# 第二章 关于MapReduce

MapReduce是一种用于数据处理的编程模型，该模型很简单，但想要写出有用的程序却不太容易。Hadoop可以运行各种语言版本的MapReduce程序，在本章中，我们将看到同一个程序的Java Ruby Python C++语言版本。

## 2.1 气象数据集

## 2.2 使用unix工具分析数据

传统处理按行存储数据的awk。在我的github下有源码

为了提高速度我们要并行处理程序来进行数据分析，出现几个问题：

1. 将任务分成大小相同的作业不是容易的事情，有个线程结束的早，就得分配下一个任务，所以总的时间取决于最长的文件的处理时间。
2. 还有一种方式就是将输入数据分成固定大小的块
3. 之后还要合并各个独立进程的运行结果，比如在这个例子中，要把所有结果拼接起来，在按照年份去排序。
4. 最后还是受限于单台计算机的处理能力

## 2.3 使用hadoop来分析数据

### 2.3.1 map和reduce

为了充分利用hadoop的并行处理的优势，我们需要将查询表示成mapreduce作业。

MapReduce作业分成两个阶段，map阶段，reduce阶段。每个阶段都以键值对作为输入输出。

Map阶段的输入是NCDC原始数据，将数据集的每一行作为文本输入，键是\*\*\*\* 我们不需要这个信息，将其忽略。

Reduce阶段基于键来对键值对进行排序和分组。

### 2.3.2 java MapReduce

写代码实现上面的原理，我们需要三个东西：map函数，reduce函数，一些来运行作业的代码。

运行测试要打包成jar文件：

1. 使用javac将所有java文件编译成class文件
2. 使用 jar cvf [\*.jar ] [\*\*\*.calss] 打包成jar包

程序先不去写，熟悉这些api

调用hadoop命令的第一个参数是类名。

在第三版中，将新的API作为主要使用的API。

新旧API之间的几个明显区别：

1. 新API倾向使用虚类，而不是接口，这样更方便扩展。
2. 新的API放在org.apache.hadoop.mapreduce包及其子包中
3. 新API充分使用上下文对象，使用户代码能与mapreduce系统通信。
4. 键值对这两类API都被推给mapper reducer。
5. 新的API中作业控制由Job 类实现
6. 新的API实现了配置的统一
7. 输出文件的命名方式稍有不同 在旧的API中 map和reduce的输出被统一命名为part-nnmm，在新的api中map的输出文件名part-m-nnnn，reduce的输出文件为part-r-nnnn(nnnn 是从0开始的表示分块序号的整数)
8. 新api中用户重载函数被声明为抛出异常java.lang.InterruptedException
9. 在新的API中 reduce传递的值是java.lang.Interable 而非 java.lang.Intertor类型。

不用在意用旧的API实现的代码

## 2.4横向扩展（sacling out）

前面的实验是针对本机上少量的数据，现在要讨论数据来自其他机器上的HDFS。

MapReduce 作业（job）：客户端需要执行一个工作单元：包括输入数据，MapReduce程序和配置信息。

Hadoop将作业分成若干小任务执行，map任务和reduce任务。

有两个节点控制着作业的执行过程，jobtracker tasktracker。Jobtracker 通过调度tasktracker上运行的任务来协调所有运行在系统上的作业。

当前在2.x版本中这两个节点已经没有了，被YARN取代了。

Hadoop将MapReduce的输入数据划分成等长的小数据块，称为分片。Hadoop为每个分片构建一个map任务。

拥有很多分片，意味着处理每个分片的时间少于处理整个输入数据所花的时间。但是如果分片分的太小，那么管理分片的总时间和构建map任务的总时间决定了作业的整个时间。

一个合理的分片大小趋向与HDFS的一个块大小（64MB）

Hadoop 在存储有输入数据的节点运行map任务，可以获得最佳性能，这就是所谓的**“数据本地化优化”**，因为它不需要使用宝贵的集群带宽资源。

为什么最佳分块大小应该与块大小相同？**因为可以确保可以储存在当个节点的最大输入块的大小。如果分片要跨越两个数据块，那么对于任何一个HDFS节点，基本上不可能同时存储着两个数据块。**

**Map任务将其输出写入磁盘**。而非HDFS，为什么？

因为map任务的输出是中间结果，放到HDFS上并备份有点小题大做；如果将map的输出传输给reduce任务失败了，可以在另一个节点再运行reduce任务。

Reduce任务不具备数据本地化的优势。因为reduce任务的输入来自多个map任务的输出，map输出通过网络传输到运行reduce任务的节点。Reduce的数据保存在HDFS中，第一个复本存在本地节点上，其他复本存储在其他机架上。

Reduce任务的数量并非由输入数据的大小决定的，事实上是独立指定的。

如果有很多reduce任务，每个map任务会根据输出进行分区。分区由用户的partition函数控制。

Map任务和reduce任务之间的数据流称为shuffle（混洗）

### 2.4.2 combiner函数

Hadoop用户允许用户针对map任务的输出指定一个combiner——combiner函数的输出作为reduce函数的输入，由于combiner属于优化方案，属于Hadoop无法确定map任务要调用combiner多少次。

Combiner的规则制约着可用函数类型。

比如 l两个map任务输出在combiner 函数的简单筛选之后，再给reduce处理

这样很有效的**减少mapper和reducer之间的数据传输量**。**在MapReduce使用combiner函数要慎重考虑**

Combiner是通过Reducer类来定义的。

## 2.5 Hadoop Streaming

Hadoop提供了MapReduce的API，还允许你使用非java的其他语言来写自己的map函数和reduce函数。Hadoop Streaming使用Unix的标准流作为Hadoop和应用程序之间的接口，所以我们可以通过任何编程语言通过标准输入输出来写MapReduce程序。

Map的输入数据通过标准输入流传递给map函数，并且是一行行的传输，最后将结果行写到标准输出。Map的输出键值对是以一**个制表符分隔**的行。

### 2.5.1 ruby版本

Streaming和 Java MapReduce API 之间的设计差异。Java API 控制的map函数一次只处理一条记录。Streaming中map程序可以自己觉得如何处理输入数据。 Java map 也可以同时处理多行。

### 2.5.2 Python版本

## 2.6 Hadoop Pipes

Hadoop Pipes是Hadoop MapReduce的C++接口名称。