Hive中Reduce个数是如何计算的

我们在使用Hive查询数据的时候经常会看到如下的输出：

```
Query ID = iteblog_20160704104520_988f81d4-0b82-4778-af98-43cc1950d357
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
```

很明显，上面的输出信息展示了Hive同设置Reduce的三种方法，本文将在源码的角度解释这三个参数的作用。在进入代码介绍之前让我们先来看看这三个参数的含义。

hive.exec.reducers.bytes.per.reducer

此参数从Hive 0.2.0开始引入。在Hive 0.14.0版本之前默认值是1G(1,000,000,000)；而从Hive 0.14.0开始，默认值变成了256M(256,000,000)，可以参见HIVE-7158和HIVE-7917。这个参数的含义是每个Reduce处理的字节数。比如输入文件的大小是1GB，那么会启动4个Reduce来处理数据。

hive.exec.reducers.max

此参数从Hive 0.2.0开始引入。在Hive 0.14.0版本之前默认值是999；而从Hive 0.14.0开始，默认值变成了1009；可以参见HIVE-7158和HIVE-7917。这个参数的含义是最多启动的Reduce个数。比如input size/hive.exec.reducers.bytes.per.reducer>hive.exec.reducers.max，那么Hive启动的Reduce个数为hive.exec.reducers.max；反之为input size/hive.exec.reducers.bytes.per.reducer。这个参数只有在mapred.reduce.tasks/mapreduce.job.reduces设置为负数的时候才有效。

mapred.reduce.tasks/mapreduce.job.reduces

此参数从Hive 0.1.0开始引入。默认值是-1。此参数的含义是Reduce的个数，典型的情况是设置成接近可用节点的质数。如果mapred.job.tracker的值是local此参数将会被忽略。在Hadoop中此参数的默认值是1；而在Hive中默认值是-1。通过将此参数设置为-1，Hive将自动计算出应该启动多少个Reduce。

源码分析

我们已经将三个参数的含义介绍完，现在我们来看看Hive代码中是如何计算Reduce个数的。首先来看看Hive是如何从输入的文件大小计算Reduce个数：

```
/**
 * Estimate the number of reducers needed for this job, based on job input,
 * and configuration parameters.
 *
 * The output of this method should only be used if the output of this
 * MapRedTask is not being used to populate a bucketed table and the user
 * has not specified the number of reducers to use.
 *
 * @return the number of reducers.
 */
public static int estimateNumberOfReducers(HiveConf conf, ContentSummary inputSummary,
                                           MapWork work, boolean finalMapRed) throws IOException {
  long bytesPerReducer = conf.getLongVar(HiveConf.ConfVars.BYTESPERREDUCER);
  int maxReducers = conf.getIntVar(HiveConf.ConfVars.MAXREDUCERS);

  double samplePercentage = getHighestSamplePercentage(work);
  long totalInputFileSize = getTotalInputFileSize(inputSummary, work, samplePercentage);

  // if all inputs are sampled, we should shrink the size of reducers accordingly.
  if (totalInputFileSize != inputSummary.getLength()) {
    LOG.info("BytesPerReducer=" + bytesPerReducer + " maxReducers="
        + maxReducers + " estimated totalInputFileSize=" + totalInputFileSize);
  } else {
    LOG.info("BytesPerReducer=" + bytesPerReducer + " maxReducers="
      + maxReducers + " totalInputFileSize=" + totalInputFileSize);
  }

  // If this map reduce job writes final data to a table and bucketing is being inferred,
  // and the user has configured Hive to do this, make sure the number of reducers is a
  // power of two
  boolean powersOfTwo = conf.getBoolVar(HiveConf.ConfVars.HIVE_INFER_BUCKET_SORT_NUM
_BUCKETS_POWER_TWO) &&
      finalMapRed && !work.getBucketedColsByDirectory().isEmpty();

  return estimateReducers(totalInputFileSize, bytesPerReducer, maxReducers, powersOfTwo);
}

public static int estimateReducers(long totalInputFileSize, long bytesPerReducer,
    int maxReducers, boolean powersOfTwo) {
  double bytes = Math.max(totalInputFileSize, bytesPerReducer);
```

```
  int reducers = (int) Math.ceil(bytes / bytesPerReducer);
  reducers = Math.max(1, reducers);
  reducers = Math.min(maxReducers, reducers);

  int reducersLog = (int)(Math.log(reducers) / Math.log(2)) + 1;
  int reducersPowerTwo = (int)Math.pow(2, reducersLog);

  if (powersOfTwo) {
    // If the original number of reducers was a power of two, use that
    if (reducersPowerTwo / 2 == reducers) {
      // nothing to do
    } else if (reducersPowerTwo > maxReducers) {
      // If the next power of two greater than the original number of reducers is greater
      // than the max number of reducers, use the preceding power of two, which is strictly
      // less than the original number of reducers and hence the max
      reducers = reducersPowerTwo / 2;
    } else {
      // Otherwise use the smallest power of two greater than the original number of reducers
      reducers = reducersPowerTwo;
    }
  }
  return reducers;
}
```

totalInputFileSize的值就是输入文件的总大小，然后通过estimateReducers函数估算Reduce个数；其中bytesPerReducer和maxReducers分别是hive.exec.reducers.bytes.per.reducer和hive.exec.reducers.max的值，powersOfTwo参数如果在使用bucket并且hive.exec.infer.bucket.sort.num.buckets.power.two设置成true才会起作用，此参数默认值为false。所以Reduce的计算公式主要就是下面四行代码

```
double bytes = Math.max(totalInputFileSize, bytesPerReducer);
int reducers = (int) Math.ceil(bytes / bytesPerReducer);
reducers = Math.max(1, reducers);
reducers = Math.min(maxReducers, reducers);
```

也就是Reduce最少为1，最大为maxReducers（就是hive.exec.reducers.max的值）。如果powersOfTwo为true，那么Reduce的值将会取小于或大于并且最接近Reduce的值，而且此值是2的指数。

上面代码仅仅是根据输入计算Reduce的个数，最终的设置是通过setNumberOfReducers函数决

定的，如下：

```
/**
 * Set the number of reducers for the mapred work.
 */
private void setNumberOfReducers() throws IOException {
  ReduceWork rWork = work.getReduceWork();
  // this is a temporary hack to fix things that are not fixed in the compiler
  Integer numReducersFromWork = rWork == null ? 0 : rWork.getNumReduceTasks();

  if (rWork == null) {
    console
        .printInfo("Number of reduce tasks is set to 0 since there's no reduce operator");
  } else {
    if (numReducersFromWork >= 0) {
      console.printInfo("Number of reduce tasks determined at compile time: "
          + rWork.getNumReduceTasks());
    } else if (job.getNumReduceTasks() > 0) {
      int reducers = job.getNumReduceTasks();
      rWork.setNumReduceTasks(reducers);
      console.printInfo("Number of reduce tasks not specified. Defaulting to jobconf value of: "
          + reducers);
    } else {
      if (inputSummary == null) {
        inputSummary =  Utilities.getInputSummary(driverContext.getCtx(), work.getMapWork(),
null);
      }
      int reducers = Utilities.estimateNumberOfReducers(conf, inputSummary, work.getMapWor
k(),
                                                        work.isFinalMapRed());
      rWork.setNumReduceTasks(reducers);
      console
          .printInfo("Number of reduce tasks not specified. Estimated from input data size: "
          + reducers);

    }
    console.printInfo("In order to change the average load for a reducer (in bytes):");
    console.printInfo("  set " + HiveConf.ConfVars.BYTESPERREDUCER.varname
        + "=<number>");
    console.printInfo("In order to limit the maximum number of reducers:");
    console.printInfo("  set " + HiveConf.ConfVars.MAXREDUCERS.varname
        + "=<number>");
    console.printInfo("In order to set a constant number of reducers:");
    console.printInfo("  set " + HiveConf.ConfVars.HADOOPNUMREDUCERS
        + "=<numbe>");
  }
```

```
}
```

如果job.getNumReduceTasks()>0，也就是mapred.reduce.tasks/mapreduce.job.reduces参数的值大于0，此时直接取mapred.reduce.tasks/mapreduce.job.reduces的值作为Reduce的个数；否则将会使用上面estimateNumberOfReducers函数估算Reduce个数，最后都是通过rWork.setNumReduceTasks(reducers)设置Reduce的个数。

总结

1、Reduce的个数对整个作业的运行性能有很大影响。如果Reduce设置的过大，那么将会产生很多小文件，对NameNode会产生一定的影响，而且整个作业的运行时间未必会减少；如果Reduce设置的过小，那么单个Reduce处理的数据将会加大，很可能会引起OOM异常。

2、如果设置了mapred.reduce.tasks/mapreduce.job.reduces参数，那么Hive会直接使用它的值作为Reduce的个数；

3、如果mapred.reduce.tasks/mapreduce.job.reduces的值没有设置（也就是-1），那么Hive会根据输入文件的大小估算出Reduce的个数。根据输入文件估算Reduce的个数可能未必很准确，因为Reduce的输入是Map的输出，而Map的输出可能会比输入要小，所以最准确的数根据Map的输出估算Reduce的个数。