Hadoop学习笔记

1. Hadoop架构

1.Hadoop介绍

Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），简称HDFS。HDFS有高容错性的特点，并且设计用来部署在低廉的硬件上；而且它提供高吞吐量来访问应用程序的数据，适合那些有着超大数据集的应用程序。Hadoop的框架最核心的设计就是：HDFS和MapReduce。HDFS为海量的数据提供了存储，MapReduce为海量的数据提供了计算。

二．HDFS

1.HDFS的组成

(1)NameNode：Master节点（也称元数据节点，元数据指用来描述一个文件的特征的系统数据，诸如访问权限、文件拥有者以及文件数据块的分布信息等），是系统唯一的管理者。负责元数据的管理(名称空间和数据块映射信息);配置副本策略；处理客户端请求。

(2)DataNode：数据存储节点(也称Slave节点)，存储实际的数据；执行数据块的读写；汇报存储信息给NameNode。

(3)Sencondary NameNode：分担Namenode的工作量；是NameNode的冷备份；

①热备份：b是a的热备份，如果a坏掉。那么b立即运行代替a的工作

②冷备份：b是a的冷备份，如果a坏掉。那么b不能立即代替a工作。但是b上存储的一些信息，减少a坏掉之后的损失

(4)Client：系统使用者，调用HDFS API操作文件;与NameNode交互获取文件元数据;与DataNode交互进行数据读写。注意：写数据时文件切分由Client完成。

下图2-1是HDFS的基本架构

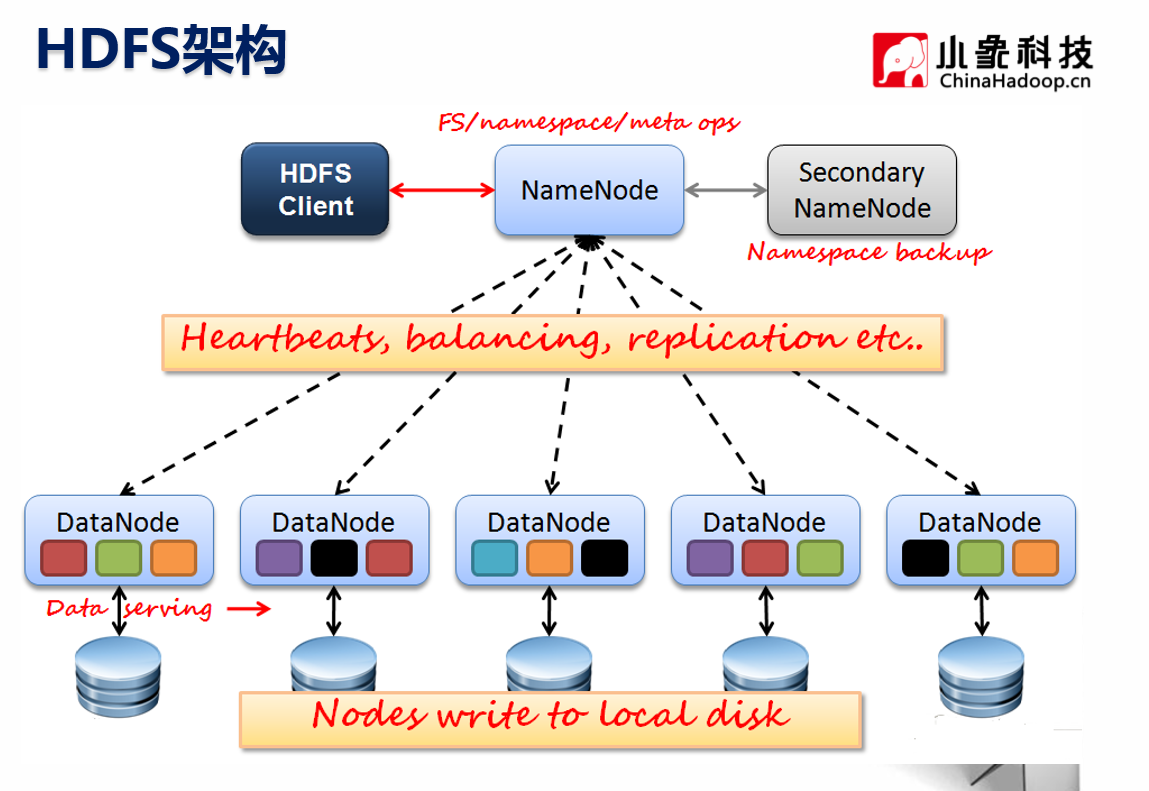


图2-1

HeartBeat：slave启动后，会主动链接IPC服务，并且每隔3秒链接一次，这个时间是可以调整的，设置heartbeat，这个每隔一段时间连接一次的机制，称为心跳机制。Slave通过心跳给master汇报自己信息，master通过心跳下达命令。

2.HDFS的数据块（Block）

(1)文件被切分成固定大小的数据块，默认数据块大小为64MB，可配置。（减小寻址开销）

(2)若文件大小不到64MB，则单独存成一个block。按文件大小被切分成若干个block，存储到不同节点上 默认情况下每个block有三个副本。

图2-2是HDFS中一个文件的分块以及不同分块的副本放置。

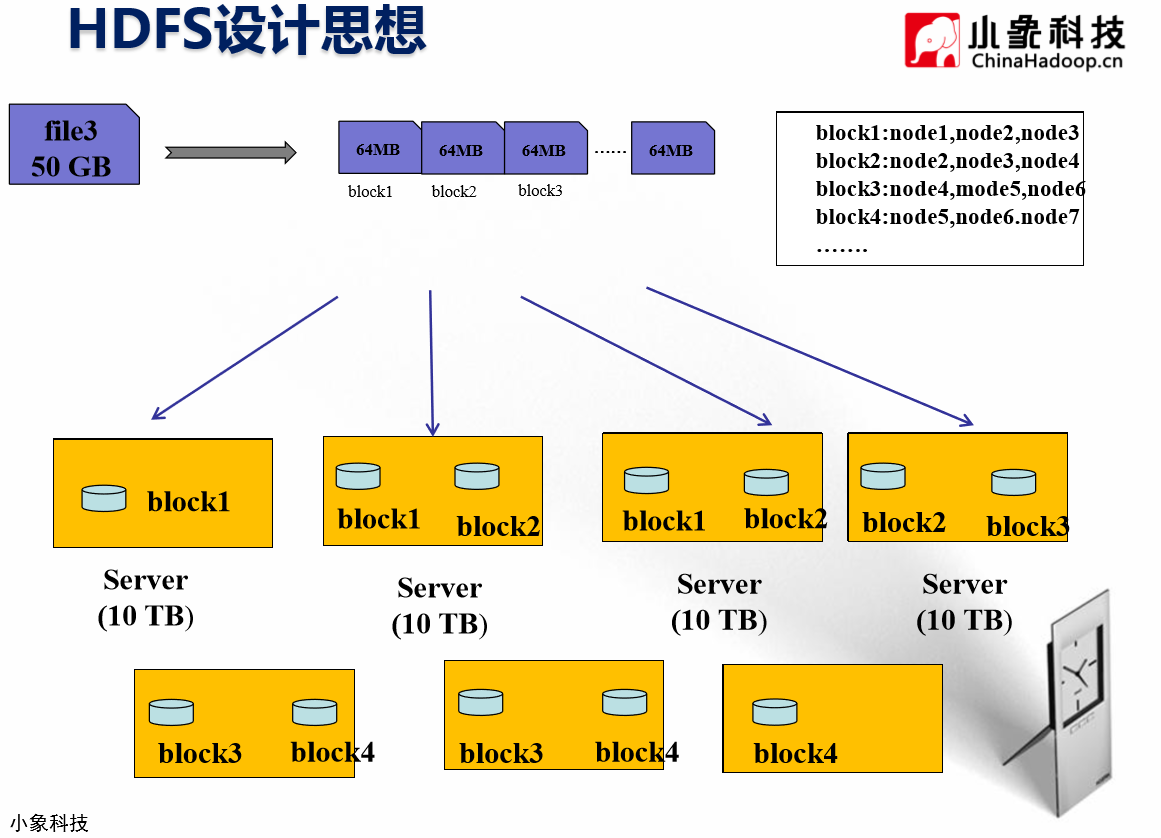


图2-2

3.HDFS中副本放置策略

（1）副本1: 同Client的节点上 

（2）副本2: 不同机架中的节点上 

（3）副本3: 与第二个副本同一机架的 另一个节点上 

（4）其他副本:随机挑选

下图2-3位副本放置的说明图

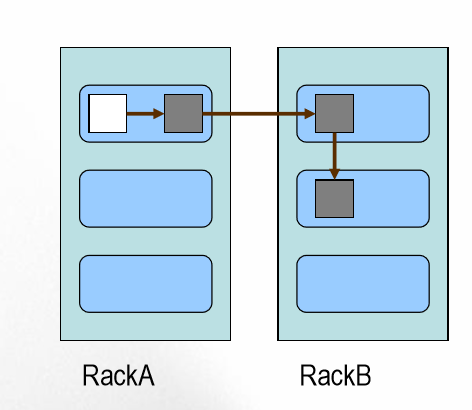


图2-3

4.HDFS的读写过程

(1)读过程交互示意图

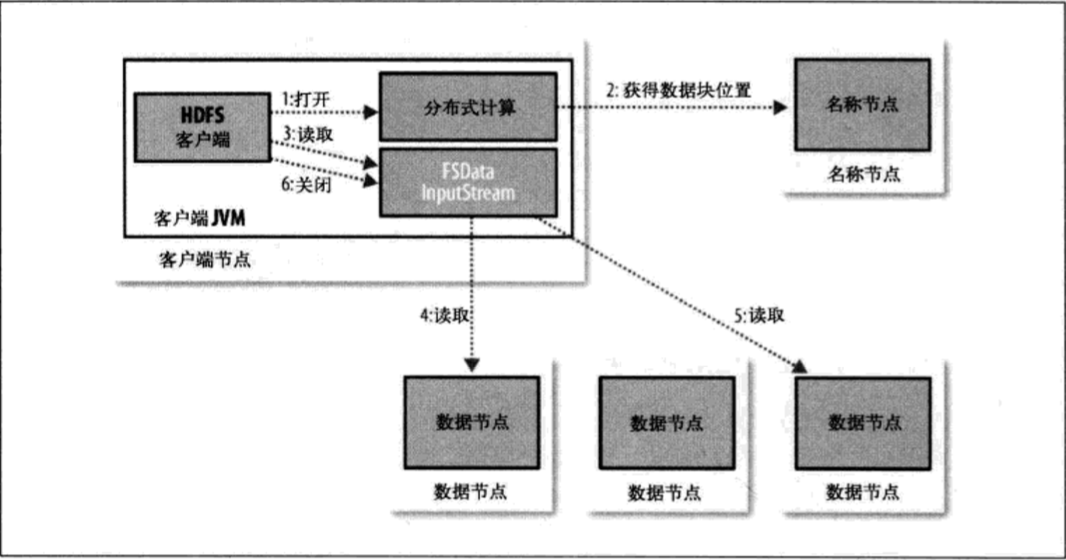
1. **使用HDFS提供的客户端Client，向远程的Namenode发起RPC请求；**
2. **Namenode会视情况返回文件的部分或者全部block列表，对于每个block，Namenode都会返回有该block拷贝的DataNode地址；**
3. **客户端Client会选取离客户端最近的DataNode来读取block；如果客户端本身就是DataNode，那么将从本地直接获取数据；**
4. **读取完当前block的数据后，关闭当前的DataNode链接，并为读取下一个block寻找最佳的DataNode；**
5. **当读完列表block后，且文件读取还没有结束，客户端会继续向Namenode获取下一批的block列表；**
6. **读取完一个block都会进行验证，如果读取dataNode时出现错误，客户端会通知NameNode，然后再从下一个拥有该block拷贝的dataNode继续读。**

图2-4

(2)写过程交互示意图

1. **使用HDFS提供的客户端Client，向远程的Namenode发起RPC请求。**
2. **NameNode会检查要创建的文件是否已经存在，创建者是否有权限进行操作，成功则会为文件创建一个记录，否则会让客户端抛出异常。**
3. **当客户端开始写入文件的时候，客户端会将文件切分成多个packets，并在内部以数据队列“data queue（ 数据队列）”的形式管理这些packets， 并向NameNode申请blocks， 获取用来存储replications的合适的DataNode列表，列表的大小根据NameNode中replication的设定而定。**
4. **开始以pipeline（ 管道） 的形式将packet写入所有的replications中。 客户端把packet以流的方式写入第一个DataNode， 该DataNode把该packet存储之后，再将其传递给在此pipeline中的下一个DataNode， 直到最后一个DataNode，这种写数据的方式呈流水线的形式。**
5. **最后一个DataNode成功存储之后会返回一个ack packet（ 确认队列）， 在pipeline里传递至客户端，在客户端的开发库内部维护着“ack queue”，成功收到DataNode返回ack packe后会从“ack queue”移除相应的packet。**
6. **如果传输过程中，有某个DataNode出现了故障，那么当前的pipeline会被关闭，出现故障的DataNode会从当前的pipeline中移除，剩余的block会继续剩下的DataNode中继续以pipeline的形式传输，同时NameNode会分配一个新的DataNode，保持replications设定的数量。**
7. **客户端完成数据的写入后，会对数据流调用close()方法，关闭数据流；**
8. **只要写入了dfs.replication.min的复本数（ 默认为1），写操作就会成功，并且这个块可以在集群中异步复制，直到达到其目标复本数。（replication的默认值为3），因为NameNode已经知道文件由哪些块组成，所以它在返回成功前只需要等待数据块进行最小量的复制。**

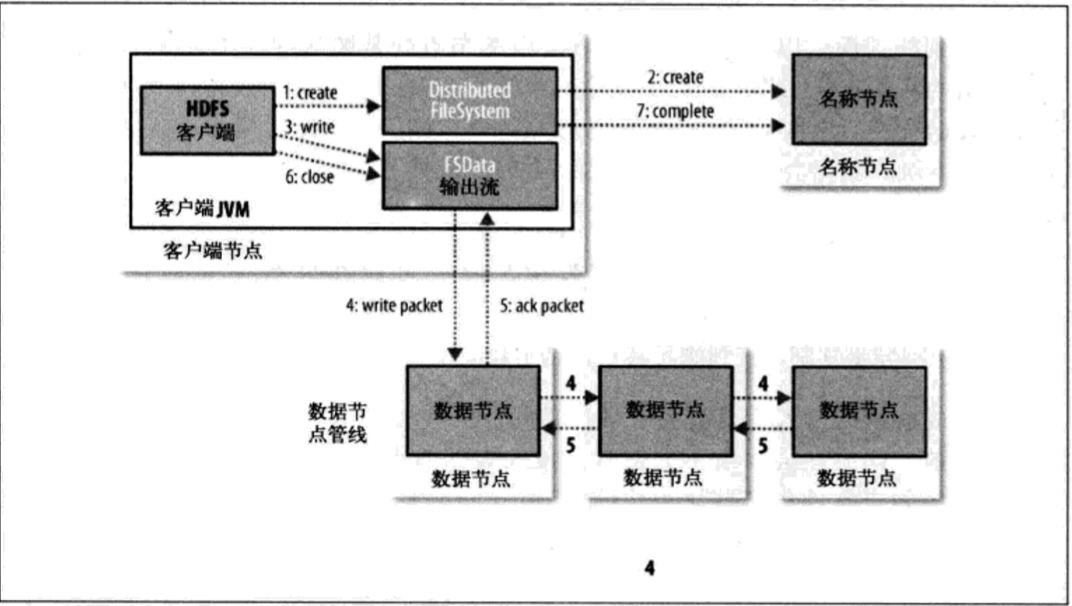


图2-5

三．MapReduce

1. MapReduce主体思想

通过分散计算来分析大量数据。一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是并行计算。

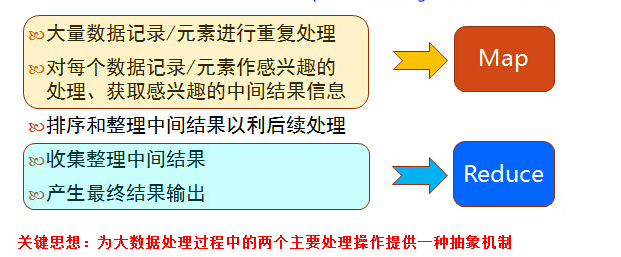


图3-1

2.MapReduce的运行原理

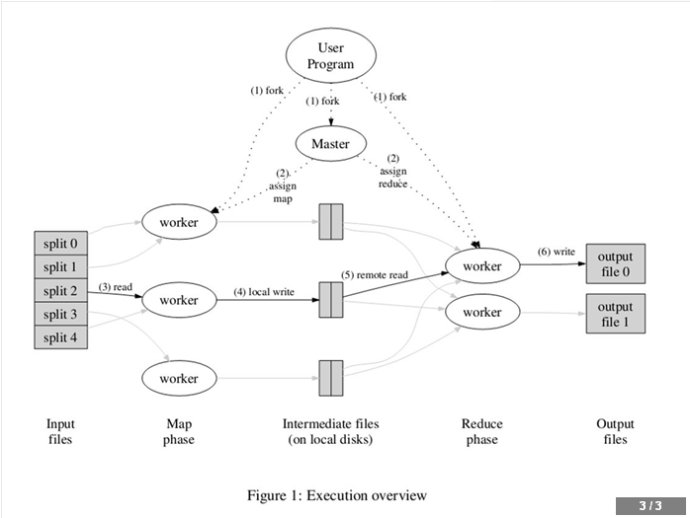


图3-3

(1).MapReduce库先把user program的输入文件划分为M份（M为用户定义），每一份通常有16MB到64MB，如图左方所示分成了split0~4；然后使用fork将用户进程拷贝到集群内其它机器上。

(2).user program的副本中有一个称为master，其余称为worker，master是负责调度的，为空闲worker分配作业（Map作业或者Reduce作业），worker的数量也是可以由用户指定的。

(3).被分配了Map作业的worker，开始读取对应分片的输入数据，Map作业数量是由M决定的，和split一一对应；Map作业从输入数据中抽取出键值对，每一个键值对都作为参数传递给map函数，map函数产生的中间键值对被缓存在内存中。

(4).缓存的中间键值对会被定期写入本地磁盘，而且被分为R个区，R的大小是由用户定义的，将来每个区会对应一个Reduce作业；这些中间键值对的位置会被通报给master，master负责将信息转发给Reduce worker。

(5).master通知分配了Reduce作业的worker它负责的分区在什么位置（不止一个地方，每个Map作业产生的中间键值对都可能映射到所有R个不同分区），当Reduce worker把所有它负责的中间键值对都读过来后，先对它们进行**排序**，使得相同键的键值对聚集在一起。因为不同的键可能会映射到同一个分区也就是同一个Reduce作业（分区少的情况），所以排序是必须的。

(6).reduce worker遍历排序后的中间键值对，对于每个唯一的键，都将键与关联的值传递给reduce函数，reduce函数产生的输出会添加到这个分区的输出文件中。

(7).当所有的Map和Reduce作业都完成了，master唤醒正版的user program，MapReduce函数调用返回user program的代码。

总结：所有执行完毕后，MapReduce输出放在了R个分区的输出文件中（分别对应一个Reduce作业）。用户通常并不需要合并这R个文件，而是将其作为输入交给另一个MapReduce程序处理。整个过程中，输入数据是来自底层分布式文件系统（HGFS）的，中间数据是放在**本地文件系统的**，最终输出数据是写入底层分布式文件系统（HGFS）的。

**Map/Reduce作业和map/reduce函数的区别**：Map作业处理一个输入数据的分片，可能需要调用多次map函数来处理每个输入键值对；Reduce作业处理一个分区的中间键值对，期间要对每个不同的键调用一次reduce函数，Reduce作业最终也对应一个输出文件。

3.Hadoop 2.6后的MapReduce原理：

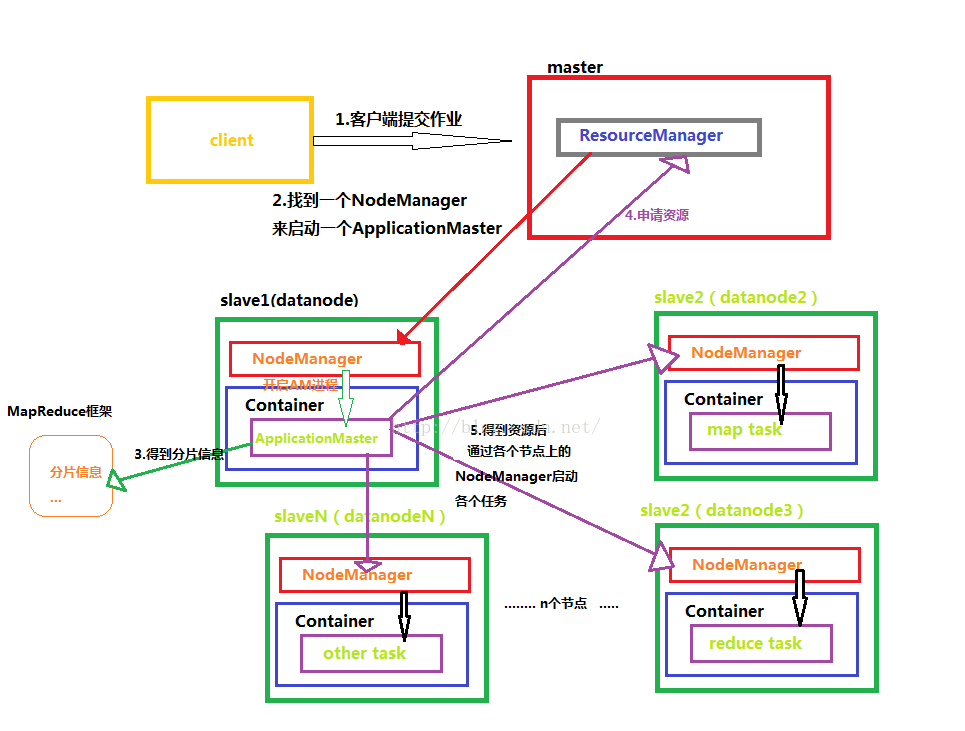
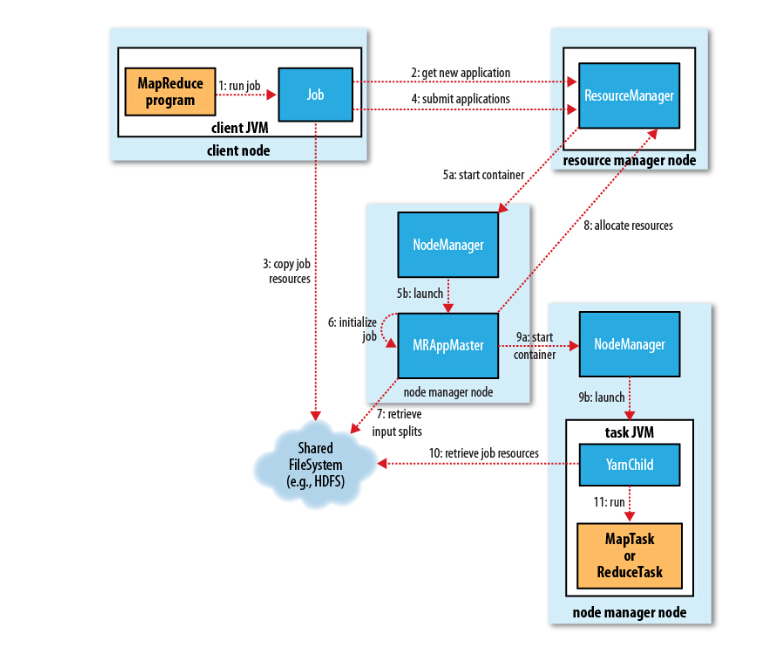


图3-4

**ResourceManager：**是YARN资源控制框架的中心模块，负责集群中所有的资源的统一管理和分配。它接收来自NM(NodeManager)的汇报，建立AM，并将资源派送给AM(ApplicationMaster)。  
**NodeManager**:简称NM，NodeManager是ResourceManager在每台机器的上代理，负责容器的管理，并监控他们的资源使用情况（cpu，内存，磁盘及网络等），以及向 ResourceManager提供这些资源使用报告。  
**ApplicationMaster**:以下简称AM。YARN中每个应用都会启动一个AM，负责向RM申请资源，请求NM启动container，并告诉container做什么事情。  
**Container**：资源容器。YARN中所有的应用都是在container之上运行的。AM也是在container上运行的，不过AM的container是RM申请的。

MapReduce程序源码运行原理：



(1).Job 提交过程

job的提交通过调用submit()方法创建一个JobSubmitter实例，并调submitJobInternal()方法。整个job的运行过程如下：

* 向ResourceManager申请application ID，此ID为该MapReduce的jobId。
* 检查output的路径是否正确，是否已经被创建。
* 计算input的splits。
* 拷贝运行job 需要的jar包、配置文件以及计算input的split 到各个节点。
* 在ResourceManager中调用submitAppliction()方法,执行job。

public void submit()

throws IOException, InterruptedException, ClassNotFoundException {

ensureState(JobState.DEFINE);

setUseNewAPI();

connect();

final JobSubmitter submitter =

getJobSubmitter(cluster.getFileSystem(), cluster.getClient());

status = ugi.doAs(new PrivilegedExceptionAction<JobStatus>() {

public JobStatus run() throws IOException, InterruptedException,

ClassNotFoundException {

return submitter.submitJobInternal(Job.this, cluster);

}

});

state = JobState.RUNNING;

LOG.info("The url to track the job: " + getTrackingURL());

}

(2).Job 初始化过程

* 当resourceManager收到了submitApplication()方法的调用通知后，scheduler开始分配container,随之ResouceManager发送applicationMaster进程，告知每个nodeManager管理器。
* 由applicationMaster决定如何运行tasks,如果job数据量比较小，applicationMaster便选择将tasks运行在一个JVM中。那么如何判别这个job是大是小呢？当一个job的mappers数量小于10个，只有一个reducer或者读取的文件大小要小于一个HDFS block时，（可通过修改配置项mapreduce.job.ubertask.maxmaps

,mapreduce.job.ubertask.maxreduces以及mapreduce.job.ubertask.maxbytes

进行调整)

* 在运行tasks之前，applicationMaster将会调用setupJob()方法，随之创建output的输出路径(这就能够解释，不管你的mapreduce一开始是否报错，输出路径都会创建)

（3）Task 任务分配

* 接下来applicationMaster向ResourceManager请求containers用于执行map与reduce的tasks（step 8),这里map task的优先级要高于reduce task，当所有的map tasks结束后，随之进行sort(这里是shuffle过程后面再说）,最后进行reduce task的开始。(这里有一点，当map tasks执行了百分之5%的时候，将会请求reduce，具体下面再总结)
* 运行tasks的是需要消耗内存与CPU资源的，默认情况下，map和reduce的task资源分配为1024MB与一个核，（可修改运行的最小与最大参数配置,mapreduce.map.memory.mb,mapreduce.reduce.memory.mb,mapreduce.map.cpu.vcores,mapreduce.reduce.reduce.cpu.vcores.)

(4).Task 任务执行

* 这时一个task已经被ResourceManager分配到一个container中，由applicationMaster告知nodemanager启动container，这个task将会被一个主函数为YarnChild的java application运行，但在运行task之前，首先定位task需要的jar包、配置文件以及加载在缓存中的文件。
* YarnChild运行于一个专属的JVM中，所以任何一个map或reduce任务出现问题，都不会影响整个nodemanager的crash或者hang。
* 每个task都可以在相同的JVM task中完成，随之将完成的处理数据写入临时文件中。

4.基于Map和Reduce的并行计算模型

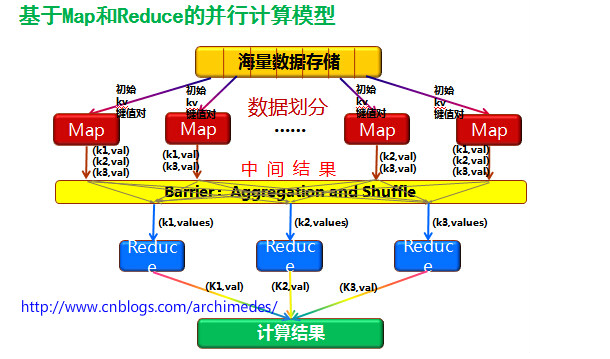


图3-2

5.MapReduce常见算法：（https://www.cnblogs.com/1130136248wlxk/p/4975116.html）

1. 单词计数
2. 数据去重
3. 排序（按某个Key进行升序或降序排列）
4. 筛选（对源数据中所有数据进行排序，取出前K个数据）TOP K
5. 选择
6. 投影
7. 分组
8. 多表连接
9. 单表连接

Zookeeper学习笔记

ZooKeeper是一个[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F/19276232" \t "_blank)的，开放源码的[分布式应用程序](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/9854429)协调服务，是[Google](https://baike.baidu.com/item/Google)的Chubby一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339" \t "_blank)的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：**配置维护、域名服务、分布式同步、组服务等**。

ZooKeeper的基本运转流程：

1、选举Leader。

2、同步数据。

3、选举Leader过程中算法有很多，但要达到的选举标准是一致的。

4、Leader要具有最高的执行ID，类似root权限。

5、集群中大多数的机器得到响应并接受选出的Leader。

参考地址：<https://www.w3cschool.cn/zookeeper/zookeeper_fundamentals.html>

Zookeeper的架构

Zookeeper的工作流（读写数据）

Zookeeper leader选举