获取原生mapreduce程序、sqoop、hive提交的jobId

需求

项目需要实现一个原生mapreduce程序、sqoop、hive和pig脚本的代理命令行，类似于azkaban的使用方式，用户可以提交编译好的mapreduce程序（可以是一个jar包，在配置中执行入口class名）、hive命令或者脚本、sqoop命令和pig脚本。由于这些命令的执行过程中可能会生成mapreduce任务并提交到hadoop集群上执行，为了方便用户查看每一个用户提交的任务的执行状态，我们需要获得用户提交任务生成的mapreduce任务的jobId，进而可以通过hadoop提供的API查看job的状态，首先需要面临的两个问题：如何判断哪些用户提交的任务会生成mapreduce任务；如何获取用户提交任务生成的一个或者多个mapreduce任务的jobId。

实现

当前使用的版本是：hadoop2.2.0官方版，hive 0.13.1和sqoop 1.4.5

mapreduce日志输出

一般情况下在命令行提交mapreduce程序的时候，会等待mapreduce程序执行直到job执行完成，在mapreduce client中默认的日志级别是INFO，以Word Count为例：

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

首先需要创建一个Job对象(org.apache.hadoop.mapreduce.Job)，然后调用接口设置各种配置，job实现了JobContext接口，后者继承自接口MRJobConfig，在MRJobConfig中可以看到所有的job的配置项，Job的状态有两个DEFINE和RUNNING（JobState），在job配置完成之后调用waitForCompletion可以同步得提交并等待job执行完成，函数定义如下：

```
1282  public boolean waitForCompletion(boolean verbose
1283                                   ) throws IOException, InterruptedException,
1284                                            ClassNotFoundException {
1285    if (state == JobState.DEFINE) {
1286      submit();                                         //提交任务，如果重复提交不会出错
1287    }
1288    if (verbose) {
1289      monitorAndPrintJob();                  //如果参数为true则打印job执行的日志输出
1290    } else {
1291      // get the completion poll interval from the client.
1292      int completionPollIntervalMillis =
1293        Job.getCompletionPollInterval(cluster.getConf());
1294      while (!isComplete()) {
1295        try {
1296          Thread.sleep(completionPollIntervalMillis);
1297        } catch (InterruptedException ie) {
1298        }
1299      }
1300    }
```

```
1301    return isSuccessful();
1302  }
```

该函数首先提交处于DEFINE状态的job，在submit函数中会创建一个JobSubmitter对象，然后调用该对象的submitJobInternal函数向hadoop cluster提交该job，完成之后设置Job的状态为RUNNING，在submitJobInternal函数中会调用JobID jobId = submitClient.getNewJobID();函数，从hadoop集群中获取jobID，并将它保存在job中，在真正向hadoop集群提交这个job之前会调用printTokens函数，该函数实现如下：

```
475  private void printTokens(JobID jobId,
476      Credentials credentials) throws IOException {
477    LOG.info("Submitting tokens for job: " + jobId);
478    for (Token<?> token: credentials.getAllTokens()) {
479      LOG.info(token);
480    }
481  }
```

所以如果保持在客户端保持默认的日志级别（INFO），我们可以通过在提交job之后监控客户端的日志输出，匹配“Submitting tokens for job:”字符串获取已提交的jobId（在2.2.0以后的版本中都可行），在打印这行日志之后才会调用submitClient.submitJob函数提交该jobId。

回到waitForCompletion函数，在提交完成之后会根据是否打开verbose开关决定是否输出该job的执行状态，如果设置为true则在monitorAndPrintJob函数中会周期性的获取map和reduce完成的进度，直到job执行完成，如果设置为false则周期性的查看job是否完成，并不输出任何执行进度信息。

从job提交过程可以看到，如果用户调用waitForCompletion函数提交job，则不论参数什么都可以通过匹配“mapreduce.JobSubmitter: Submitting tokens for job:”字符串来获取jobID信息，不过这也有一个前提是需要保证log4j的mapreduce.JobSubmitter类的日志级别配置为INFO或者以下，否则该日志不会输出。

类似于azkaban执行命令行程序的方式，可以使用ProcessBuilder启动子进程执行程序，再将进程的标准输出和标准错误输出重定向到两个线程中，这两个线程阻塞得读取进程的标准输出和标准错误输出。

原生mapreduce程序另外一种方案

在上面的mapreduce的job提交过程分析中可以看到，用户的mapreduce程序的上下文保存在job对象中，我们可以通过定义一个接口，该接口定义getJob函数，该函数返回用户配置完成但是尚未提交的job对象，可以强制性的约束用户必须实现该接口，这样我们就可以获取用户需要提交到hadoop的job，由我们的执行服务器负责提交该job，这样我们可以在提交之后（通过调用job的submit方法，该方法不会等待job执行完成）获得提交任务的jobID。

但是该方案需要用户实现我们定义的接口，增加了用户的学习成本，并且对于之前hadoop平台上已经提交的mapreduce程序需要作出一定的修改。

sqoop命令

我们当前使用的sqoop是1.4.5版本，对应的是sqoop1，它通过命令行的方式执行的，无论是在linux命令行中提交sqoop命令还是在azkaban中调用sqoop程序，都是通过调用org.apache.sqoop.Sqoop类的进程函数runTool(String [] args, Configuration conf)完成的，在该函数执行import和export命令实现时，以export为例，它会调用org.apache.sqoop.mapreduce.ExportJobBase的runExport函数，该函数会在判断一些错误情况后和初始化之后创建一个mapreduce的Job对象，然后配置该对象，然后调用runJob(job)方法提交该job，提交的方式如下：

```
300  protected boolean doSubmitJob(Job job)
301    throws IOException, InterruptedException, ClassNotFoundException {
302    return job.waitForCompletion(true);
303  }
```

这和原生的mapreduce程序的提交方式是相同的，所以我们也可以通过抓取日志的方式来获取sqoop提交的jobId，对于export和import命令，每一个命令只会向hadoop提交一个job。

顺便说一句，sqoop2使用的是客户端/服务器的方式进行交互的，可以通过SqoopClient提交任务，提交完成之后会返回一个MSubmission对象，可以通过调用该对象的submission.getExternalId()函数获取hadoop 的jobid，参见http://sqoop.apache.org/docs/1.99.2/ClientAPI.html

hive命令和脚本

在使用hive的过程中，既可以通过hive命令行的方式直接和hadoop进行交互（hive命令只不过是对hadoop程序的一个封装），也可以通过引用hive的jdbc驱动来实现客户端/服务器的方式提交，这种方式类似于mysql的命令执行方式，在使用这种方式hiveServer直接和hadoop交互。

hive中可以通过设置hooks来获取每一个命令的执行，可以设置三种hooks，分别在每一条命令执行之前、执行之后和执行失败的情况下被调用，

可以在配置文件中设置这些hook的类名，可以设置多个hook，每一个hook之间通过逗号分隔，这些Hook的都需要实现org.apache.hadoop.hive.ql.hooks.ExecuteWithHookContext接口并实现run函数，该函数的接口如下：

```
public interface ExecuteWithHookContext extends Hook {
  void run(HookContext hookContext) throws Exception;
}
```

在每一条hive命令的执行过程中在执行前和执行后会回调配置的hook的run函数，该函数的参数是当前执行的命令的上下文，包括该命令的配置信息（hookContext.getConf()）、执行计划（hookContext.getQueryPlan()）、hook类型（HookContext.getHookType()，包括PRE*EXEC*HOOK, POST*EXECHOOK*, *ONFAILURE*HOOK）等，在执行计划中包含本次执行的命令（queryString）和hive赋予的id（queryId）。

但是我们不能通过hook获得提交到mapreduce的job信息，但是在hive中存在一种结构化日志，这个日志的格式如下：

```
QueryStart QUERY_STRING="select count(age), max(age), min(age) from person group by name" QUERY_ID="hzfengyu_20150227114848_8ea098
 //QueryStart 日志在每一个query开始执行的时候输出，包含输入的命令字符串和hive的queryId。
TaskStart TASK_NAME="org.apache.hadoop.hive.ql.exec.mr.MapRedTask" TASK_ID="Stage-1" QUERY_ID="hzfengyu_20150227114848_8ea0982a-e0
 //TaskStart 在每当hive创建并向hadoop提交一个job的时候输出，它包括该mapreduce的job的jobName，本次query的queryId等信息。
TaskProgress TASK_HADOOP_PROGRESS="2015-02-27 11:48:44,494 Stage-1 map = 0%,  reduce = 0%" TASK_NUM_REDUCERS="1" TASK_NAME="org.ap
 //TaskProgress 是在hive向hadoop提交job之后等待这些job执行完成的过程中，周期性的向hadoop获取每一个提交的job的执行状态，该日志包括该job的执行状态，
TaskEnd TASK_RET_CODE="0" TASK_HADOOP_PROGRESS="2015-02-27 14:56:33,614 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 295.99
 //TaskEnd 在每一个task执行完成的时候输出，包括task执行的结果、taskName、queryId和jobId等信息。
QueryEnd QUERY_STRING="select count(age), max(age), min(age) from person group by name" QUERY_ID="hzfengyu_20150227144646_bd23133d
 //QueryEnd 在整个命令执行完成的时候输出，包括命令字符串、queryId、执行结果和执行过程中mapreduce的job数目。
```

以上日志信息中，query对应的是每一条hive的sql语句的执行，每一个task对应的是hive内部执行解析完成的SQL语句的Task信息，不同的SQL语句对应着不同的Task，在hive中一共有10多种具体的Task来处理不同的SQL，例如对于create table语句则有org.apache.hadoop.hive.ql.exec.DDLTask处理，需要提交到hadoop集群的Task是由org.apache.hadoop.hive.ql.exec.mr.ExecDriver实现的，而TaskProgress信息正式在该类处理Task的过程中输出的，在Task的处理过程中调用JobClient.submitJob(job)来提交hadoop任务，每一个Task最多提交一个Job，但是一个hive的query语句可以产生多个Task，每一个Task提交之后该线程会周期性的从hadoop集群中拉取当前任务的执行状态，然后根据配置的频率定期的输出到格式化日志。

hive中有session的概念，这个session指的是hive与hadoop之间的会话，所以每一次启动hive命令的时候都是重新启动一个新的session，而hiveServer的运行过程中始终保持同一个session，重启之后重新创建一个session，上面介绍的格式化日志是每一个session生成的，每一个session中所有命令的格式化日志都输出到一个日志文件中，文件名是以sessionId+随机数的方式命名的（histFileName = conf*file*loc + File.separator + "hive*joblog*" + *ss.getSessionId()* + ""+ Math.abs(randGen.nextInt()) + ".txt";），我们可以通过匹配sessionId来获取当前session的格式化日志文件，在hive中，session是和一个线程关联的，每一个sessionState对象保存在ThreadLocal对象中。如果我们希望所有的query的结构化日志都输出到一个文件中，我们就需要使用一个线程来执行所有的命令，否则需要对每一个线程监听不同的日志文件的输出。

有了以上的信息，就可以获取每一个hive提交到hadoop的jobId了，具体方案如下：在执行命令之前，为当前线程创建一个hive的Session，代码如下（azkaban的代码）：

```
final HiveConf hiveConf = new HiveConf(SessionState.class);
//初始化hive的配置信息
ss = new CliSessionState(hiveConf);
SessionState.start(ss);     //在这里设置ss与当前线程的关联

logger.info("SessionState = " + ss);
ss.out = System.out;
ss.err = System.err;
ss.in = System.in;
```

得到session之后，可以通过ss.getHiveHistory().getHistFileName()获取当前session格式化日志文件名，然后我们监听该文件的输出（类似于tail），根据query和task中输出的queryId信息关联每query和task，然后通过QueryStart信息中QUERY_STRING的信息关联用户输入的命令和hive的QueryId，通过格式化日志我们可以获取每一个hive命令的mapreduce的jobId信息，如果只需要这些信息，hive的hook就不需要设置了，但是通过hook可以获取hive内部执行的每一个query的HookContext 对象，并且基于此进行编程，而格式化日志只能够获得hive输出的固定的日志信息。

在hive中，格式化日志是由一些配置项控制的，相关的配置项如下：

- hive.querylog.location：格式化日志存放的目录，日志文件根据sessionId动态创建的，默认的存放路径为/tmp/${user.name}下
- hive.querylog.enable.plan.progress：是否开启记录执行计划的执行日志，包括TaskProgress信息，在处理SQL语句解析出的Task执行过程中输出，默认为true。

- hive.session.history.enabled：是否开启格式化日志的输出，这是一个总的开关，如果不打开将不会输出任何格式化日志信息，默认为false。
- hive.exec.counters.pull.interval：向hadoop集群中拉取Task执行状态的周期，单位为ms，默认为1000ms
- hive.querylog.plan.progress.interval：Task执行过程中输出格式化日志的频率，单位为ms，默认为6000ms，注意，实现过程中，线程使用以上一个时间作为sleep的参数，在每次wakeup之后判断当前时间与上次输出时间的差值之差是否大于该配置来决定是否输出该日志，所以一般情况下该配置需要配置为上一个配置项的整数倍，否则会有一定的延迟。
- hive.exec.pre.hooks：在每一个Query执行之前回调的接口，需要继承ExecuteWithHookContext 接口
- hive.exec.post.hooks：在每一个Query执行之后回调的接口，需要继承ExecuteWithHookContext 接口
- hive.exec.failure.hooks：在每一个Query执行失败时回调的接口，需要继承ExecuteWithHookContext 接口

如果设置hooks，需要将hook的文件编译完成打包成jar包放入到hive的classpath下面，可以放到hive目录的lib目录下。

除此之外，如果直接调用hive命令行程序，其实就是对hadoop jar命令的封装，所以它还是依赖于hadoop的配置，在hadoop的配置中可以设置任务提交过程中是根据mapreduce.framework.name配置项来决定提交到集群还是本地提交，默认的配置是local，意味着提交到本地，那么产生的jobId包含local字段，在hive中此类Task是由org.apache.hadoop.hive.ql.exec.mr.MapredLocalTask实现，在这个Task的执行过程中不会输出TaskProgress信息，因此需要将mapreduce.framework.name配置成yarn（hadoop 2.x环境）。

实际的使用过程中，在开启hive格式化日志的前提下，每一个hive的session一个线程，该线程通过tail的方式扫描该session的格式化日志文件，然后通过QueryStart日志中输出的queryString和QueryId进行关联，然后再通过TaskProgress的输出关联QueryId和提交到hadoop的jobId，但是这个过程可能有一定的延迟，由hive.querylog.plan.progress.interval来决定延迟的时间。