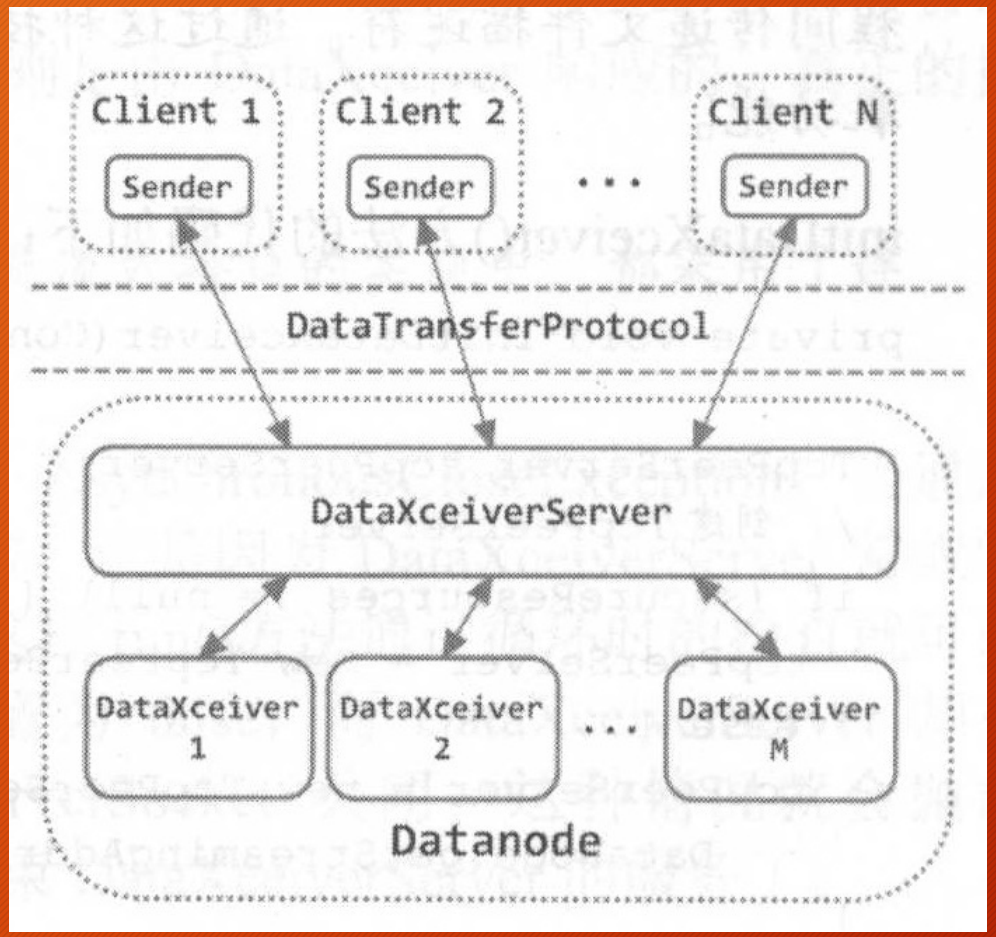
Hadoop 2.X HDFS源码剖析 (徐鹏)



Data Transfer to and from DataNode

Hadoop 2.X HDFS源码剖析 (徐鹏)

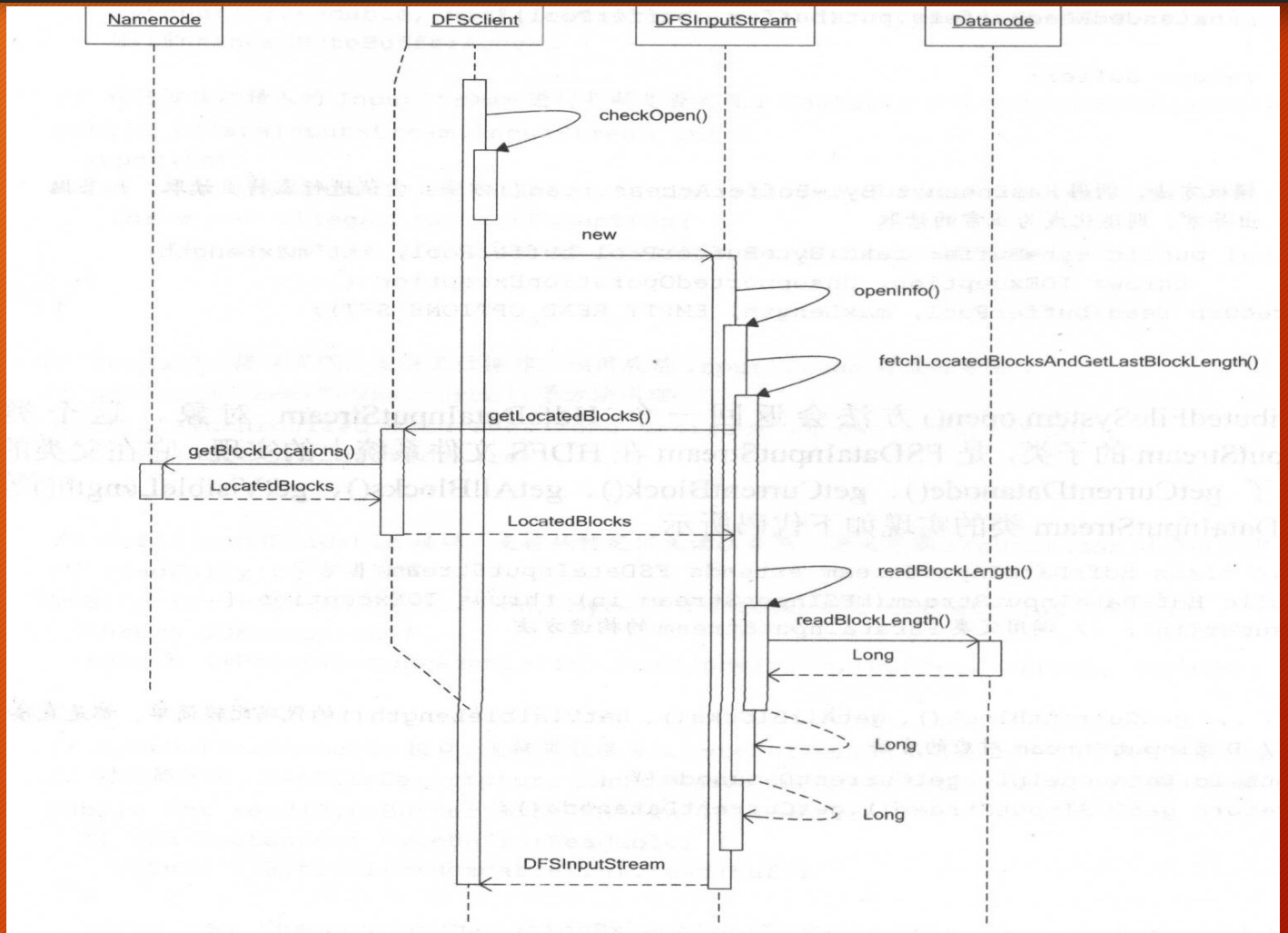
hadoop-hdfs-project/hadoop-hdfs/src/main/java/org/apache/hadoop/hdfs/server/datanode/DataXceiverServer.java




```
public void run() {
    Peer peer = null;
    while (datanode.shouldRun
&&
!datanode.shutdownForUpgrade)
{
      try {
        peer =
peerServer.accept();
…

DataXceiver.create(peer,
datanode, this)
```

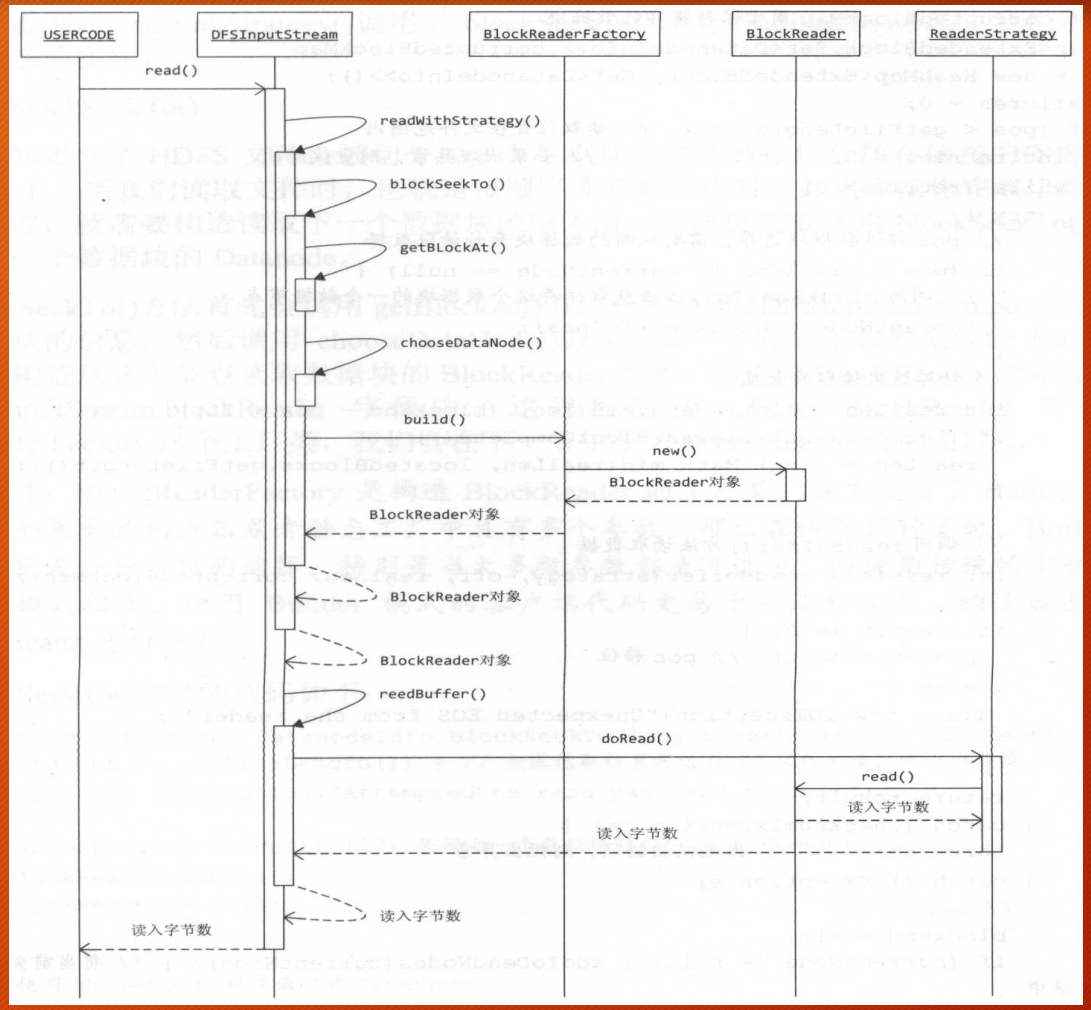
DFS Client Namenode and Datanode Interaction

Hadoop 2.X HDFS源码剖析 (徐鹏)



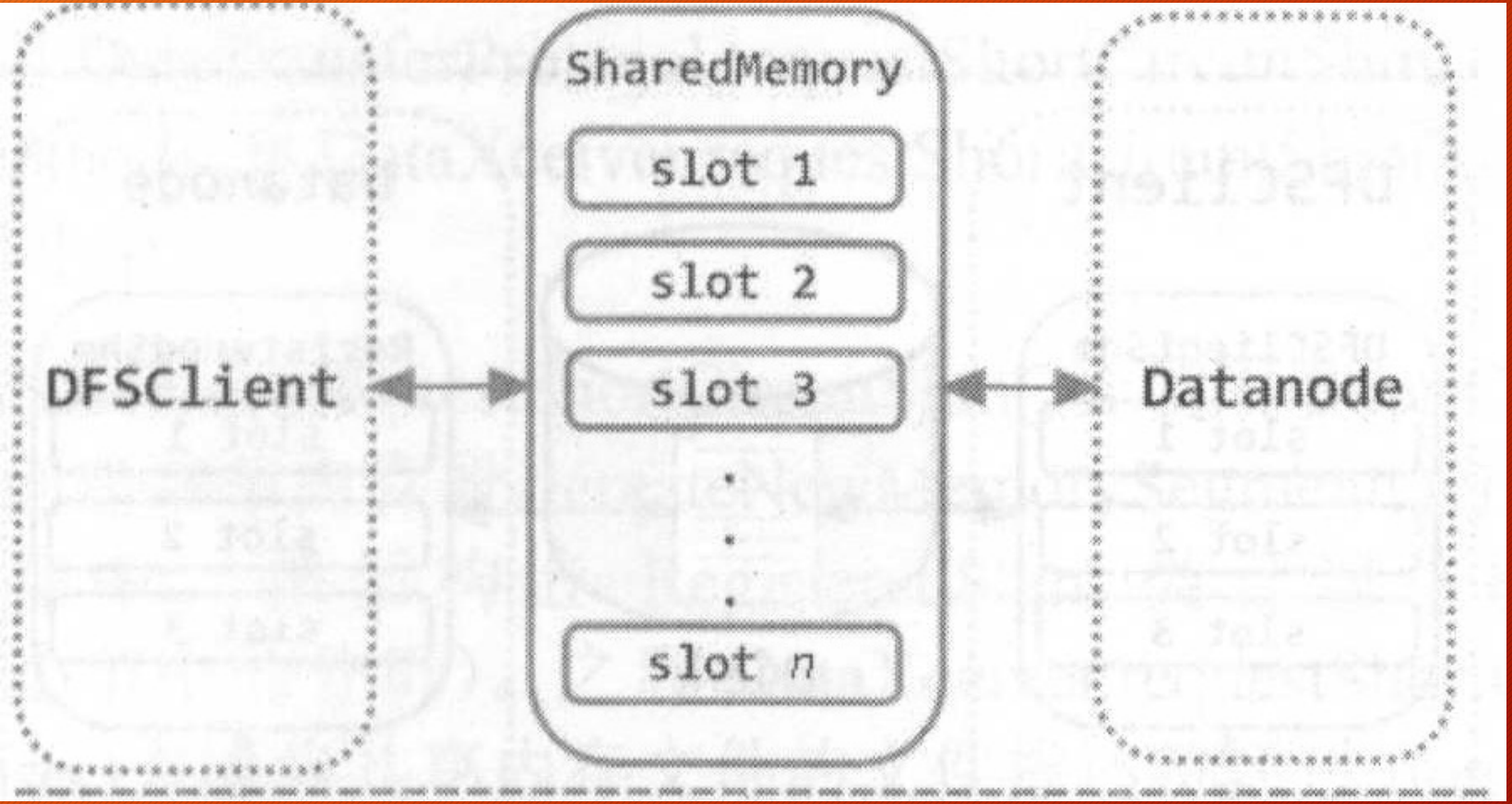
Reading process

Hadoop 2.X HDFS源码剖析 (徐鹏)



ShortCircuit

Hadoop 2.X HDFS源码剖析 (徐鹏)



Source Code References (HDFS-Local FS relationship, Commands, Block Report)

Methods	Purpose
DataXCeiverServer, DataXCeiver	Command Receiving entry point
BlockReceiver	Saves into Disk
BlockSender	Retrieves from the disk
DataBlockScanner	Periodic proactive scanning of blockpool and reporting to namenode, checksum in each block
BPOfferService (DN), BPServiceActor (NN)	Block Report

Commands: `hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/shell`

Miscellaneous: HTTP, Native, RDMA

1. **hadoop-hdfs-httpfs**:
    1. Purpose: This directory pertains to the HTTPFS component of Hadoop HDFS. HTTPFS is a server that provides HTTP access to HDFS. It allows for operations on HDFS over HTTP protocol. This can include operations like reading and writing files in HDFS, listing HDFS directories, and getting HDFS file status.
    2. Use Cases: HTTPFS is particularly useful for providing access to HDFS from outside the Hadoop cluster or for applications and services that interact with HDFS using HTTP REST API. It's an alternative to WebHDFS and is often used for compatibility with certain firewalls or for other security reasons.
2. **hadoop-hdfs-native-client**:
    1. Purpose: The hadoop-hdfs-native-client directory is associated with the native client implementation for HDFS. Native HDFS clients are typically written in C++ and offer HDFS access through a native library, as opposed to the Java-based Hadoop API.
    2. Use Cases: The native client is useful when there is a need to access HDFS from applications written in C++ or other languages where using a Java client is not ideal. It can provide performance benefits in certain scenarios and is more convenient for applications already built in a native language.

RDMA (Third Party): http://hibd.cse.ohio-state.edu/static/media/rdma-hadoop/rdma-hadoop-2.x-1.3.5-userguide.pdf

HDFS Tuning I

1. **dfs.replication**:
    1. Default: 3
    2. Description: The default block replication for files stored in HDFS. This determines how many copies of each data block are stored across the cluster.
2. **dfs.blocksize**:
    1. Default: 128MB
    2. Description: The block size for new files created in HDFS. A larger block size may improve the efficiency of large-scale data processing.
3. **dfs.namenode.name.dir**:
    1. Description: Comma-separated list of directories where the NameNode will store its metadata. This property is crucial for NameNode durability.
4. **dfs.datanode.data.dir**:
    1. Description: Comma-separated list of directories where DataNodes will store the data blocks. These directories are usually on different drives to ensure redundancy.

HDFS Tuning II

1. **dfs.datanode.handler.count**:
    1. Default: 10
    2. Description: The number of server threads for the DataNode to handle block read/write requests.
2. **dfs.namenode.checkpoint.period**:
    1. Default: 3600 (1 hour)
    2. Description: The number of seconds between two periodic checkpoints in the Secondary NameNode.
3. **dfs.namenode.checkpoint.txns**:
    1. Default: 1 million
    2. Description: The number of transactions after which the NameNode will trigger an automatic checkpoint by the Secondary NameNode.
4. **dfs.namenode.http-address** and **dfs.namenode.https-address**:
    1. Description: Configures the HTTP and HTTPS address (respectively) of the NameNode, including the port numbers.

Data Transfer Protocol implementation

hadoop-hdfs/src/main/java/org/apache/hadoop/hdfs/protocol/datatransfer/Receiver.java

```java
/** Process op by the corresponding method. */
protected final void processOp(Op op) throws IOException {
  switch(op) {
  case READ_BLOCK:
    opReadBlock();
    break;
  case WRITE_BLOCK:
    opWriteBlock(in);
    break;
  case REPLACE_BLOCK:
    opReplaceBlock(in);
    break;
  case COPY_BLOCK:
    opCopyBlock(in);
    break;
  case BLOCK_CHECKSUM:
    opBlockChecksum(in);
    break;
  case BLOCK_GROUP_CHECKSUM:
    opStripedBlockChecksum(in);
    break;
  case TRANSFER_BLOCK:
    opTransferBlock(in);
    break;
  case REQUEST_SHORT_CIRCUIT_FDS:
    opRequestShortCircuitFds(in);
    break;
  case RELEASE_SHORT_CIRCUIT_FDS:
    opReleaseShortCircuitFds(in);
    break;
  case REQUEST_SHORT_CIRCUIT_SHM:
    opRequestShortCircuitShm(in);
    break;
  default:
    throw new IOException("Unknown op " + op + " in data stream");
G.debug("{}: {}
```

Interaction with protobuf

```java
/** Receive OP_WRITE_BLOCK */
private void opWriteBlock(DataInputStream in) throws IOException {
  final OpWriteBlockProto proto = OpWriteBlockProto.parseFrom(vintPrefixed(in));
  final DatanodeInfo[] targets = PBHelperClient.convert(proto.getTargetsList());
  TraceScope traceScope = continueTraceSpan(proto.getHeader(),
      proto.getClass().getSimpleName());
  try {
    writeBlock(PBHelperClient.convert(proto.getHeader().getBaseHeader().getBlock()),
        PBHelperClient.convertStorageType(proto.getStorageType()),
        PBHelperClient.convert(proto.getHeader().getBaseHeader().getToken()),
        proto.getHeader().getClientName(),
        targets,
        PBHelperClient.convertStorageTypes(proto.getTargetStorageTypesList(), targets.length),
        PBHelperClient.convert(proto.getSource()),
        fromProto(proto.getStage()),
        proto.getPipelineSize(),
        proto.getMinBytesRcvd(), proto.getMaxBytesRcvd(),
        proto.getLatestGenerationStamp(),
        fromProto(proto.getRequestedChecksum()),
        (proto.hasCachingStrategy() ?
            getCachingStrategy(proto.getCachingStrategy()) :
          CachingStrategy.newDefaultStrategy()),
        (proto.hasAllowLazyPersist() ? proto.getAllowLazyPersist() : false),
        (proto.hasPinning() ? proto.getPinning(): false),
        (PBHelperClient.convertBooleanList(proto.getTargetPinningsList())),
        proto.getStorageId(),
        proto.getTargetStorageIdsList().toArray(new String[0]));
  } finally {
    if (traceScope != null) traceScope.close();
  }
}
```

- Endianness
- Compression

Mahout: Kmeans initial reference

```java
public static void iterateMR(Configuration conf, Path inPath, Path priorPath, Path outPath, int numIterations)
  throws IOException, InterruptedException, ClassNotFoundException {
  ClusteringPolicy policy = ClusterClassifier.readPolicy(priorPath);
  Path clustersOut = null;
  int iteration = 1;
  while (iteration <= numIterations) {
    conf.set(PRIOR_PATH_KEY, priorPath.toString());

    String jobName = "Cluster Iterator running iteration " + iteration + " over priorPath: " + priorPath;
    Job job = new Job(conf, jobName);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(ClusterWritable.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(ClusterWritable.class);

    job.setInputFormatClass(SequenceFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    job.setMapperClass(CIMapper.class);
    job.setReducerClass(CIReducer.class);

    FileInputFormat.addInputPath(job, inPath);
    clustersOut = new Path(outPath, Cluster.CLUSTERS_DIR + iteration);
    priorPath = clustersOut;
    FileOutputFormat.setOutputPath(job, clustersOut);

    job.setJarByClass(ClusterIterator.class);
    if (!job.waitForCompletion(true)) {
      throw new InterruptedException("Cluster Iteration " + iteration + " failed processing " + priorPath);
    }
    ClusterClassifier.writePolicy(policy, clustersOut);
    FileSystem fs = FileSystem.get(outPath.toUri(), conf);
    iteration++;
    if (isConverged(clustersOut, conf, fs)) {
      break;
    }
  }
  Path finalClustersIn = new Path(outPath, Cluster.CLUSTERS_DIR + (iteration - 1) + Cluster.FINAL_ITERATION_SUFFIX);
  FileSystem.get(clustersOut.toUri(), conf).rename(clustersOut, finalClustersIn);
}
```

community/mahout-mr/mr/src/main/java/org/apache/mahout/clustering/iterator/ClusterIterator.java

community/mahout-mr/mr/src/main/java/org/apache/mahout/clustering/iterator/CIMapper.java

community/mahout-mr/mr/src/main/java/org/apache/mahout/clustering/iterator/CIReducer.java

Mahout: Kmeans Sample Mapper and Reducer

```java
public class CIMapper extends Mapper<WritableComparable<?>,VectorWritable,IntWritable,ClusterWritable> {

  private ClusterClassifier classifier;
  private ClusteringPolicy policy;

  @Override
  protected void setup(Context context) throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    String priorClustersPath = conf.get(ClusterIterator.PRIOR_PATH_KEY);
    classifier = new ClusterClassifier();
    classifier.readFromSeqFiles(conf, new Path(priorClustersPath));
    policy = classifier.getPolicy();
    policy.update(classifier);
    super.setup(context);
  }

  @Override
  protected void map(WritableComparable<?> key, VectorWritable value, Context context) throws IOException,
      InterruptedException {
    Vector probabilities = classifier.classify(value.get());
    Vector selections = policy.select(probabilities);
    for (Element el : selections.nonZeroes()) {
      classifier.train(el.index(), value.get(), el.get());
    }
  }

  @Override
  protected void cleanup(Context context) throws IOException, InterruptedException {
    List<Cluster> clusters = classifier.getModels();
    ClusterWritable cw = new ClusterWritable();
    for (int index = 0; index < clusters.size(); index++) {
      cw.setValue(clusters.get(index));
      context.write(new IntWritable(index), cw);
    }
    super.cleanup(context);
  }

}
```

```java
public class CIReducer extends Reducer<IntWritable,ClusterWritable,IntWritable,ClusterWritable> {

  private ClusterClassifier classifier;
  private ClusteringPolicy policy;

  @Override
  protected void reduce(IntWritable key, Iterable<ClusterWritable> values, Context context) throws IOException,
      InterruptedException {
    Iterator<ClusterWritable> iter = values.iterator();
    Cluster first = iter.next().getValue(); // there must always be at least one
    while (iter.hasNext()) {
      Cluster cluster = iter.next().getValue();
      first.observe(cluster);
    }
    List<Cluster> models = new ArrayList<>();
    models.add(first);
    classifier = new ClusterClassifier(models, policy);
    classifier.close();
    context.write(key, new ClusterWritable(first));
  }

  @Override
  protected void setup(Context context) throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();
    String priorClustersPath = conf.get(ClusterIterator.PRIOR_PATH_KEY);
    classifier = new ClusterClassifier();
    classifier.readFromSeqFiles(conf, new Path(priorClustersPath));
    policy = classifier.getPolicy();
    policy.update(classifier);
    super.setup(context);
  }

}
```

Context.write saves into hdfs

Data Splitting in Hadoop I

1.Client Code:

A user or a client program initiates the submission of a MapReduce job, typically using the hadoop jar command or programmatically using the Hadoop Job API.

2.Job Submission:

Internally, the Job object (or a related class) is used to configure and submit the job. The submit method is often involved in this process.

3.Job Initialization:

As part of the job initialization phase, the Hadoop framework sets up various configurations, prepares resources, and performs other necessary tasks.

Data Splitting in Hadoop II

1. **InputFormat and getSplits:**

   During the job initialization, when the input format is being configured, the getSplits method of the relevant InputFormat (e.g., FileInputFormat) is called. This is where the logic for splitting input data is implemented.

2. **Split Generation:**

   The getSplits method generates an array of InputSplit objects based on the characteristics of the input data (e.g., file sizes, locations).

3. **Map Task Assignment:**

   The generated input splits, along with their locations, are used by the Hadoop framework for task scheduling. The scheduler, in collaboration with the ResourceManager, assigns map tasks to nodes in the cluster, considering data locality.

Data Splitting: Source Code Reference

- File: /Users/aiotmrdev/projects/hadoop-trunk/hadoop-mapreduce-project/hadoop-mapreduce-client/hadoop-mapreduce-client-core/src/main/java/org/apache/hadoop/mapred/FileInputFormat.java
- hadoop-mapreduce-project/hadoop-mapreduce-client/hadoop-mapreduce-client-core/src/main/java/org/apache/hadoop/mapreduce/JobSubmitter.java
- `submitJobInternal->writeSplits->writeNewSplits`
- getSplits
- RecordReader, LineRecordReader.

Splitting in FileInputFormat

```java
/**
 * Generate the list of files and make them into FileSplits.
 * @param job the job context
 * @throws IOException
 */
public List<InputSplit> getSplits(JobContext job) throws IOException {
  StopWatch sw = new StopWatch().start();
  long minSize = Math.max(getFormatMinSplitSize(), getMinSplitSize(job));
  long maxSize = getMaxSplitSize(job);

  // generate splits
  List<InputSplit> splits = new ArrayList<InputSplit>();
  List<FileStatus> files = listStatus(job);

  boolean ignoreDirs = !getInputDirRecursive(job)
    && job.getConfiguration().getBoolean(INPUT_DIR_NONRECURSIVE_IGNORE_SUBDIRS, false);
  for (FileStatus file: files) {
    if (ignoreDirs && file.isDirectory()) {
      continue;
    }
    Path path = file.getPath();
    long length = file.getLen();
    if (length != 0) {
      BlockLocation[] blkLocations;
      if (file instanceof LocatedFileStatus) {
        blkLocations = ((LocatedFileStatus) file).getBlockLocations();
      } else {
        FileSystem fs = path.getFileSystem(job.getConfiguration());
        blkLocations = fs.getFileBlockLocations(file, 0, length);
      }
```

Mapper invoking reader

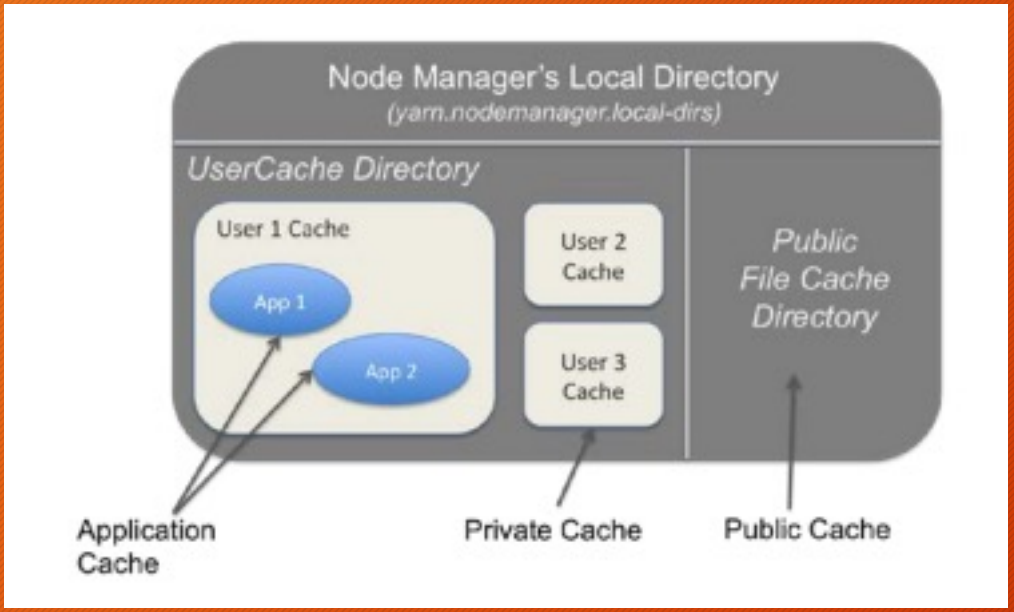
```java
/**
 * Called once at the beginning of the task.
 */
protected void setup(Context context
                     ) throws IOException, InterruptedException {
  // NOTHING
}

/**
 * Called once for each key/value pair in the input split. Most applications
 * should override this, but the default is the identity function.
 */
@SuppressWarnings("unchecked")
protected void map(KEYIN key, VALUEIN value,
                   Context context) throws IOException, InterruptedException {
  context.write((KEYOUT) key, (VALUEOUT) value);
}

/**
 * Called once at the end of the task.
 */
protected void cleanup(Context context
                       ) throws IOException, InterruptedException {
  // NOTHING
}

/**
 * Expert users can override this method for more complete control over the
 * execution of the Mapper.
 * @param context
 * @throws IOException
 */
public void run(Context context) throws IOException, InterruptedException {
  setup(context);
  while (context.nextKeyValue()) {
    map(context.getCurrentKey(), context.getCurrentValue(), context);
  }
  cleanup(context);
}
```

Reading from splits

```
public LineRecordReader(Configuration job, FileSplit split,
    byte[] recordDelimiter) throws IOException {
  this.maxLineLength = job.getInt(org.apache.hadoop.mapreduce.lib.input.
    LineRecordReader.MAX_LINE_LENGTH, Integer.MAX_VALUE);
  start = split.getStart();
  end = start + split.getLength();
  final Path file = split.getPath();
  compressionCodecs = new CompressionCodecFactory(job);
  codec = compressionCodecs.getCodec(file);
```

```
  if (start != 0) {
    start += in.readLine(new Text(), 0, maxBytesToConsume(start));
  }
  this.pos = start;
}
```

Resource Localization



Learning Yarn
ISBN 978-1-78439-396-0

Resource Localization source code reference

- Ref: https://blog.cloudera.com/management-of-application-dependencies-in-yarn/
- hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/containermanager/localizer/ResourceLocalizationService.java : run()
- hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/DefaultContainerExecutor.java:204: startLocalizer : localizer.runLocalization(nmAddr);
- => runLocalization (hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/containermanager/localizer/ContainerLocalizer.java)

THANK YOU