SecondNameNode：

SNN是一个用于检测HDFS集群状态的辅助守护进程，每个集群有一个，通常独占一个服务器（小型机群中，SNN可运行在从节点上），SNN与namenode的不同在于他不接收和记录HDFS的任何实时变化，相反，他和namenode通信，根据集群的所配置的时间间隔获取HDFS元数据的快照。当namenode失效时候，SecondNameNode可以减少停机时间并降低数据丢失的风险。

JobTracker：

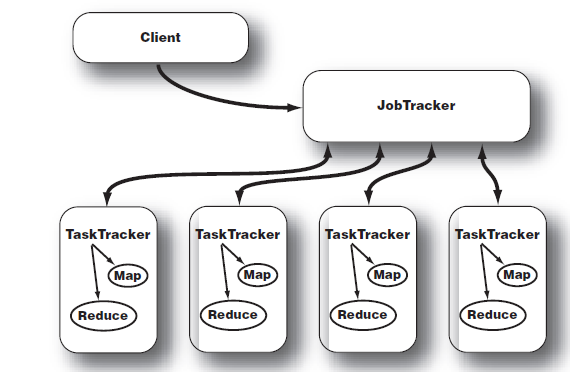
该守护进程是应用程序和Hadoop的纽带（大型集群中，JobTracker独立驻留在一个服务器中），一旦代码提交到集群，JobTracker就会确定执行计划，包括决定处理那些文件，为不同的任务分配节点以及监控所有任务的运行，如果任务失败，JobTracker将自动重启任务，但所分配的节点可能不同，同时受到预定义的重试次数限制。

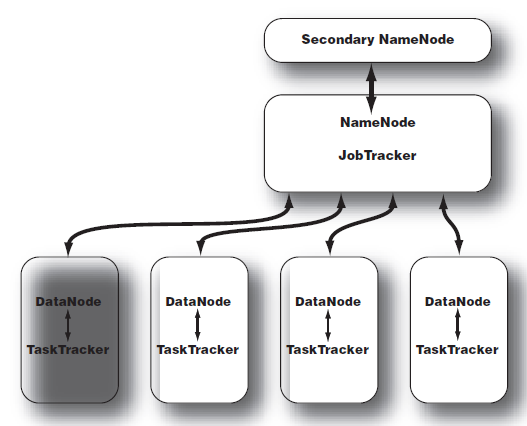
每个集群只有一个JobTracker守护进程，通常在服务器的主节点上。

TaskTracker：

计算守护进程也遵循主从架构，JobTracker作为主节点，监测MapReduce作业的整个执行过程，TaskTracker管理各个任务在每个从节点的执行情况。每个TaskTracker执行JobTracker分配的任务，每个从节点上只有一个TaskTracker，但是每个TaskTracker可以生成多个JVM（Java虚拟机）来并行执行任务。

TaskTracker持续不断地与JobTracker通信，给他发送“心跳”，如果没有心跳说明从节点以崩溃，，进而要重新提交相应的任务到集群的其他节点中。





SSH协议：

SSH协议采用公钥加密来生成一对用户验证密钥————公钥，私钥。公钥被本地存储在集群每个节点上，私钥则由主节点在试图访问远端节点时发送过来。

运行Hadoop：

第一件事就是告诉所有节点Java的位置，在hadoop-env.sh中定义JAVA\_HOME环境变量使之指向Java安装目录，，服务器上，我们指定为:

export JAVA\_HOME=/usr/local/jvm/jdk

hadoop-env.sh中的其他参数配置保持默认也能很好地工作，可日后做个性化设置（如日志目录位置，Java类所在的目录等）。

在配置Hadoop的xml文件中，core-site.xml和map-site.xml，分别指定了NameNode和JobTracker的主机名，在hdfs-site.xml指定了hdfs的默认副本数，masters中指定SNN位置（现无），slaves中指定从节点位置。

HDFS文件操作：

**基本文件命令：**

Hadoop fs –cmd <args>

**文件列表：**

hadoop fs –ls

**添加文件和目录：**

hadoop fs –mkdir /user/username #创建文件

Hadoop的mkdir命令会自动创建父目录

hadoop fs –ls / #对目录进行检查（会显示复制因子）

hadoop fs –lsr / #查看全部子目录

hadoop fs –put example.txt 等价于hadoop fs –put example.txt /user/username #上传本地文件到HDFS中

**检索文件：**

hadoop fs –get example.txt #将文件从HDFS取回本地

hadoop fs –cat example.txt #输出文件

Hadoop命令支持管道命令

hadoop fs –tail example.txt #查询会后1000字节

**删除文件：**

hadoop fs –rm example.txt #删除文件、空目录

**查看帮助：**

hadoop fs –help ls

PutMerge()函数：

开发一个PutMerge程序，用于合并文件后放入HDFS，相反，Hadoop中有个getmerge命令，用于把一组HDFS文件复制到本地计算机以前进行合并。

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class PutMerge {

public static void main(String[] args) throws IOException {

Configuration conf = new Configuration();

FileSystem hdfs = FileSystem.get(conf);

FileSystem local = FileSystem.getLocal(conf);

Path inputDir = new Path(args[0]);

Path hdfsFile = new Path(args[1]);

try {

FileStatus[] inputFiles = local.listStatus(inputDir);

FSDataOutputStream out = hdfs.create(hdfsFile);

for (int i=0; i<inputFiles.length; i++) {

System.out.println(inputFiles[i].getPath().getName());

FSDataInputStream in =

➥ local.open(inputFiles[i].getPath()); r

byte buffer[] = new byte[256];

int bytesRead = 0;

while( (bytesRead = in.read(buffer)) > 0) {

out.write(buffer, 0, bytesRead);

}

in.close();

}

out.close();

} catch (IOException e) {

e.printStackTrace();

}

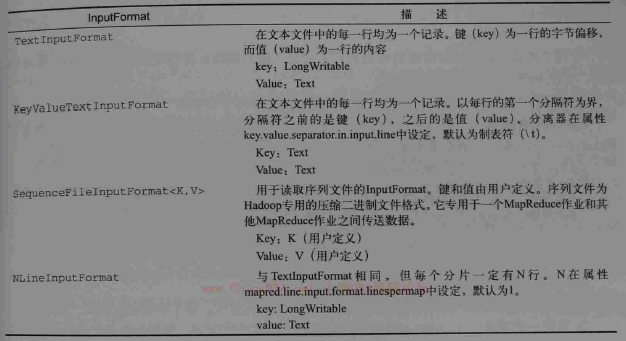
}

}

Hadoop默认地将输入文件中每一行视为一个记录而键/值对分别为该行的字节偏移和内容

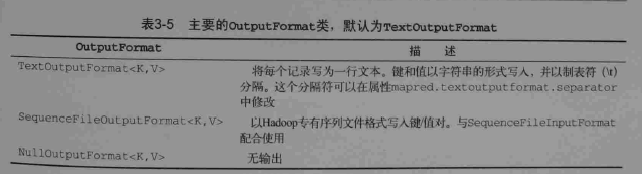
InputFormat：

Hadoop分割和读取输入文件的方式被定义在InputFormat接口的一个实现中。TextInputFormat是InputFormat的默认实现。



OutputFormat：

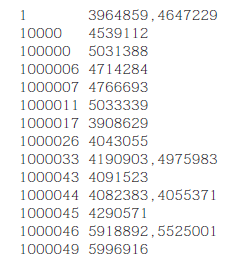
当MapReduce输出数据到文件时，使用的是OutputFormat类，它与InputFormat类相似。因为每个reducer仅需将它的输出写入自己的文件中，输出无需分片。输出文件放在一个公用目录中，通常命名为part-nnnnn，这里的nnnnn是reducer的分区ID。



编写Hadoop基础程序：

分析数据文件cite75\_99.txt、apat63\_99.txt。

**第一个程序将读取的专利引用数据并对它进行倒排，对于每一个专利，我们希望找到那些引用它的专利进行合并。**



图中4539112引用了10000.

程序如下

public class MyJob extends Configured implements Tool {

public static class MapClass extends MapReduceBase

implements Mapper<Text, Text, Text, Text> {

public void map(Text key, Text value,

OutputCollector<Text, Text> output,

Reporter reporter) throws IOException {

output.collect(value, key);

}

}

public static class Reduce extends MapReduceBase

implements Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterator<Text> values,

OutputCollector<Text, Text> output,

Reporter reporter) throws IOException {

String csv = "";

while (values.hasNext()) {

if (csv.length() > 0) csv += ",";

csv += values.next().toString();

}

output.collect(key, new Text(csv));

}

}

public int run(String[] args) throws Exception {

Configuration conf = getConf();

JobConf job = new JobConf(conf, MyJob.class);

Path in = new Path(args[0]);

Path out = new Path(args[1]);

FileInputFormat.setInputPaths(job, in);

FileOutputFormat.setOutputPath(job, out);

job.setJobName("MyJob");

job.setMapperClass(MapClass.class);

job.setReducerClass(Reduce.class);

job.setInputFormat(KeyValueTextInputFormat.class);

job.setOutputFormat(TextOutputFormat.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.set("key.value.separator.in.input.line", ",");//用于设置KeyValueTextInputFormat的分隔符

JobClient.runJob(job);

return 0;

}

public static void main(String[] args) throws Exception {

int res = ToolRunner.run(new Configuration(), new MyJob(), args);

System.exit(res);

}

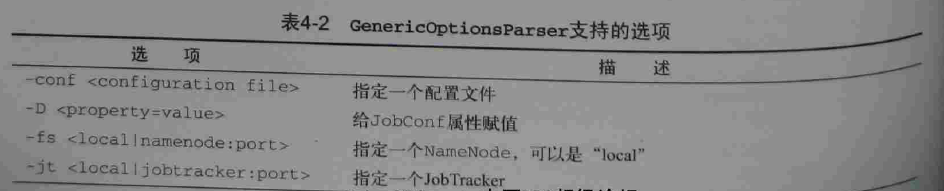
}

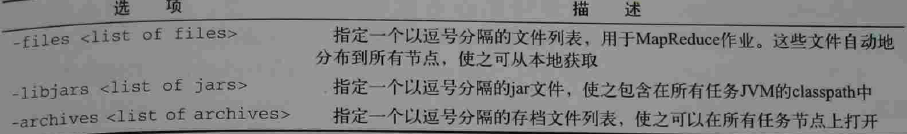
我们可以用以下方法执行MyJob类：

bin/Hadoop jar playground/MyJob.jar MyJob input/cite75\_99.txt output

如果我们只想看到mapper的输出（用于调试），可以用选项-D mapred.reduce.tasks=0将reducer的数目设置为0.

Bin/Hadoop jar playground/MyJob.jar MyJob –D mapred.reduce.tasks=0 input/cite75\_99.txt output





Hadoop的Streaming：

我们的Hadoop程序都是用Java写的，Hadoop也支持用其他语言来编程，这需要用到名为Streaming的通用API，在实际应用中，Streaming主要用于编写简单、短小的MapReduce程序，它们可以通过脚本语言编程，开发更加敏捷，并能够充分利用非Java库。

**通过Unix命令使用Streaming：**

bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar

➥ -input input/cite75\_99.txt

➥ -output output

➥ -mapper 'cut -f 2 -d ,'

➥ -reducer 'uniq'

**通过脚本使用Streaming：**

cat input.txt | **RandomSample.py** 10 >sampled\_output.txt

RandomSample.py 是是Python程序，参数是10。

bin/hadoop jar contrib/streaming/hadoop-0.19.1-streaming.jar

➥ -input input/cite75\_99.txt

➥ -output output

➥ -mapper 'RandomSample.py 10'

➥ -file RandomSample.py

➥ -D mapred.reduce.tasks=1

-D mapred.reduce.tasks=1，因为没有设定特殊的reducer，这里默认使用IdentityReudcer，顾名思义IdentityReudcer是把输入直接转向输出，这里可以设置reducer的个数为任一非零的值，从而得到一组确定数目的输出文件。