Hive MapReduce 提交流程分析

梁堃 2012-4-25

目录

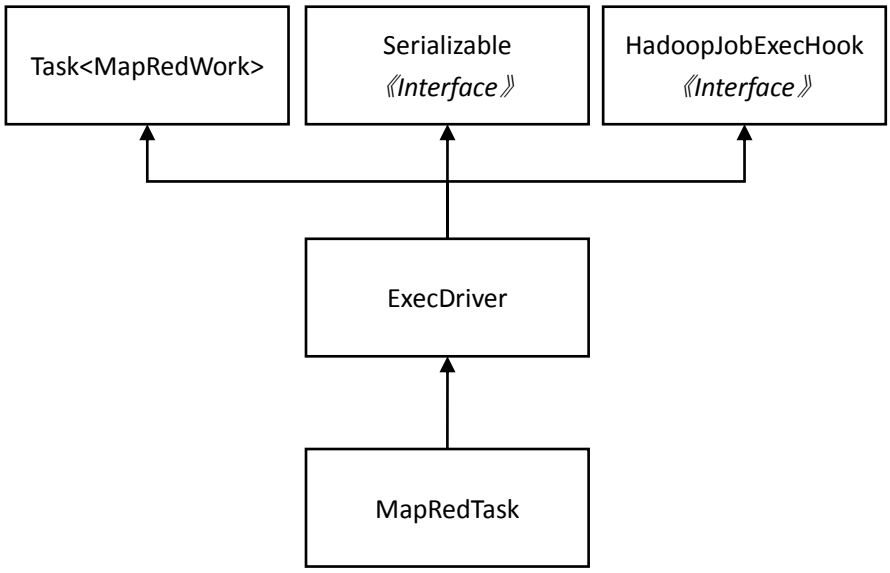
背景

本文专门分析 Hive 向 Hadoop 提交一个 MapReduce 任务的流程及其任务参数。分析基于社区版 Hive 0.8.0 源代码进行。对于 Hive 整体结构和流程的分析，请见《Hive 代码分析.pdf》一文。

在 Hive 中，一个 SQL 会被编译成一组 Task 组成的 DAG。ql.Driver 类会根据 DAG 的依赖关系，依次执行各个 Task。所有的 Task 都是 ql.exec.Task 的子类。每个 Task 的执行主要包含两步：

1. 调用 Task 类的 initialize()方法。
   该方法初始化一些 Task 的内部状态，如记录 queryPlan、设置 isdone=false，获取 Hive 实例等。一般会被子类 override，已完成子类任务一些特定的初始化逻辑。
2. 调用 Task 类的 executeTask()方法
   该方法会调用抽象方法 execute()完成实际的工作，各个实际的 Task 子类需要实现(override)这个 execute()方法。

Hive 编译得到的 MapReduce 任务也是通过 Task 的方式执行的。这里，主要涉及的类包括如下几个。



其中，最重要的类包括：

1. Task：如上所述，任务的基类；
2. MapRedWork：该类用于描述一个 MapReduce 任务的相关信息；
3. ExecDriver/MapRedTask：这两个类用于提交一个 MapReduce 任务。其中，ExecDriver 也用于执行 MapredLocalTask。

任务提交流程

本部分用于分析 MapReduce 任务的提交流程。流程会从直接的 MapRedTask 类开始，中会穿插对 ExecDriver 代码的调用。本文不理会 MapredLocalTask，它用在 MapJoin 中，用于生成 map 使用的 hash 表。在流程中，会以几个表格的形式列出 MapReduce 任务的主要参数。

构造

MapRedTask 构造函数直接调用 ExecDriver 的构造函数，后者初始化了两个对象，一个是保存在 console 成员中的 LogHelper，用于打印执行信息；另一个是保存在 jobExecHelper 成员中的 HadoopJobExecHelper，用于提交任务后打印进度信息等。

initialize()

MapRedTask 没有 initialize 方法，所以，initialize 由 ExecDriver 完成。注意，在 Local 模式下，即执行 MapredLocalTask 时，该方法不会被调用。该方法的主要工作包括：

1. 创建一个 JobConf 实例，保存在 job 成员中。
2. 在 job 中设置属性 "hive.added.files.path" 、 "hive.added.jars.path" 、 "hive.added.archives.path"属性，分别对应于当前 SessionState 中的三类被用户 add 的资源。
3. 根据 job 创建一个新的 HadoopJobExecHelper，放在 jobExecHelper 中。

MapRedTask.execute()

该方法主要提供下面两种功能：

1. 根据情况和配置决定，是否在一个单独的 JVM 中提交任务；
2. 在“最后一分钟”调整 MapReduce 任务参数，包括 Reducer 的个数、是否 Local 执行该 MapReduce 任务。

单独 JVM 执行

在以下条件下会采用 Local 方式执行一个 MapRedTask。

情况一："hive.exec.submitviachild"属性为 true，表示用户设置采用子 JVM 进程提交任务。

情况二： "mapred.job.tracker"属性是"local"，表示本地执行。此时，也需要使用单独的 JVM 进程执行任务。这又包含两种可能：

1. hadoop 客户端本来就设置为 local 执行。
2. "hive.exec.mode.local.auto"优化开关设置为 true，并且，Hive 认为当前的情况满足这个优化条件(总输入小于某个预定义的大小，如 128M；总 Map task 数目不超过某个预定义的大小，如 4；Reduce task 数目不超过 1)，则 Hive 会将 mapred.job.tracker 属性设置为 local。

在使用单独 JVM 执行的情况下，通过执行一个外部的 hadoop jar 命令提交任务，这个 hadoop 命令会新启动一个 JVM、加载 hive 的 jar 包，并调用 ExecDriver 类的 main 方法，将必要的信息通过命令行参数传递给这个 main。后续执行的流程与非单独 JVM 的执行流程类似。

非单独 JVM 执行

如果不满足上面一节的条件，则该 MapReduce 任务是真的需要提交到某个 Hadoop 机群执行的，并且，提交是与 Hive 在同一个进程中进行的。这种情况下，MapRedTask 的 execute() 主要做两件事：

1. 在 MapRedWork 实例中设置 reducer 的数目信息；
2. 调用基类 ExecDriver 的 execute 方法完成任务的提交。

Reducer 数据的设置方法如下：

1. 若 MapRedWork 不包含 reducer，那么这是个仅包含 Mapper 的任务，将 reducer 的数目设置为 0；
2. 否则：
   a) 如果在 work 实例中已经设置了 reducer 的数目，无需修改，这是编译器设置的；
   b) 如果 work 中没有设置，job 中设置了 reducer 的数目，以 job 中的数目设置 work 中 reducer 的数目；
   c) 如果 work 和 job 中都没有设置，根据如下公式设置：(totalInputBytes + bytesPerReducer -1) / bytesPerReducer，bytesPerReducer 来自配置文件，并且保证 reducer 至少是 1，最多不超过配置文件中指定的最大 reducer 数目。

ExecDriver.execute()

该方法主要分为三大部分(前两部分的代码相互交杂)：

1. 执行任务需要的准备代码，如临时目录的创建等；
2. 设置 MapReduce 任务的参数；
3. 提交任务并打印进度信息，任务结束后打印结果信息，清理，返回状态码。

准备工作

MapReduce 任务的准备工作主要包括以下这些。

1. 调用 work 的 isInvalid()方法，验证当前 work 是一个正确的 work。
   其实该方法仅仅是检查了 reducer 的个数。
2. 创建 MR 临时目录。
   如果是 local 执行模式，该目录是 "java.io.tmpdir"/"user.name"/"executionId"/-mr-"pathId"；否则该目录是 hive 配置中 "hive.exec.scratchdir 指定的路径"/"executionId"/-mr-"pathId"，其中的 pathId 是单调增加的，以保证在一个任务范围内，每个临时目录均不同。这个临时目录路径会被转成全限定的 URI。

3. 如果该 MapRedWork 存在对应的 MapredLocalWork，并且，该 MapRedWork 的执行模式不是 local 模式：
   此时，localwork 中必然存在一个 tmpFileURI，它是一个 local 的临时目录；同时 work 中必然存在一个 tmpHDFSFileURI，它是一个 MR 临时目录。临时目录的创建与 2 中的临时目录获取方式类似。这两个是由 optimizer/physical/MapJoinResolver 类设置的。
   将 localwork 的 tmpFileURI 中的文件(MapJoin 使用的 hashtables)打成一个.tar.gz 包，并上传至 HDFS 的临时目录(设置其副本数为 10，以便高并发访问)。并将这个包加入 DistributedCache 中。
4. 将 MapRedWork 序列化到 MR 临时目录中，并且也将它加入 DistributedCache，设置副本数是 10。
   将这个路径设置在 job 中，属性名为"hive.exec.plan"。并在 Utilities 类中维护 jobID 到 work 的对应关系。
5. 为每个 FileSinkOperator 创建对应的临时目录。
   从 Operator 中取得 FileSinkDesc 描述，并从后者获取 dirName。这个 dirName 是 optimizer/GenMRFileSink1 设置的。为 dirName 生成一个 tempDir，并创建这个 tempDir。这里，临时目录的路径就是 hive.exec.scratchdir 指定的路径，但是它的 scheme 和 authority 则是与 FileSinkDesc 中 dest 的相同的。即，临时路径是创建在目标机群上的。

MapReduce 任务参数

MapReduce 任务的参数如下表所示。

参数	值
OutputFormat	NullOutputFormat
OutputCommitter	NullOutputCommitter
mapred.committer.job.setup.cleanup.needed	false
MapperClass	ExecMapper
MapOutputKeyClass	HiveKey
MapOutputValueClass	BytesWritable
PartitionerClass	配置" hive.mapred.partitioner"; 默认" ql.io.DefaultHivePartitioner"
numMapTasks、MaxSplitSize、MinSplitSize、MinSplitSizePerNode、MinSplitSizePerRack、numReduceTasks	根据 work 中的相应值设置，如果没有，则保持 job 默认状态。
ReducerClass	ExecReducer
hive.input.format、hive.index.compact.file、hive.index.blockfilter.file、hive. input. format. sorted	根据 work 设置
hive.exec.plan	MR 临时目录中以 JobID 为名的文件
mapred.reduce.tasks.speculative.execution	根据 hive. mapred. reduce. tasks. speculative. execution 设置，默认为 true
InputFormat	根据 hive.input.format 设置，默认为

	ql.io.CombineHiveInputFormat
tmpjars、tmpfiles、tmparchives	根据 job 中 hive.added.{jars,files,archives}.path 的设置，其中，tmpjars 还包含 hive.aux.jars.path 属性中的路径。
FileInputFormat.setInputPaths	将 work 中每个输入路径都加入 inputPaths 中。如果一个 alias 没有路径与之对应，或者与之对应的路径是空的，则使用一个空文件代替。空文件所在的目录是 MR 临时目录，机群就是默认的 hadoop 机群。（注意：如果这个空路径对应的是 Non-Native 的表，则不需要创建临时目录和空文件，而仅仅是将这个路径名加入到 inputPaths 中就可以了）

任务提交与结果清理

该部分较为简单。提交任务后，使用 jobExecHelper.progress()方法不断打印进度信息。任务执行完毕后，在一个 finally{}里面将准备工作中创建的临时目录、文件等清理掉；同时，将 Utilities 的内部状态更新。返回运行的返回值。