[Hadoop及相关技术简介 1](#_Toc366672612)

[Hadoop HDFS 3](#_Toc366672615)

[Hadoop MapReduce 7](#_Toc366672620)

[Hadoop程序示例 12](#_Toc366672623)

[学习建议 14](#_Toc366672624)

# Hadoop及相关技术简介

## Hadoop的历史

Hadoop这个名字不是一个缩写，它是一个虚构的名字。该项目的创建者，Doug Cutting如此解释Hadoop的得名："这个名字是我孩子给一头吃饱了的棕黄色大象命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。Googol就是由小孩命名的。"

**2003年 Google发表描述Google分布式文件系统(简称GFS)的论文**

**2004年 Google在发表了题为"MapReduce: Simplified Data Processing on Large Clusters"的论文**

2004年-- 最初的版本(现在称为HDFS和MapReduce)由Doug Cutting和Mike Cafarella为解决大规模网页抓取、存储和处理的成本问题。

2005年12月-- Nutch移植到新的框架，Hadoop在20个节点上稳定运行。

2006年1月-- Doug Cutting加入雅虎。

**2006年2月-- Apache Hadoop项目正式启动以支持MapReduce和HDFS的独立发展。**

2006年2月-- 雅虎的网格计算团队采用Hadoop。

**08年1月 – hadoop成为apache顶级项目**

08年4月-- 赢得世界最快1 TB数据排序在900个节点上用时209秒。

09年4月-- 赢得每分钟排序，59秒内排序500 GB(在1400个节点上)和173分钟内排序100 TB数据(在3400个节点上)。

现在：hadoop技术已广泛应用与个互联网公司、政府和科研机构。

## Hadoop到底是什么？

Hadoop是一个分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。**简单地说来，Hadoop是一个易用的大规模数据存储和计算平台。**

**Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），简称HDFS**

**Hadoop 实现了MapReduce 计算模型。**

下面列举hadoop主要的一些特点：

1 **扩容能力（Scalable）**：能可靠地（reliably）存储和处理千兆字节（PB）数据。

2 **成本低（Economical）**：可以通过普通机器组成的服务器群来分发以及处理数据。这些服务器群总计可达数千个节点。

3 **高效率（Efficient）**：通过分发数据，hadoop可以在数据所在的节点上并行地（parallel）处理它们，这使得处理非常的快速。

4 **可靠性（Reliable）**：hadoop能自动地维护数据的多份复制，并且在任务失败后能自动地重新部署（redeploy）计算任务。

围绕hadoop衍生了许多子项目，比如：

* [**Apache Pig**](http://hortonworks.com/hadoop/pig)Apache pig是用来处理大规模数据的高级查询语言。

* [**Apache Hive**](http://hortonworks.com/hadoop/hive)基于Hadoop的数据仓库，将数据存储在HDFS上，并支持类SQL语句的查询.

* [**Apache HBase**](http://hortonworks.com/hadoop/hbase)基于Hadoop的非关系型数据库，HBase是Google Bigtable的开源实现.

* [**Apache ZooKeeper**](http://hortonworks.com/hadoop/zookeeper)  
  是一个针对大型分布式系统的可靠协调系统，提供的功能包括：配置维护、名字服务、分布式同步、组服务等。

* [**Apache Sqoop**](http://hortonworks.com/hadoop/sqoop)  
  Sqoop是一个用于Hadoop和关系型数据库间的数据交换工具，可以将一个关系型数据库（例如 ： MySQL ,Oracle ,Postgres等）中的数据导进到Hadoop的HDFS中，也可以将HDFS的数据导进到关系型数据库中.

* [**Apache Oozie**](http://hortonworks.com/hadoop/oozie)  
  Hadoop作业调度工具.

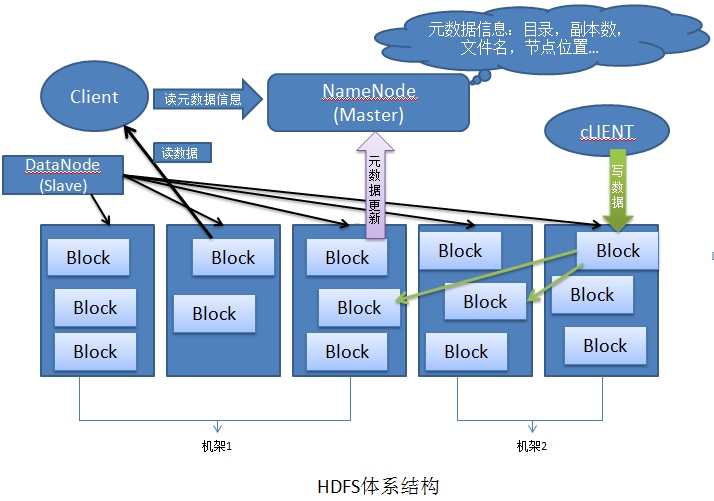
* **Apache Mahout**  
  基于Hadoop的机器学习算法包.

# Hadoop HDFS

Hadoop Distributed File System，简称**HDFS，是一个分布式文件系统。**HDFS有着高容错性（fault-tolerant）的特点，并且设计用来部署在低廉的（low-cost）硬件上。而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。

HDFS是hadoop项目的一部分，可以说是Hadoop和Hbase的基石.

**HDFS是一个主从(Master/Slave)的架构，一个HDFS集群是由一个名字节点(NameNode)，若干数据节点(DataNode)组成。**



## 数据块 (block)

**HDFS默认的基本存储单位是64M的数据块**，和普通文件系统相同的是，HDFS中的文件是被分成64M一块的数据块存储的。 **不同于普通文件系统的是，HDFS中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间**。

## 元数据节点(NameNode)

**主要用来管理文件系统的命名空间，其将所有的文件和文件夹的元数据保存在一个文件系统树中**。 这些信息也会在硬盘上保存成以下文件：命名空间镜像(namespace image)及修改日志(edit log),**其还保存了一个文件包括哪些数据块，分布在哪些数据节点上**。然而这些信息并不存储在硬盘上，而是在系统启动的时候从数据节点收集而成的。

## 数据节点(DataNode)

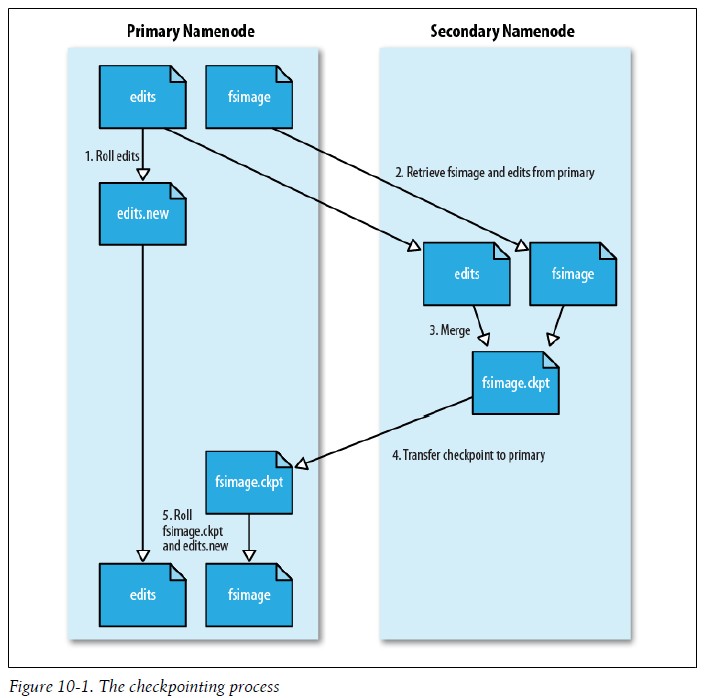
**真正存储数据的地方。**客户端(client)或者元数据信息(namenode)可以向数据节点请求写入或者读出数据块。 **周期性的向元数据节点回报其存储的数据块信息**。

## 从元数据节点(Secondary NameNode)

从元数据节点并不是元数据节点出现问题的时候的备用节点，它和元数据节点负责不同的事情。

其主要功能就是周期性将元数据节点命名空间的镜像文件和修改日志文件合并，以防日志文件过大。

合并过后的命名空间镜像文件也在从元数据节点保存了一份，以防元数据节点失败的时候，可以恢复。secondarynamenode一般来说不应该和namenode在一起。

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/2012081314565068.jpg)

Secondary NameNode主要是做Namespace image和Edit log合并的。

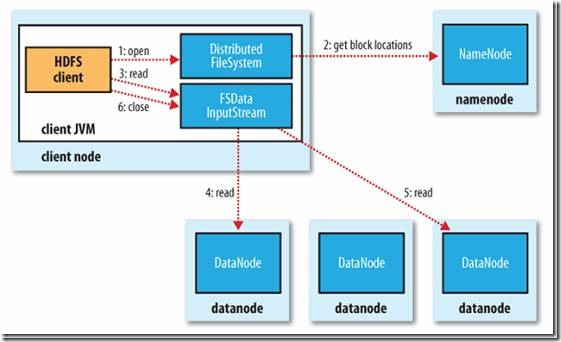
**当客户端执行写操作，则NameNode会在edit log记录下来**，并在内存中保存一份文件系统的元数据。

**fsimage是存于硬盘的元数据检查点，**Hadoop不会对每个文件操作都写出到fsimage，这样是很慢的，但是每个文件操作都会在提交后运行前先写入edits编辑日志**，这样在namenode出现故障后，就会将fsimage和edits编辑日志结合读入内存，重建元数据信息。**

由于Edit log不断增长，在NameNode重启时，会造成长时间NameNode处于安全模式，即不可用状态，是非常不符合Hadoop的设计初衷。**所以要周期性合并Edit log，但是这个工作如果由NameNode来完成，会占用大量资源，这样就出现了Secondary NameNode，它可以进行image检查点的处理工作**。步骤如下：

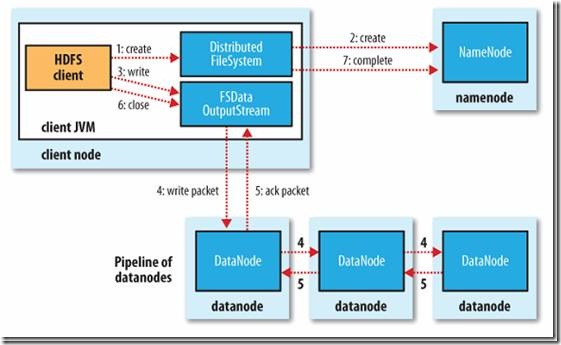
* Secondary NameNode请求NameNode进行edit log的滚动（即创建一个新的edit log），将新的编辑操作记录到新生成的edit log文件；
* 通过http get方式，读取NameNode上的fsimage和edits文件，到Secondary NameNode上；
* 读取fsimage到内存中，即加载fsimage到内存，然后执行edits中所有操作并生成一个新的fsimage文件，即这个检查点被创建；
* 通过http post方式，将新的fsimage文件传送到NameNode；
* NameNode使用新的fsimage替换原来的fsimage文件，让 Secondary NameNode创建的edits替代原来的edits文件；并且更新fsimage文件的检查点时间。整个处理过程完成。

Secondary NameNode的处理，是将fsimage和edites文件周期的合并，不会造成nameNode重启时造成长时间不可访问的情况。

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/100926154807.png)

文件读取流程

* 客户端(client)用FileSystem的open()函数打开文件
* DistributedFileSystem**用RPC调用元数据节点**，得到文件的数据块信息。
* 对于每一个数据块，元数据节点返回保存数据块的数据节点的地址。
* DistributedFileSystem**返回FSDataInputStream给客户端**，用来读取数据。
* 客户端调用stream的read()函数开始读取数据。
* DFSInputStream连接保存此文件第一个数据块的**最近的数据节点**。
* Data从数据节点读到客户端(client)
* 当此数据块读取完毕时，DFSInputStream关闭和此数据节点的连接，然后连接此文件下一个数据块的最近的数据节点。
* 当客户端读取完毕数据的时候，调用FSDataInputStream的close函数。
* 在读取数据的过程中，如果客户端在与数据节点通信出现错误，则**尝试连接包含此数据块的下一个数据节点**。
* **失败的数据节点将被记录，以后不再连接**。

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/100926154727.png)

HDFS写文件过程

 客户端调用create()来创建文件

* DistributedFileSystem用RPC调用元数据节点，在文件系统的命名空间中创建一个新的文件。
* **元数据节点首先确定文件原来不存在**，并且客户端有创建文件的权限，然后创建新文件。
* DistributedFileSystem返回DFSOutputStream，客户端用于写数据。
* 客户端开始写入数据，DFSOutputStream将数据分成块，写入data queue。
* Data queue由Data Streamer读取，**并通知元数据节点分配数据节点**，用来存储数据块(每块默认复制3块)。分配的数据节点**放在一个pipeline**里。
* Data Streamer将数据块写入pipeline中的第一个数据节点。第一个数据节点将数据块发送给第二个数据节点。第二个数据节点将数据发送给第三个数据节点。（**分散数据分发任务到其他节点上**）
* DFSOutputStream为发出去的数据块保存了ack queue，等待pipeline中的数据节点告知数据已经写入成功。
* 如果数据节点在写入的过程中失败：
* 关闭pipeline，将ack queue中的数据块放入data queue的开始。
* 当前的数据块在已经写入的数据节点中被元数据节点赋予新的标示，则错误节点重启后能够察觉其数据块是过时的，会被删除。
* 失败的数据节点从pipeline中移除，另外的数据块则写入pipeline中的另外两个数据节点。
* 元数据节点则被通知此数据块是复制块数不足，将来会再创建第三份备份。
* 当客户端结束写入数据，则调用stream的close函数。此操作将所有的数据块写入pipeline中的数据节点，并等待ack queue返回成功。**最后通知元数据节点写入完毕**。

# Hadoop MapReduce

MapReduce中最重要的两个词就是Map（映射）和Reduce（规约）。

面对复杂问题，古人教导我们要“**分而治之**”，英文中对应的词是Divide and Conquer。Map/Reduce其实就是Divide/Conquer的过程，通过把问题Divide，使这些Divide后的Map运算高度并行，再将Map后的结果Reduce（根据某一个Key），得到最终的结果。

Map：

由用户定义，将输入通过一定规则转化为一系列 中间《key，value》值对。MapReduce框架会将这些中间值对根据Key值汇集起来，交给reduce。

Reduce：

也由用户定义，接受各个Map传来的《Key,list of values》将所有相同key值的value汇集起来。进行化简。

MapReduce论文中给出了这样一个例子：在一个文档集合中统计每个单词出现的次数。

Map操作的输入是每一篇文档，将输入文档中每一个单词的出现输出到中间文件中去。

map(String key, String value):

// key: document name

// value: document contents

for each word w in value:

EmitIntermediate(w, “1″);

比如我们有两篇文档，内容分别是

A － “I love programming”

B － “I am a blogger, you are also a blogger”。

B文档经过Map运算后输出的中间文件将会是：

I,1

love,1

programming,1

B文档经过Map运算后输出的中间文件将会是：

I,1

am,1

a,1

blogger,1

you,1

are,1

a,1

blogger,1

Reduce操作的输入是单词和出现次数的序列。用上面的例子来说，就是 (“I”, [1, 1]), (“love”, [1]), (“programming”, [1]), (“am”, [1]), (“a”, [1,1]) 等。然后根据每个单词，算出总的出现次数。

reduce(String key, Iterator values):

// key: a word

// values: a list of counts

int result = 0;

for each v in values:

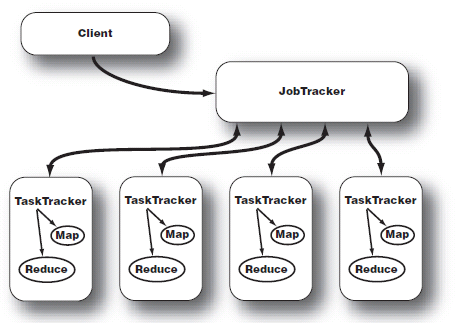
result += ParseInt(v);

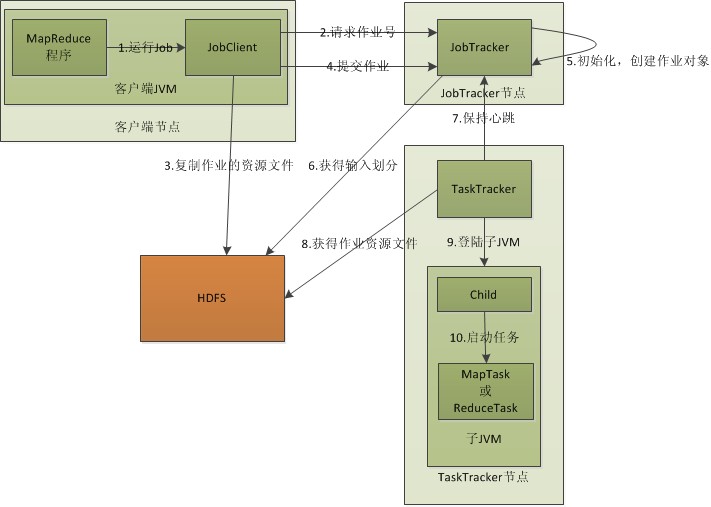
Emit(AsString(result));

最后输出的最终结果就会是：(“I”, 2″), (“a”, 2″)……

## ****流程分析****

## Jobtracker 和 Tasktracker



  
1.在客户端启动一个作业。

2.向JobTracker请求一个Job ID。

3.**将运行作业所需要的资源文件复制到HDFS上**，包括MapReduce程序打包的JAR文件、配置文件和客户端计算所得的输入划分信息。这些文件都存放在JobTracker专门为该作业创建的文件夹中。文件夹名为该作业的Job ID。

4.**JobTracker接收到作业后，将其放在一个作业队列里，等待作业调度器对其进行调度**，

当作业调度器根据自己的调度算法调度到该作业时，会根据输入划分信息为每个划分创建一个map任务，并将map任务分配给TaskTracker执行。

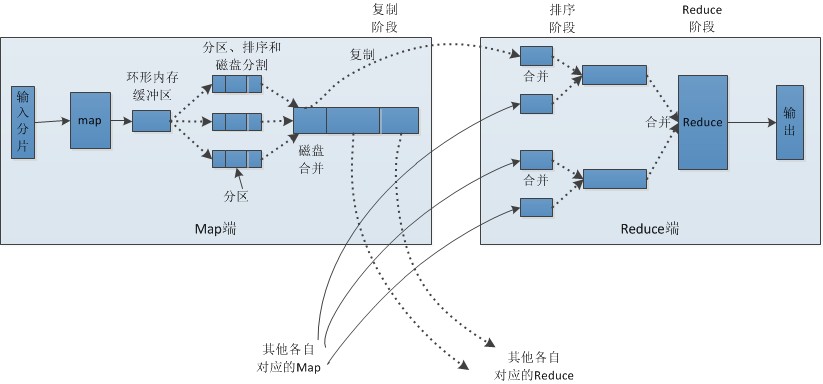
对于map和reduce任务，TaskTracker根据主机核的数量和内存的大小有固定数量的map槽和reduce槽。

这里需要强调的是：map任务不是随随便便地分配给某个TaskTracker的，这里有个概念叫：

**数据本地化（Data-Local）**。意思是：将map任务分配给含有该map处理的数据块的TaskTracker上，同时将程序JAR包复制到该TaskTracker上来运行，这叫“运算移动，数据不移动”。

而分配reduce任务时并不考虑数据本地化。

5.**TaskTracker每隔一段时间会给JobTracker发送一个心跳，告诉JobTracker它依然在运行，同时心跳中还携带着很多的信息，比如当前map任务完成的进度等信息**。当JobTracker收到作业的最后一个任务完成信息时，便把该作业设置成“成功”。当JobClient查询状态时，它将得知任务已完成，便显示一条消息给用户。  
  
以上是在客户端、JobTracker、TaskTracker的层次来分析MapReduce的工作原理的，下面我们再细致一点，从map任务和reduce任务的层次来分析分析。  
  
Map、Reduce任务中Shuffle和排序的过程



**Map端：**   
  
1．每个输入分片会让一个map任务来处理，默认情况下，以HDFS的一个块的大小（默认为64M）为一个分片，当然我们也可以设置块的大小。**map输出的结果会暂且放在一个环形内存缓冲区中**（该缓冲区的大小默认为100M，由io.sort.mb属性控制），当该缓冲区快要溢出时（默认为缓冲区大小的80%，由io.sort.spill.percent属性控制），会在本地文件系统中创建一个溢出文件，将该缓冲区中的数据写入这个文件。  
  
2．在写入磁盘之前，线程首先根据reduce任务的数目将数据划分为相同数目的分区，**也就是一个reduce任务对应一个分区的数据**。这样做是为了避免有些reduce任务分配到大量数据，而有些reduce任务却分到很少数据，甚至没有分到数据的尴尬局面。其实**分区就是对数据Key值进行hash的过程。然后对每个分区中的数据进行排序**，如果此时设置了Combiner，将排序后的结果进行Combine操作，这样做的目的是让尽可能少的数据写入到磁盘。  
  
3．当map任务输出最后一个记录时，可能会有很多的溢出文件，这时需要将这些文件合并。**合并的过程中会不断地进行排序和combine操作**，目的有两个：1.尽量减少每次写入磁盘的数据量；2.尽量减少下一复制阶段网络传输的数据量。**最后合并成了一个已分区且已排序的文件**。  
  
4．将分区中的数据拷贝给相对应的reduce任务。有人可能会问：分区中的数据怎么知道它对应的reduce是哪个呢？其实map任务一直和其父TaskTracker保持联系，而TaskTracker又一直和JobTracker保持心跳。**所以JobTracker中保存了整个集群中的宏观信息**。只要reduce任务向JobTracker获取对应的map输出位置就ok了哦。  
  
到这里，map端就分析完了。那到底什么是Shuffle呢？Shuffle的中文意思是“洗牌”，如果我们这样看：**一个map产生的数据，结果通过hash过程分区却分配给了不同的reduce任务，这就是shuffle。**   
  
**Reduce端：**   
  
1．Reduce会接收到不同map任务传来的数据，并且每个map传来的数据都是有序的。如果reduce端接受的数据量相当小，则直接存储在内存中，如果数据量超过了该缓冲区大小的一定比例，则对数据合并后溢写到磁盘中。  
  
2．随着溢写文件的增多，后台线程会将它们根据Key值合并成一个更大的有序的文件，这样做是为了给后面的合并节省时间。其实不管在map端还是reduce端，MapReduce都是反复地执行排序，合并操作。  
  
3．最后的有序大文件里面，相同key值的记录会在一起，使用iterator迭代这些记录，就得到了《Key，list of value》。之后就是reduce函数运行，最终输出到HDFS上。

# Hadoop程序示例

1. **public** **class** WordCount
2. {
4. /\*\*
5. \* MapReduceBase类:实现了Mapper和Reducer接口的基类（其中的方法只是实现接口，而未作任何事情）
6. \* Mapper接口：
7. \* WritableComparable接口：实现WritableComparable的类可以相互比较。所有被用作key的类应该实现此接口。
8. \* Reporter 则可用于报告整个应用的运行进度，本例中未使用。
9. \*
10. \*/
11. **public** **static** **class** Map **extends** MapReduceBase **implements**
12. Mapper<LongWritable, Text, Text, IntWritable>
13. {
14. /\*\*
15. \* LongWritable, IntWritable, Text 均是 Hadoop 中实现的用于封装 Java 数据类型的类，这些类实现了WritableComparable接口，
16. \* **都能够被串行化从而便于在分布式环境中进行数据交换，你可以将它们分别视为long,int,String 的替代品**。
17. \*/
18. **private** **final** **static** IntWritable one = **new** IntWritable(1);
19. **private** Text word = **new** Text();
21. /\*\*
22. \* Mapper接口中的map方法：
23. \* void map(K1 key, V1 value, OutputCollector<K2,V2> output, Reporter reporter)
24. \* 映射一个单个的输入k/v对到一个中间的k/v对
25. \* 输出对不需要和输入对是相同的类型，输入对可以映射到0个或多个输出对，可以输出多次。
26. \* OutputCollector接口：收集Mapper和Reducer输出的<k,v>对。
27. \* OutputCollector接口的collect(k, v)方法:增加一个(k,v)对到output
28. \*/
29. **public** **void** map(LongWritable key, Text value,
30. OutputCollector<Text, IntWritable> output, Reporter reporter)
31. **throws** IOException
32. {
33. String line = value.toString();
34. StringTokenizer tokenizer = **new** StringTokenizer(line);
35. **while** (tokenizer.hasMoreTokens())
36. {
37. word.set(tokenizer.nextToken());
38. output.collect(word, one);
39. }
40. }
41. }
43. **public** **static** **class** Reduce **extends** MapReduceBase **implements**
44. Reducer<Text, IntWritable, Text, IntWritable>
45. {
46. **public** **void** reduce(Text key, Iterator<IntWritable> values,
47. OutputCollector<Text, IntWritable> output, Reporter reporter)
48. **throws** IOException
49. {
50. **int** sum = 0;
51. **while** (values.hasNext())
52. {
53. sum += values.next().get();
54. }
55. output.collect(key, **new** IntWritable(sum));
56. }
57. }
59. **public** **static** **void** main(String[] args) **throws** Exception
60. {
61. /\*\*
62. \* JobConf：map/reduce的job配置类，向hadoop框架描述map-reduce执行的工作
63. \* 构造方法：JobConf()、JobConf(Class exampleClass)、JobConf(Configuration conf)等
64. \*/
65. JobConf conf = **new** JobConf(WordCount.**class**);
66. conf.setJobName("wordcount");           //设置一个用户定义的job名称
68. conf.setOutputKeyClass(Text.**class**);    //为job的输出数据设置Key类
69. conf.setOutputValueClass(IntWritable.**class**);   //为job输出设置value类
71. conf.setMapperClass(Map.**class**);         //为job设置Mapper类
72. conf.setCombinerClass(Reduce.**class**);      //为job设置Combiner类
73. conf.setReducerClass(Reduce.**class**);        //为job设置Reduce类
75. conf.setInputFormat(TextInputFormat.**class**);    //为map-reduce任务设置InputFormat实现类
76. conf.setOutputFormat(TextOutputFormat.**class**);  //为map-reduce任务设置OutputFormat实现类
78. /\*\*
79. \* InputFormat描述map-reduce中对job的输入定义
80. \* setInputPaths():为map-reduce job设置路径数组作为输入列表
81. \* setInputPath()：为map-reduce job设置路径数组作为输出列表
82. \*/
83. FileInputFormat.setInputPaths(conf, **new** Path(args[0]));
84. FileOutputFormat.setOutputPath(conf, **new** Path(args[1]));
86. JobClient.runJob(conf);         //运行一个job
87. }
88. }

cd hadoop

bin/hadoop fs –mkdir hdfstest

bin/hadoop fs –put 本地目录/file hdfstest/

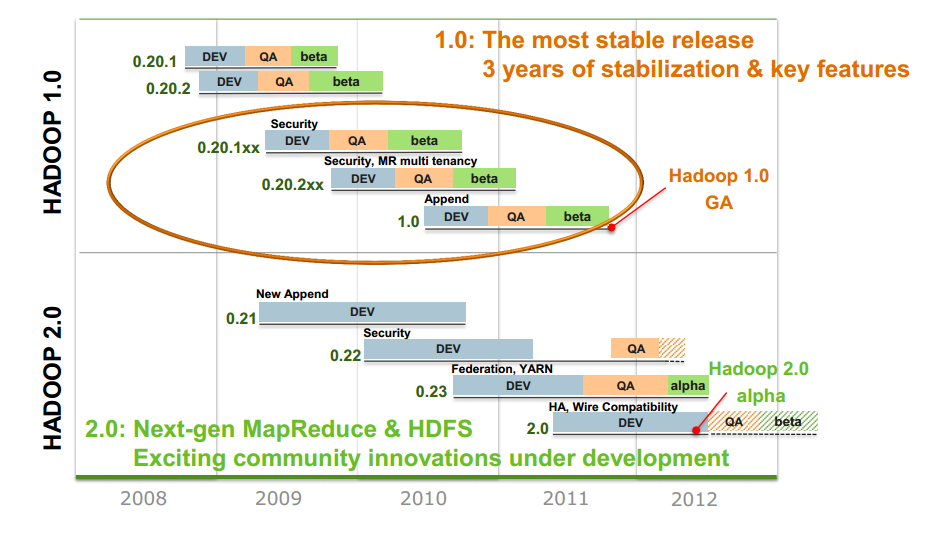
hadoop jar yourjar.jar com.tseg.test.wordcount hdfstest/file hdfstest/output

# Hadoop参数调整

Hadoop在不同环境中需要根据CPU、内存、节点数、磁盘、网络环境设置相应的参数，主要是设置Hadoop/conf文件夹下的core-site.xml hdfs-site.xml mapred-site.xml。



# Hadoop 2.0 简介



目前已发布Hadoop 2.1 beta版

## 1. HDFS HA (manual failover)

Hadoop HA分支merge进了该版本，并支持热切，主要特性包括：

（1） NN配置文件有改变，使得配置更加简单

（2） **NameNode分为两种角色：active NN与 standby NN，active NN对外提供读写服务，一旦出现故障，便切换到standby NN。**

（3） 支持Client端重定向，也就是说，当active NN切换到standby NN过程中，Client端所有的进行时操作都可以无缝透明重定向到standby NN上，**Client自己感觉不到切换过程**。

（4） DN同时向active NN和standby NN汇报block信息。

具体设计文档参考：<https://issues.apache.org/jira/browse/HDFS-1623>

## 2. HDFS Federation

**传统HDFS是master/slave结构，其中，master（也就是NameNode）需要存储所有文件系统的元数据信息，且所有文件存储操作均需要访问多次NameNode，因而NameNode成为制约扩展性的主要瓶颈所在**。为了解决该问题，引入了HDFS Federation，**允许HDFS中存在多个NameNode，且每个NameNode分管一部分目录**，而DataNode不变，也就是“**从中央集权专政变为各个地方自治**”，进而缩小了故障带来的影响范围，并起到一定的隔离作用。具体参考：

<http://dongxicheng.org/mapreduce-nextgen/nextgen-mapreduce-introduction/>

## 3. 性能基准测试

该版本中**为HDFS和YARN添加了性能的基准测试集**，

其中HDFS测试包括：

（1） **dfsio基准** 测试HDFS I/O读写性能

（2）**slive基准** 测试NameNode内部操作的性能

（3）**scan基准** 测试MapReduce作业访问HDFS的I/O性能

（4） **shuffle基准** 测试shuffle阶段性能

（5） **compression基准** 测试MapReduce作业中间结果和最终结果的压缩性能

（6） **gridmix-V3基准** 测试集群吞吐率

YARN 测试包括:

**（1） ApplicationMaster扩展性基准测试**

主要测试调度task/container的性能。与1.0版本比较，大约快2倍。

**（2） ApplicationMaster恢复性基准测试**

测试YARN重启后，**作业恢复速度**。稍微解释一下ApplicationMaster恢复作业的功能：在作业执行过程中，Application Master会不断地将作业运行状态保存到磁盘上，比如哪些任务运行完成，哪些未完成等，这样，一旦集群重启或者master挂掉，重启后，可复原各个作业的状态，并只需重新运行未运行完成的哪些任务。

**（3） ResourceManager扩展性基准测试**

通过不断向Hadoop集群中添加节点测试RM的扩展性。

**（4） 小作业基准测试**

专门测试批量小作业的吞吐率

具体参考：

<http://hortonworks.com/blog/delivering-on-hadoop-next-benchmarking-performance/>

## 4. Wire-compatibility for both HDFS & YARN

Hadoop RPC采用了Hadoop自己的一套序列化框架对各种对象进行序列化反序列，但存在一个问题：扩展性差，很难添加新的数据类型同时保证版本兼容性。为此，**Hadoop 2.0将数据类型模块从RPC中独立出来，成为一个独立的可插拔模块，这样允许用户根据个人爱好使用各种序列化/反序列化框架**，比如thrift，arvo，protocal Buffer等，默认情况采用Protocal Buffer。

<http://hortonworks.com/blog/rpc-improvements-and-wire-compatibility-in-apache-hadoop/>

## 5. HDFS快照

**用户可在任意时间对HDFS做快照，这样，在HDFS出现故障时，可将数据恢复到某个时间点的状态**。具体参考：

<http://hortonworks.com/blog/snapshots-for-hdfs/>

## 6. HDFS HA自动切换

前面介绍的第一个功能“HDFS HA”当前只能实现人工切换，也就是说，管理员运行某个命令，使得acitve NN切换到standby NN上。**以后将支持自动切换**，也就是说，监控模块可检测出active NN何时出现故障，并自动将之切换到standby NN上，这样可大大较小Hadoop集群运维人员的工作量。具体参考：

<http://s.apache.org/hdfs-autofailover>

## 7. MR2/YARN

**YARN是一套资源统一管理和调度平台，可管理各种计算框架，包括MapReduce，Spark，MPI等**。尽管它是完全重写而成，但其思想是从MapReduce衍化而来的，并克服了它在扩展性和容错性等方面的众多不足。



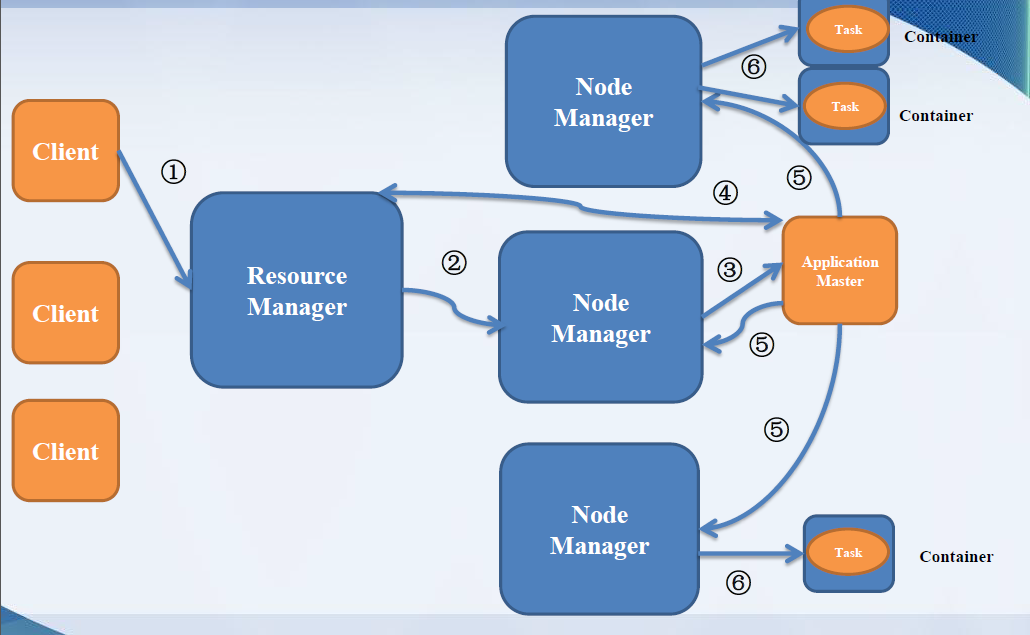
具体参考：

<http://hadoop.apache.org/common/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html>

**MapReduce 在hadoop 0.23版本中经历了一次大修改。现在叫做MapReduce 2.0 (MRv2) 或者 YARN**。

JobTracker在MRv2 中被拆分成了两个主要的功能使用守护进程执行：资源管理和任务的调度与监视。这个想法创建一**个全局的资源管理（global ResourceManager (*RM*)）和为每个应用创建一个应用管理（ApplicationMaster (*AM*)）**。一个应用可以使一个MR jobs的经典场景或者是一串连续的jobs。

ResourceManager 和每个slave节点的NodeManager (*NM*)构成一个资源估算框架。ResourceManager 对在系统中所有应用的资源分配拥有最终的最高级别仲裁权。



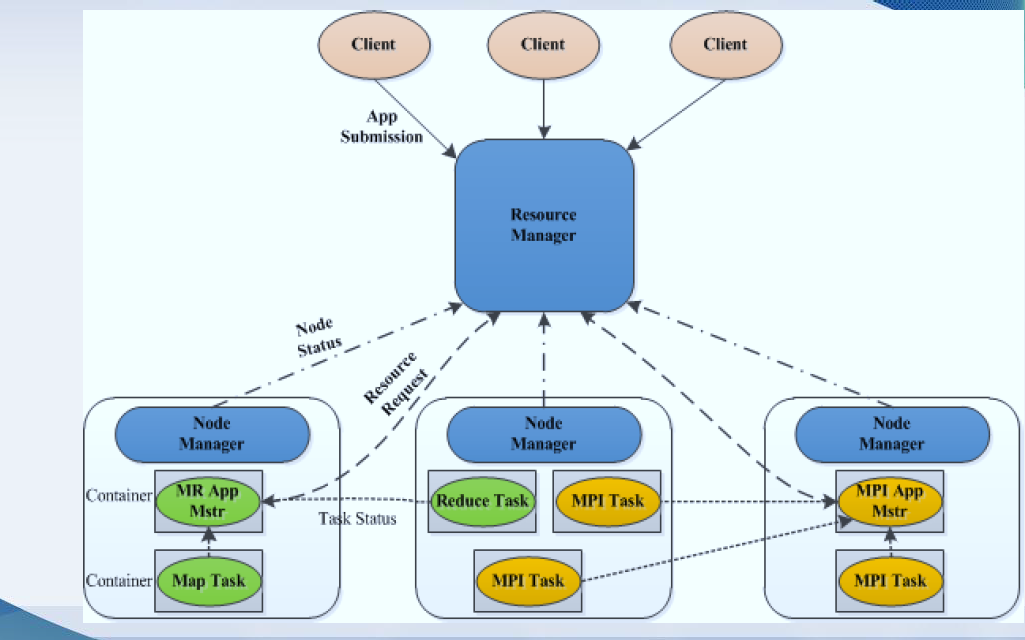
**ApplicationsManager**负责接收 job-submissions，与第一个container谈判用哪个ApplicationMaster 来执行application ，为失败的ApplicationMaster  container提供重启服务。

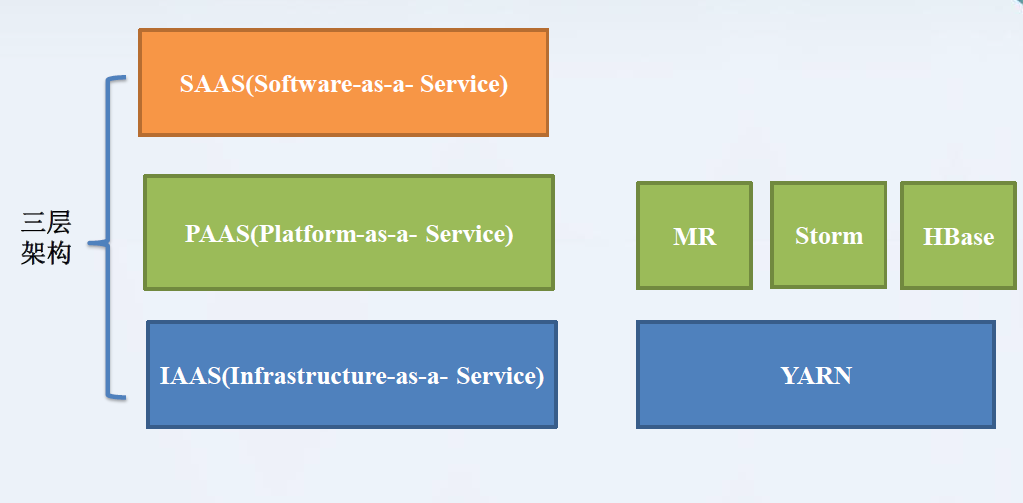
**NodeManager**是每个节点的代理对**container**负责，监控它们的资源使用情况（内存，CPU，硬盘，网络带宽），给 ResourceManager/Scheduler提交报告。

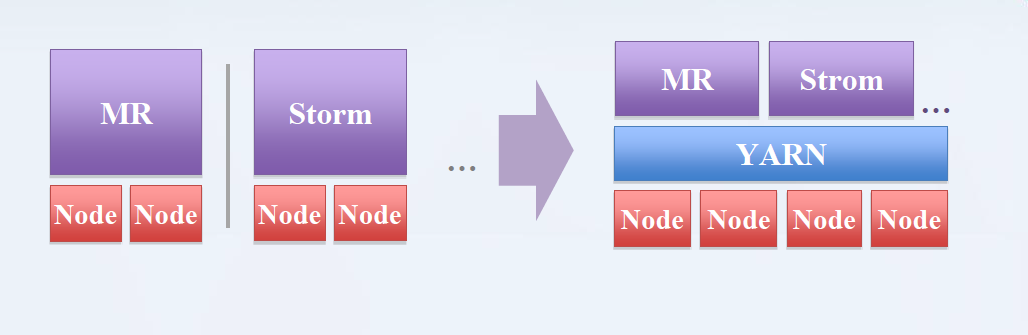
每个应用的**ApplicationMaster**负责与调度器谈判资源占用的**containers**数量，追踪状态和监控进程。

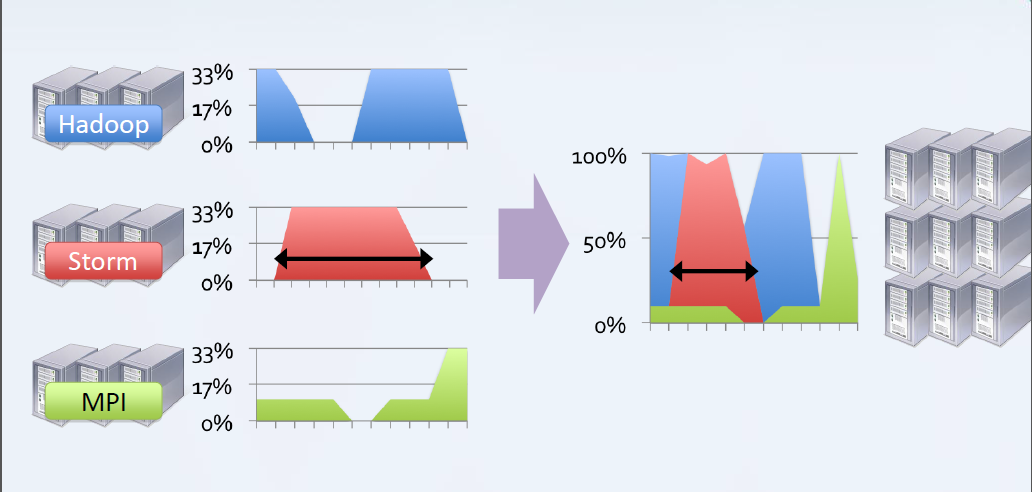


**MRV2 与 之前版本（hadoop-0.20.205）保持兼容。这意味着所有的job只需要重新编译而不做任何修改就可以在MRv2 上运行。**









以下汇总了目前正在进行中的一些尝试将系统运行在YARN上的项目（**很多项目很不完善，有兴趣的可以参与进去，这是机会。**），这些项目涉及分布式领域各个方面的系统，包括离线处理、在线计算、图算法、迭代式算法等，从这些项目可以看出来，在将来，YARN将变成一个轻量级的IAAS层，统一管理和调度各种系统，进而逐步实现所谓的“大集群理念”。

****

1. MapReduce On YARN ：YARN天生支持，目前已非常完善。

2. Tez On YARN ：一个DAG计算框架，直接修改自MapReduce，继承了MapReduce的扩展性好和容错性好等优点， https://issues.apache.org/jira/browse/TEZ。

3. Storm On YARN ：实时计算框架Storm运行在YARN上， https://github.com/yahoo/storm-yarn ，项目状态：开发进行中，已发布一个版本。

4. Spark On YARN ：实时/内存计算框架Spark运行在YARN上： https://github.com/tweetmagik/spark-yarn ，项目状态：已可用。

5. BSP On YARN ：BSP模型在YARN上的实现： https://github.com/jpatanooga/KnittingBoar，项目状态：发布一个实验版本。

6. HBase On YARN ：HBase运行在YARN上， https://issues.apache.org/jira/browse/HBASE-4329，项目状态：进行中，Hortonworks开源的Hoya： http://hortonworks.com/blog/introducing-hoya-hbase-on-yarn/，项目状态：进行中。

7. Kafka On YARN ：Kafka运行在YARN上， https://github.com/kkasravi/kafka-yarn，项目状态：进行中。

8. Giraph On YARN ：图算法库运行在YARN上， https://issues.apache.org/jira/browse/GIRAPH-13 ，项目状态：进行中。

9. MPI on YARN ：MPI运行在YARN上， https://issues.apache.org/jira/browse/MAPREDUCE-2911 ，项目状态：进行中（该项目是所有项目中最难的，简单将mpich移植到YARN上，比如这个实现： mpich2-yarn，意义不大， YARN当前的调度模型决定了难以将MPI移植到YARN上，同时保证MPI本身的各种优化机制不丢失）。

直接在YARN上编写和管理应用程序是比较麻烦的，因此在这两方面，也有一些项目在进行中：

10. YARN Application Management ： http://continuuity.github.io/weave/，项目状态：已可用。

11. Write application on YARN ： https://github.com/cloudera/kitten ，项目状态：已可用。

**PS：github是个好网站，有兴趣的同学可以在上面下载开源代码阅读，对提升编码能力很有帮助。**

# 学习建议

* Hadoop权威指南/ Hadoop技术内幕
* 阅读hadoop源码，更直观，更透彻
* Hadoop1.0已得到广泛应用，但仍存在许多问题。请各位关注下Hadoop2.0
* 使用“印象笔记”、“有道云笔记”等工具管理知识，将遇到的问题、学到的知识点记录下来
* **多实践：自己安装hadoop、自己对比调参效果**
* 主动了解业内新技术：INFQ、CSDN

# 讲义参考文献：

* <http://hadoop.apache.org/> ： hadoop主页
* <http://subject.csdn.net/hadoop/>: csdn上的hadoop专题页
* <http://hbtc.csdn.net/> ： 2012 Hadoop与大数据技术大会PPT， 2013年的会11月召开
* <http://dongxicheng.org/> ：《Hadoop技术内幕》作者的博客，详解hadoop
* <http://blog.sina.com.cn/s/blog_53765cf901016edo.html>： 介绍Hadoop2.0

# 练习

分组完成：

1. Wordcount
2. gourpby
3. 实践hadoop 参数修改和集群启停