第六章 MapReduce的工作机制

## 6.1 剖析MapReduce作业的运行机制

在hadoop2.0之后引入了YARN，对mapreduce.framework.name 属性设置，local表示本地的作业运行器，“classic”表示经典的MapReduce框架（使用一个jobtracker 和多个tasktracker），yarn表示新的框架。

### 6.1.1 经典的MapReduce1 （跳过）

### 6.1.2 YARN（MapReduce 2）

YARN将JobTracker的3划分为多个实体，从而改善经典的MapReduce面临的扩展瓶颈问题。

YARN将两种角色划分成两个独立的守护进程： 1.管理集群资源使用的资源管理器 2. 管理集群上运行任务生命周期的应用管理器。

应用服务器与资源管理器协商集群的计算资源：容器，在这些容器上运行特定应用程序的进程。容器有集群节点上运行的节点管理器件监视，确保应用程序使用的资源不会超过分配给他的资源。

实际上MapReduce还是YARN应用的一种形式。YARN设计的精妙之处在于不同的YARN应用可以在同一个集群上共存。

YARN上MapReduce比经典的MapReduce包括更多的实体：

提交MapReduce作业的客户端；

YARN的资源管理器

YARN的节点管理器

MapReduce应用程序master负责协调运行MapReduce作业的任务

分布式文件系统

Hadoop使用YARN运行MapReduce的过程

1. 作业提交、

MapReduce2的作业提交api和MapReduce1的相同

从资源管理器获取新的作业ID

作业客户端计算输入分片并将作业资源（作业JAR，配置，分片信息）复制到HDFS。

最后调用资源管理器的SubmitApplication（）方法提交作业。

1. 作业初始化

资源管理器收到submitApplication（）消息后，将请求传递给调度器。调度器分配一个容器，然后资源管理器在节点管理器的管理下在容器中启动应用程序的master进程。

Application master是一个java程序，它的主类是MRAPPMaster。对作业进行初始化。它将接受来自任务的进度和完成报告。接受来**自共享文件系统的在客户端计算的输入分片大小，**

接下来application master决定如何运行构成MapReduce作业的各个任务，如果作业很小就选择在同一个JVM上运行任务。（MapReduce1 从不在单个taskTracker上运行小作业，小于10Mapper且1 Reducer 输入小于宇哥HDFS块的任务。）

1. 任务分配  
   如果作业不适合作为Uber任务（小任务）运行，那么application master会为该作业中的所有map任务和reduce任务向**资源管理器**请求容器。

调度器使用map的数据本地化等信息做调度决策，理想情况下，任务被分配到数据本地化的节点。调度器会相对于非本地化分配优先使用机架本地化的分配。

在YARN上资源的被分为更细的粒度。应用程序请求最小到最大范围内任意最小值倍数的内存容量。

1. 任务执行

一旦资源管理器的调度器为任务分配了容器，application master就通过与节点管理器通信来启动容器。该任务由主类YarnChild的java应用程序执行。

1. 进度和状态更新

在YARN’下运行时，任务每三秒向application master汇报进度和状态

客户端每秒钟查询一次applicationmaster以接受进度更新。

1. 作业完成

除了想Application master 查询进度外，客户端每5秒还通过调用job的waitForCompletion（）来检查作业是否完成。

作业完成后application ，master和任务容器清理其工作状态

## 6.2 失败

### 6.2.1 经典的MapReduce中的失败

### 6.2.2 YARN中的失败

在YARN运行的MapReduce程序，需要考虑一下几种实体的失败

1. 任务运行失败
2. Appliaction master 运行失败

YARN中的应用程序在失败的时候有几次尝试机会。

1. 节点管理器运行失败

如果节点管理器失败，就会停止向资源管理器发送心跳信息，并被移除可用节点资源管理器池。默认值为10分钟决定资源管理器认为节点管理器失效的等待时间。

如果应用程序失败次数过高，那么节点管理器可能被拉黑。

1. 资源管理器运行失败

资源管理器失败后，由管理员启动一个新的资源管理器实例并恢复到保存状态。状态由系统中的节点管理器和运行的程序组成。

## 6.3 作业的调度

MapReduce1 默认调度器是最初基于队列的FIFO的调度器，还有两个多用户调度器：公平调度器，容量调度器

### 6.3.1 公平调度器

作业都放在每个用户的作业池中。公平调度器支持抢占机制，公平调度器是一个后续模块。

### 6.3.2 容量调度器

容量调度器允许用户或组织为每一个用户模拟出一个使用FIFO调度策略的独立的MapReduce集群。公平调度器类似容量调度器。

## 6.4 shuffle和排序

这个部分本书讲的是0.20版本

Shuffle 是MapReduce的心脏，是奇迹发生的地方。

### 6.4.1 Map端

Map函数开始输出时，并不是简单将其写入磁盘，这个过程很复杂，它利用缓冲的方式写到内存并出于效率的考虑进行的预排序。

每个map任务都有一个环形缓冲区用来存储输出，在默认情况下是100MB（io.sort.mb）,y一旦达到阈值（io.sort.spill.percent）缺省0.8 ，一个后台线程遍把内容spill溢出到磁盘中。如果在溢出期间，缓冲区被填满，map会被阻塞知道写磁盘完成。

在写磁盘之前，线程根据数据最终传输的reducer把数据**划分成相应的分**区。每次缓冲区达到阈值是就会新建**溢出文件**（spill file），如果有**combiner**，combiner会使输出的数据更少。所以在map写完最后一个输出时，会有几个溢出文件，在任务完成之前他们会被**合并成一个已分区和排序的输出文件**。

**压缩map输出再写入磁盘**，将mapred.compress.map.output 设置为true，使用的压缩库mapred.map.output.compression.codec.

Reducer通过HTTP方式得到输出文件的分区

在MapReduce中使用Netty，默认情况下是处理数量的两倍。

### 6.4.2 Reduce端

每个map任务的完成时间不同，一旦一个map任务完成，reduce任务就开始复制其输出，默认的复制线程是5，mapred.reduce.parallel.copies属性。

Reduce如何知道map输出在哪个机器上？

在MapReduce中任务直接通知其程序master

为了合并，压缩的map输出在内存中都要解压缩。

复制完map输出之后，Reduce任务进入合并阶段（因为排序在map端已经做过了）

每趟合并的文件数实际上比示例中展示的有所不同，**因为最后一趟总是直接合并到reduce**

### 6.4.3 配置调优

Map端调优属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名称 | 类型 | 默认值 | 说明 |
| Io.sort.mb | Int | 100MB | Map输出的内存缓冲区大小 |
| Io.sort.record.percent | float | 0.05 | 在1.x版本后被删除 |
| Io.sort.spill.percent | Float | 0.80 | Map输出缓冲区的阈值 |
| Io.sort.factor | Int | 10 | 排序中一次最多合并的流数 |
| Min.num.spills.for.combine | int | 3 | 运行combiner所需的最少溢出文件数 |
| Mapred.compress.map.output | Boolean | false | 压缩map输出 |
| Mapred.map.output.compression.codec | Class name | Org.acache.hadoop.io.compress.DefaultCodec | 用于压缩map输出的解码器 |
|  |  |  |  |

**总的原则是给shuffle过程尽量多的内存**

所以编写map函数和reduce函数要少用内存。运行map和Reduce任务的JVM，其大小由mapred.child.java.opts属性设置。

**在map端可以通过避免多次溢出写磁盘来获得最佳性能。一次是最佳情况**

**MapReduce**计数器计算在作业运行阶段溢出写磁盘的记录数。这对调优很有帮助。

**在Reduce端，中间数据全部驻留在内存中，就能获得最佳性能。** 在默认情况下这是不可能发生的，因为所有的内存一般都预留给reduce函数。但如果Reduce函数的内存需求不大，就能提升性能。

Reduce端的调优参数

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名称 | 类型 | 默认值 | 描述 |
| Mapred.reduce.parallel.copies | Int | 5 | 将map输出复制到reducer的线程数 |
| Mapred.reduce.copy.backoff | Int | 300 | 在声明失败后，reducer获得一个map输出所花的最大时间 |
| Io.sort.factor | Int | 10 | 排序文件一次最多合并的流数量 |
| Mapred.job.shuffle.input.buffer.percent | Float | 0.70 | 在shuffle阶段分配给map输出的缓存区占堆空间的百分比 |
| Mapred.job.shuffle.merge.percent | Float | 0.66 | Map 输出缓存区的阈值使用比例 |
| Mapred.inmem.merge.threshold | Int | 1000 |  |
|  |  |  |  |

Hadoop使用默认的4KB的缓冲区，所以在集群中要增加这个值。

2008年，Hadoop在通用TB字节排序基准测试中获胜，它使用了一个优化方法，将中间数据保存中Reduce的内存中。

## 6.5 任务的执行

MapReduce用户对任务执行的更多控制。

### 6.5.1 任务执行环境

Hadoop为map和Reduce任务提供任务运行环境的相关信息。

Streaming环境变量

### 6.5.2 推测执行

可能出现部分任务运行缓慢，出现拖后腿现象。

**Hadoop不会尝试诊断或修复执行慢的任务**。相反，在一个任务运行比预期慢的时候，它会尽量检测，并启动一个相同的任务作为备份。这就是所谓的任务的**“推测执行**”

如果**同时启动两个重复的任务，**他们会互相竞争，导致推测执行无法工作，并且对集群资源也是一种浪费。但是在**作业的所有任务都启动之后才启动推测任务**，一个任务完成，任何在运行的重复任务都会被关闭。

推测执行时一种优化措施