MR-JOB执行过程及JOB提交源码分析

2015 06-26

东风化宇 Hadoop 围观4662次 6 条评论

本文的行文思路如下：

在上文《Hadoop MapReduce编程模型》中提到MapReduce主要有两部分组成：编程模型和运行时环境，并对MR编程模型进行了介绍。

本文继续介绍MR运行时环境，主要包括如下三部分：

- YARN框架简介
- MR JOB的执行过程分析
- JOB提交阶段的源码分析

本文中对一些专有名词的表述可能出现中英文以及缩略词混用的情况，事先做个约定：

> Job —— 作业
>
> Task —— 任务
>
> MR —— MapReduce
>
> RM —— ResourceManager
>
> NM —— NodeManager
>
> AM —— AppMaster —— ApplicationMaster

一、YARN框架简介

对于节点数超出4000的大型集群，Hadoop MapReduce(v1)开始面临着扩展性的瓶颈（原因后文会有分析）。2010年，雅虎的一个团队开始设计下一代的MapReduce框架，YARN应运而生。

YARN负责管理集群上的各种资源，它也是master/slave结构：ResourceManager为master，NodeManager为slave，RM负责对NM上的资源进行统一管理和调度。当用户提交一个应用程序时（可以是MR程序、Spark程序或Storm程序等），会启动一个对应的ApplicationMaster用于跟踪和管理这个程序。它负责向RM申请资源，并在NM上启动应用程序的子任务。

YARN的基本组成结构（组件）如下：

1）ResourceManager

RM是一个**全局**的资源管理器，负责整个YARN集群上的资源管理和分配。它由如下两个组件构成：

- Scheduler(调度器)：调度器根据各个应用的资源需求进行资源分配。资源分配单位用一个抽象概念“资源容器”（Resource Container，简称Container）表示。
- Applications Manager(应用程序管理器)：应用程序管理器负责管理整个系统中的所有应用程序，如启动应用程序对应的ApplicaitonMaster、监控AM运行状态并在失败时重启它。

2）ApplicationMaster

当客户端提交一个应用程序至YARN集群上时，启动一个对应的AM用于跟踪和管理这个程序。AM的主要功能包括：

- 向RM调度器请求资源
- 在NM上启动/停止任务
- 监控所有任务的运行状态，并在任务运行失败时重新为任务申请资源以重启任务

注：AM是YARN对外提供的一个接口，不同的计算框架提供该接口的实现，如MRAppMaster、SparkAppMaster等等，使得该类型的应用程序可以运行在YARN集群上。

3）NodeManager

NM是**每个节点上**的资源和任务管理器。NM的主要功能包括：

- 周期性向RM汇报本节点上的资源使用情况和Container的运行状态
- 接收并处理来自AM的任务启动/停止等各种请求

4）Container

Container是YARN中的资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起。当AM向RM申请资源时，RM为AM返回的资源便是用Container表示的。

扩展阅读：

AM发送的资源请求形式如**<Priority，HostName，Resource，#Containers>**，说明如下：

- Priority：本次请求的优先级，值越小优先级越高。RM将优先为优先级高的请求分配资源。
- HostName：期望分配的资源所在节点。
- Resource：表示需要的资源量。
- #Containers：本次请求所需“符合以上资源描述的Container的数量”。

举例说明：“<10, nodeX, memory:2GB | CPU:1, 2>”表示请求2个满足以下条件的Container：位于节点nodeX上、2GB内存、1个CPU，请求的优先级为10。这意味着，如果没有优先级更高的资源请求，需要优先分配这两个Container，此时RM返回一个Container列表，每个Container主要包括如下内容：<ContainerID, NodeId, NodeHttpAddress, Resource, ContainerToken>。

当AM收到该Container后，将与Id为NodeId的NM通信，并使用ContainerToken进行安全认证，以要求它启动一个占用Resource资源量的Container，并在Container中启动任务。

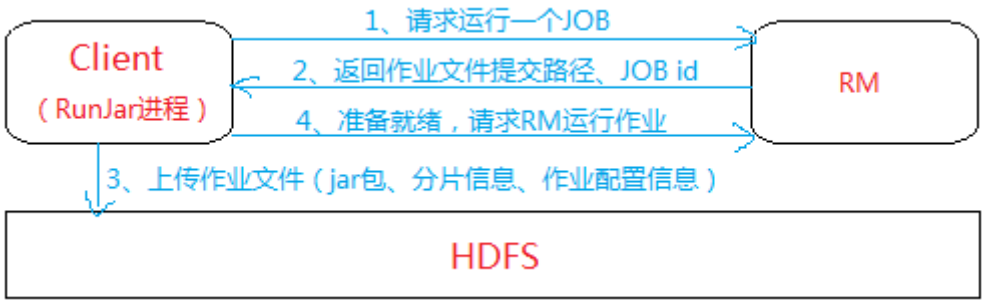
二、MR JOB的执行过程分析

MR-JOB的执行过程是指：从使用hadoop jar命令提交一个MR-JOB至作业结束。这个过程可分为如下三个阶段：

- 作业提交阶段
- 作业的初始化阶段
- 任务调度阶段

1、作业提交阶段

作业提交阶段主要在客户端完成，过程如下图：



当作业提交成功后，在HDFS上可以查看到相关作业文件：

```
[hadoop@hadoop01 ~]$ hadoop fs -ls /tmp/hadoop-yarn/staging/hadoop/.staging/job_1435030889365_0001
Found 4 items
-rw-r--r--  10 hadoop supergroup     270265 2015-06-22 20:46 /tmp/hadoop-yarn/staging/hadoop/.stag
001/job.jar
-rw-r--r--  10 hadoop supergroup        123 2015-06-22 20:46 /tmp/hadoop-yarn/staging/hadoop/.stag
001/job.split
-rw-r--r--   1 hadoop supergroup         22 2015-06-22 20:46 /tmp/hadoop-yarn/staging/hadoop/.stag
001/job.splitmetainfo
-rw-r--r--   1 hadoop supergroup      77015 2015-06-22 20:46 /tmp/hadoop-yarn/staging/hadoop/.stag
001/job.xml
```

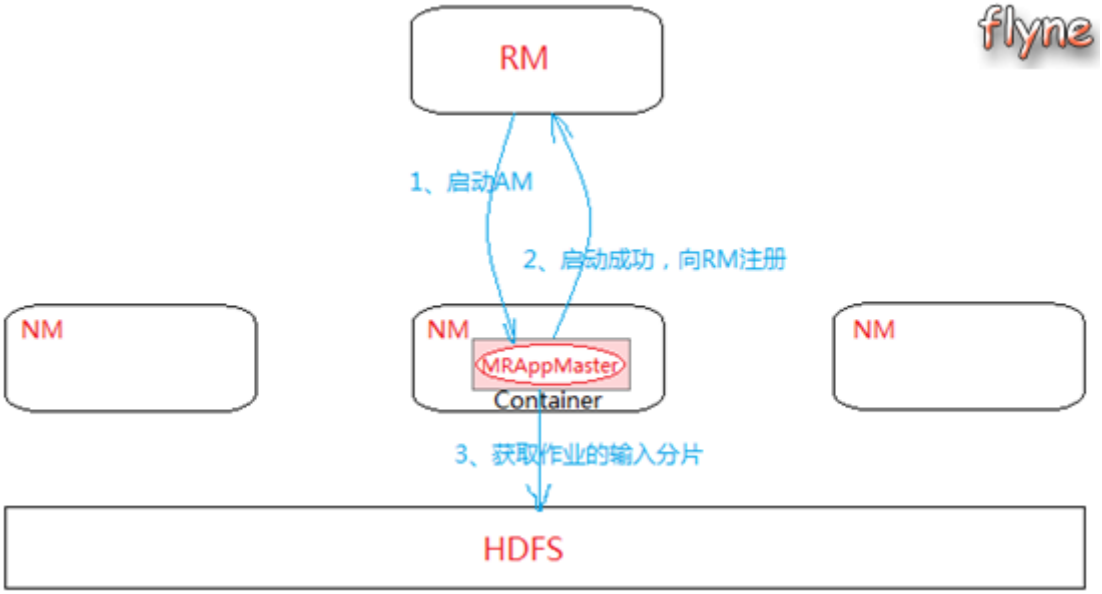
这些文件都是在Client端生成并上传至HDFS，对这些作业文件的说明如下：

- 作业jar包：job.jar
- 作业输入分片信息：job.split、job.splitmetainfo
- 作业配置信息：job.xml

这些作业文件存放在/tmp/hadoop-yarn/staging/hadoop/.staging/job_1435030889365_0001路径下，该路径又称为作业提交路径submitJobDir，它包括如下两部分：

- jobStagingArea：/tmp/hadoop-yarn/staging/hadoop/.staging是作业的staging目录
- jobID：唯一标识集群上的一个MR 任务

2、作业的初始化阶段



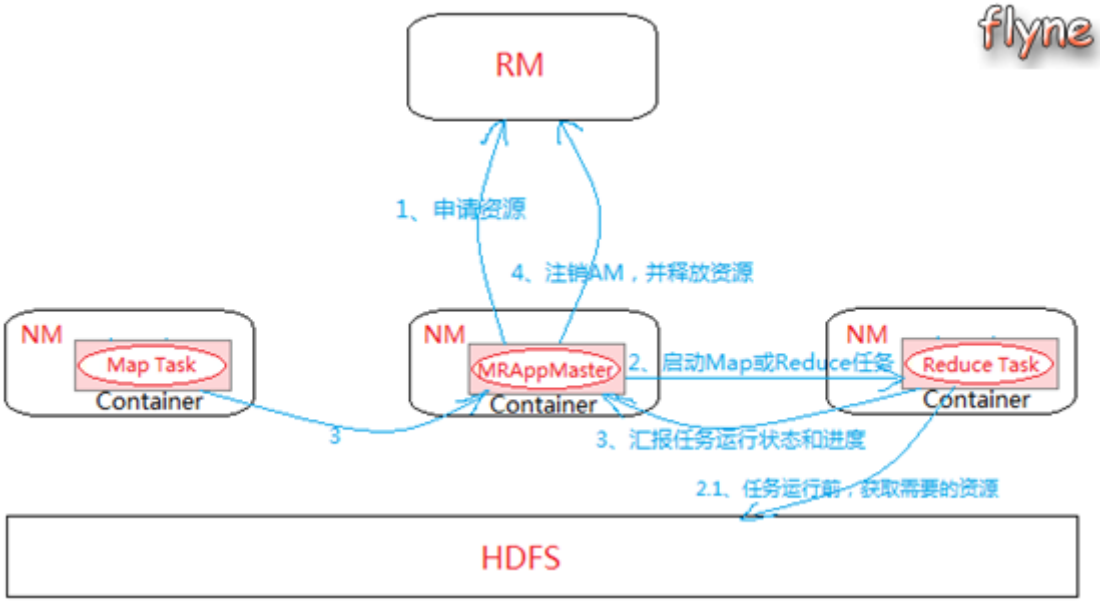
对上图更详细的表述如下：

步骤1：当MR作业提交至YARN后，RM为该作业分配第一个Container，并与对应的NM通信，在Container中启动作业的MRAppMaster。

步骤2：MRAppMaster向RM注册自己。这使得客户端可以直接通过RM查看应用程序的运行状态。

步骤3：MRAppMaster读取作业的输入分片。作业的分片数(split)决定了启动的map任务数。

3、作业执行阶段



对上图更详细的表述如下：

步骤1：MRAppMaster采用轮询的方式向RM申请任务所需资源。

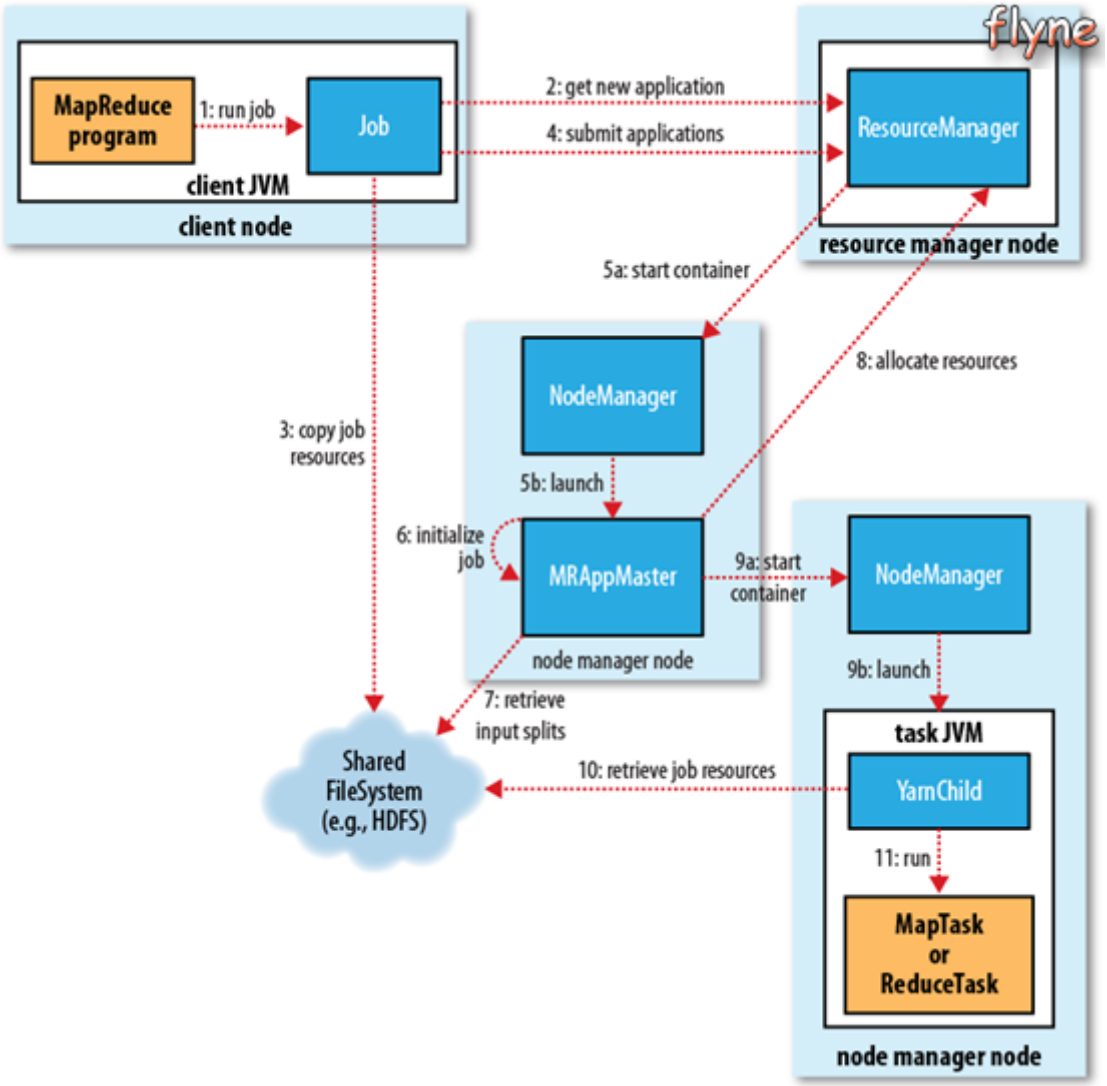
步骤2：资源分配成功后，MRAppMaster就与对应的NM通信，并启动YarnChild来执行具体的Map任务和Reduce任务。在任务运行前，还要将任务所需文件本地化（包括作业Jar包、配置信息等）。

注：每个输入分片对应一个Map任务；Reduce任务数由mapreduce.job.reduces属性确定。

步骤3：在YARN上运行时，任务每3秒钟向AM汇报进度和状态。

步骤4：应用程序运行完成后，AM向RM注销自己，并释放所有资源。

至此，MR作业执行结束。下图附上《Hadoop权威指南》中关于YARN上运行MR作业的流程图，如果对本章节贴出的流程图有疑问，可以留言。



说明：上述步骤并没有考虑Shuffle过程，这是为了方便表述。关于Shuffle过程参考本文最后一节。

4、MRv1的不足

以上分析了YARN上MR程序的执行过程（MRv2的设计），下面对比MRv2说明MRv1的设计及其不足，从而让读者理解文章开头所说的“对于节点数超出4000的大型集群，MRv1开始面临着扩展性的瓶颈”。

在继续后文之前，先补充一句话（不知道放哪好）：MRv1也是master/slave结构，JobTracker为master，TaskTracker为slave。

1）在MRv2中，设计了RM和AM：RM用于响应客户端请求，以及任务执行过程中的资源分配，AM用于任务调度和监控，分工明确。在MRv1中，JobTracker相当于RM + AM，既要做资源分配，又要做任务调度，因此任务量很大。

2）在MRv2中，每个JOB都对应一个AM，提交一个Job产生一个MRAppMaster，这样一个MRAppMaster只要监控一个Job。而在MRv1中，整个集群中的JobTracker只有一个，如果在一个集群中提交了1000个JOB执行，那么JobTracker就要负责监控这1000个JOB的状态，负载很大。

综上，在MRv1中，整个分布式计算过程都由MR框架实现（资源分配 + 任务管理），负载很大。YARN框架将JobTracker的只能划分为多个独立的实体，从而改善了MRv1面临的扩展瓶颈问题。

扩展阅读：

YARN提供了一个AppMaster编程接口，如果让计算框架在YARN框架上运行，只需实现APPMaster接口即可，在实现类中具体管理自己的任务。因此，YARN上既可以运行MR程序，也可以跑Spark、Storm程序，只需实现自己的任务调度即可（StormAppMaster、SparkAppMaster）。这也使得YARN框架对任务的执行没有任何侵入，任务怎么执行（如分成Map任务、Reduce任务），YARN不懂。

因此，可以说引入YARN框架是Hadoop的一个里程碑事件，它使得Hadoop变得更通用。

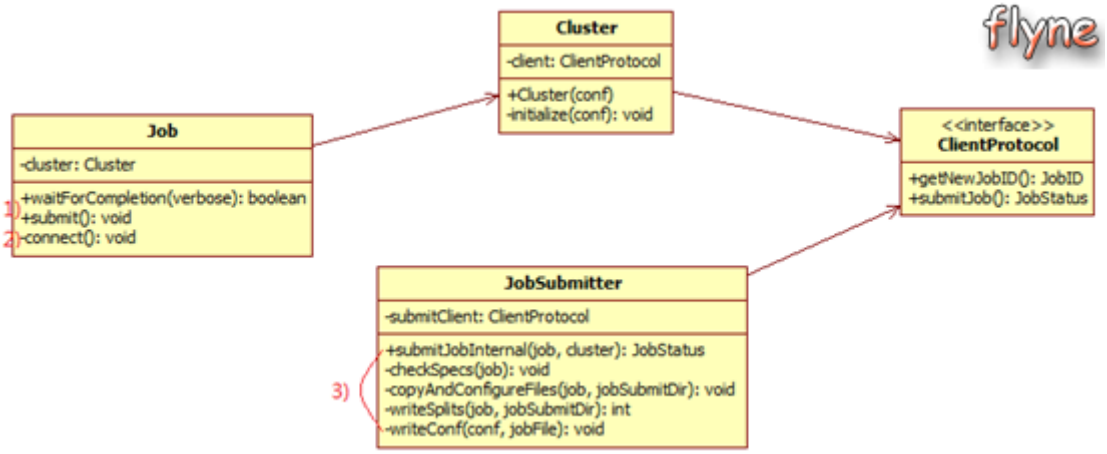
三、JOB提交阶段的源码分析

上一节对MR JOB的执行过程进行了分析，这里对其中的作业提交阶段的源码进行分析。这一阶段涉及到的类有：

- **Job**：给用户使用的类，可以用来设置作业相关信息、提交作业以及查询作业执行状态等。

- **JobSubmitter**：作业提交器，它是真正的提交作业到集群上的类，框架内部使用。
- **Cluster**：用于获取MR集群信息（RPC调用）。
- **ClientProtocol**：客户端和RM进行RPC通信时遵循的通信协议。

它们的关系如下类图所示：



JOB提交阶段的源码分析如下（使用wordcount程序为例）：

1）在main方法中调用job.waitForCompletion(true)，waitForCompletion中又调用了job.submit()，于是，提交过程开始了。submit函数如下：

```
01  public void submit(){
02    ……
03    //主要是获取客户端和RM进行RPC通信时使用的代理对象
04    connect();
05
06    //获取JobSubmitter对象，并调用其submitJobInternal方法
07    final JobSubmitter submitter =
08        getJobSubmitter(cluster.getFileSystem(), cluster.get
09    ……
10    submitter.submitJobInternal(Job.this, cluster);
11    ……
12   }
```

2）job.connect()：话说JOB提交至YARN上的阶段主要是客户端和RM进行RPC通信，这里使用的通信协议是ClientProtocol，因此需要在客户端获得ClientProtocol的代理对象，这是通过job.connect()完成的。

```
01  private synchronized void connect(){
02    ……
03      cluster =  new Cluster(conf);
04    ……
05  }
06
07  public Cluster(Configuration conf){
08    ……
09    initialize(null, conf);
10  }
11
12  private void initialize(null, conf) {
13    ……
14    //frameworkLoader中封装了LocalClientProtocolProvider和Yar
```

```
    for (ClientProtocolProvider provider : frameworkLoader)

      ClientProtocol clientProtocol = null;
      ……
      //这里拿到RPC通信使用的代理对象
      clientProtocol = provider.create(conf);
      if (clientProtocol != null) {
        ……
        //保存到Cluster中的client成员变量
        client = clientProtocol;
        break;
      } //end if
      ……
    } //end for
  }
```

3）submitter.submitJobInternal():客户端拿到通信的代理对象后，接下来就是和RM进行通信，并完成一系列操作，主要包括：

①检查作业输出路径是否存在，若存在抛出异常。

②获取jobStagingArea、jobID，并拼接出submitJobDir。submitJobDir目录用于存放客户端提交的作业文件。

③提交作业jar文件(job.jar)

④计算作业的输入分片，并提交分片信息(job.split、job.splitmetainfo)

⑤将配置对象（conf）写入job.xml

⑥客户端准备就绪，请求RM运行作业

submitter.submitJobInternal()中的主要代码如下：

```
  JobStatus submitJobInternal(Job job, Cluster cluster)  {
    // ①
    checkSpecs(job);

    ……

    // ②
    Path jobStagingArea = JobSubmissionFiles.getStagingDir
    ……
    JobID jobId = submitClient.getNewJobID();
    job.setJobID(jobId);
    Path submitJobDir = new Path(jobStagingArea, jobId.toS

    ……
    try {
      ……

      // ③
      copyAndConfigureFiles(job, submitJobDir);

      // ⑤-1 获取作业配置文件(job.xml)的路径
      Path submitJobFile = JobSubmissionFiles.getJobConfPa

      // ④
      int maps = writeSplits(job, submitJobDir);
      conf.setInt("mapreduce.job.maps", maps);
```

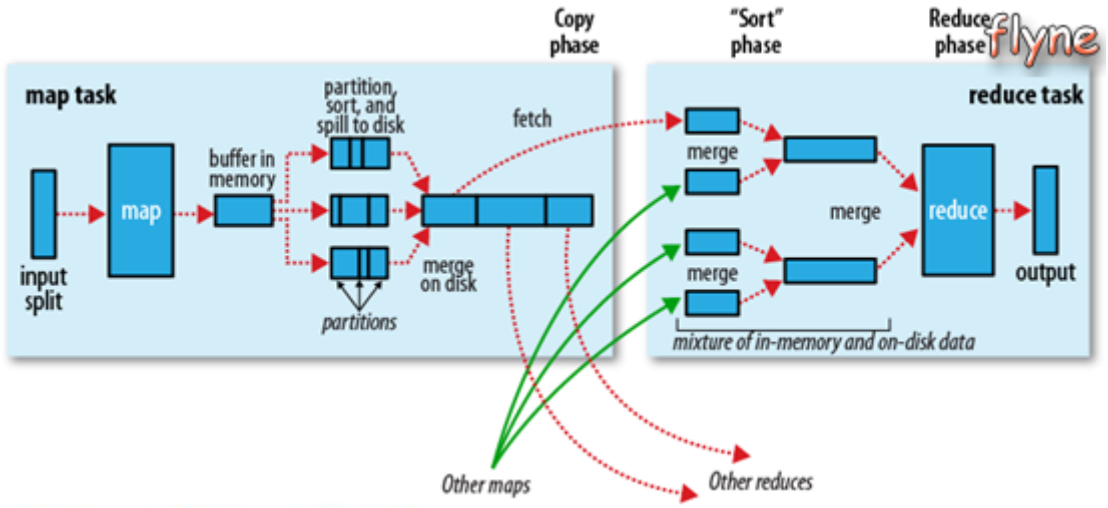
```
29
30          ……
31
32          // ⑤-2 Write job file to submit dir
33          writeConf(conf, submitJobFile);
34
35          ……
36          // ⑥
37          submitClient.submitJob(
38              jobId, submitJobDir.toString(), job.getCredentia
39          ……
40        } finally {
41          ……
42        }
43      }
```

关于每个步骤的细节不作分析，有兴趣可以查看相应的源码，还是容易看懂的。至此，JOB提交阶段结束。

四、补充

1、Shuffle过程

从Map任务产生输出到Reduce任务消化输入的整个过程就叫Shuffle。下图展示了这一过程：



1）Map端

- 每个map有一个环形内存缓冲区，用于存储任务的输出。默认大小100MB（io.sort.mb属性），一旦达到阀值0.8（io.sort.spill.percent），一个后台线程便开始把缓冲区中的内容写到磁盘上的溢出文件(spill file)，该文件位于mapred.local.dir指定的目录下。
- 写磁盘前，要对数据分区并排序（partition、sort）。如果有combiner，还要对排序后的数据进行combine。
- 最后合并全部溢出文件为一个分区且排序的文件，Map任务结束。

2）Reduce端

- NM为分区文件运行Reduce任务。
- 复制阶段通过Http方式把Map输出文件的分区复制到Reducer的内存或磁盘。一个Map任务完成，Reduce就开始复制输出。
- 排序阶段合并Map输出。
- Reduce阶段，将数据输入reduce函数。

Hadoop基础

Hadoop MapReduce编程模型 ⇒

您可能还会对这些文章感兴趣！

- Hadoop MapReduce编程模型
- Hadoop RPC机制及HDFS源码分析
- HDFS体系结构及操作(命令行、Java
- Hadoop基础及伪分布式环境搭建
- Hadoop、Storm学习准备篇

《*MR-JOB执行过程及JOB提交源码分析*》有 6 条评论

方便面 说：
2015年9月11日下午1:52
博主你好，步骤2：MRAppMaster向RM注册自己。这使得客户端可以直接通过RM查看应用程序的运行状态。这里主要指的是客户端ClientProtocol通过该代理就可以查看集群的总体运行情况了吗？可以这样理解吗？望回复！
回复

方便面 说：
2015年9月11日下午1:59
MRAppMaster对应一个Job，也就是说，该Job在RM上注册了自己，RM就有该Job的信息。所以可以查看。具体的实现是在哪里？还是我的理解都错了？
回复

汪杰宇 说：
2015年9月11日下午5:56
可以这样理解。除了相关的API，Yarn也对外提供了web界面来查看yarn集群上的任务状态（默认的端口号是8088）。可以查阅相关的YARN资料~PS：这个其实在http://www.flyne.org/article/1065 的2.2.3节中有所提及。
回复