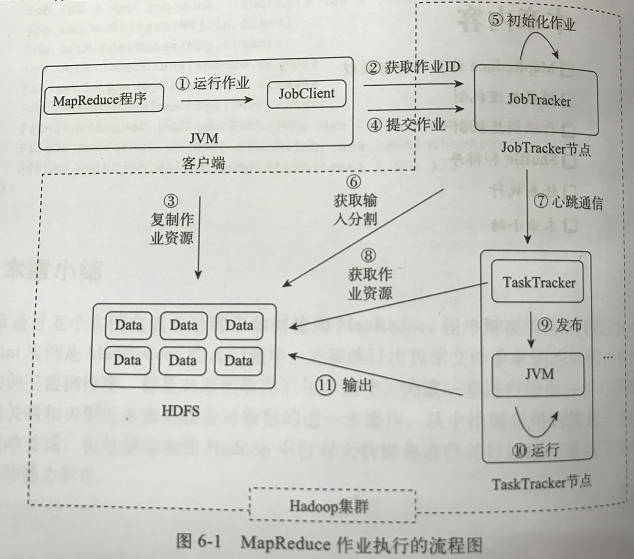
# MapReduce作业的执行流程

一个MapReduce作业的执行流程基本是：代码编写、作业配置、作业提交、map任务分配和执行、处理中间结果、reduce任务分配和执行、作业完成。每个任务执行过程中，包含输入准备、任务执行和输出结果。



client：编写MapReduce代码、配置作业、提交作业；

JobTracker：初始化作业、分配作业、与TaskTracker通信，协调整个作业的执行；

TaskTracker：与JobTracker保持通信，在分配的数据片段上执行Map和Reduce任务。

HDFS：保存作业的数据、配置信息等，保存作业的结果。

客户端提交作业的过程大致如下：

1. 获取一个作业的ID；
2. 检查相关路径，比如输入文件和输出文件是否存在；
3. 对作业的输入进行划分，划分信息写入Job.split文件；
4. 把运行需要的资源，比如作业的jar文件、配置文件、输入的划分等复制到HDFS中；
5. 真正的提交作业。

JobTracker上进行作业的初始化（步骤5），主要是从HDFS中读取Job.split得到划分的信息（步骤6）。然后在JobTracker上对map和reduce的任务进行初始化，这里的初始化是JobTracker上初始化了相关的对象，并没有将任务分发给TaskTracker上。JobTracker和TaskTracker之间的通信和任务的分配是通过心跳机制完成的（步骤7）。JobTracker会通过心跳告诉JobTracker是否存活、是否准备执行新的任务。JobTracker接收到信息后，如果有待分配的任务，它就会给TaskTracker分配一个任务，分配信息封装在心跳信息中。TaskTracker从心跳信息得知如果是一个task的话会把这个task加入本机的任务队列中。

TaskTracker申请到新的任务后的第一步是将任务本地化，即将任务运行所必须的数据、配置信息、程序代码从HDFS复制到TaskTracker的本地（步骤8）：

1. 将job.split复制到本地；
2. 将job.jar复制到本地；
3. 将job的配置信息写入job.xml;
4. 创建本地任务目录，解压job.jar;
5. 本地启动任务（步骤9）。

# 错误处理

在hadoop集群中同一时刻只会有一个JobTracker，预防单点JobTracker出现故障的办法就是创建多个备用JobTracker节点。

如果TaskTracker一定时间内没有与JobTracker通信，那么JobTracker会将此TaskTracker从等待任务调度的TaskTracker集合中移除，同时JobTracker会要求此TaskTracker上的任务立刻返回，如果此TaskTracker任务是在mapping阶段的map任务，那么JobTracker会要求其他的TaskTracker重新执行**所有**原本由故障TaskTracker执行的map任务；如果任务是reduce阶段的reduce任务，那么JobTracker会要求其他TaskTracker重新执行故障TaskTracker未完成的reduce任务。可见map任务出现问题，map所在的TaskTracker上所有的map任务都要重新执行，而reduce任务出现问题，只需要再次执行reduce任务就可以了。比如一个TaskTracker已经完成了被分配的3个reduce任务中的2个，因为reduce任务完成就会把结果写到HDFS上，所以第3个任务出现问题也只需要重新执行就可以了，但是对于map来说，即使完成了部分的map任务，reduce仍然无法获取此节点上所有map的所有输出，所以只能把TaskTracker上所有的map任务重新执行。

还会遇到用户代码缺陷或进程崩溃引起的任务失败。如果是代码缺陷会在执行的过程中抛出异常，任务的JVM进程会自动退出，TaskTracker会将此次任务标记失败；对于进程崩溃，TaskTracker的监听程序会发现进程退出，TaskTracker也会将此次任务标记失败；对于死循环或者执行时间太长的程序，由于TaskTracker没有接收到进度更新，也会将此任务标记失败，并杀死程序对应的进程。

TaskTracker将任务标记失败后会将自身的任务计数减1，以便向JobTracker申请新的任务。TaskTracker会通过心跳告诉JobTracker本地的一个任务失败，JobTracker收到任务失败的通知后重置任务状态，将其加入到调度队列重新分配该任务执行（JobTracker会尝试避免将失败的任务再次分配给之前失败的TaskTracker）。如果任务尝试了一定的次数都是失败，那么就不会在重试了，整个作业也就失败了。

# 作业调度机制

早期的hadoop采用FIFO调度算法，后来FIFO调度器也加入了设置优先级的功能，但是由于不支持优先级抢占，这种调度算法仍然不符合云计算中采用并行计算来提供服务的宗旨。后来提供了支持多用户同时服务和集群资源公平共享的调度器，即公平调度器和容量调度器。

公平调度器：当集群只有一个作业运行时，它将使用整个集群；当有其他用户提交作业时，系统会将TaskTracker节点空闲时间片分配给这些新的作业，并保证每一个作业都得到大概等量的CPU时间。

# Shuffle和排序

为了让reduce能够并行处理map结果，必须对map的输出进行一定的排序和分割，将map输出进一步调整交给reduce的过程就是shuffle。

总体来说，shuffle过程包含在map和reduce两端中。在map端的shuffle过程是对map结果进行分区（partition）、排序（sort）和分割（spill），然后将属于一个划分的输出合并在一起（merge）写在磁盘上，同时按照不同的划分将结果发送给对应的reduce。reduce端会将各个map送来的属于同一个划分的输出进行合并（merge），然后对merge的结构进行排序，最后交给reduce处理。

## Map端

map端的输出内存缓冲区内容达到设定的阈值

## Reduce端