六 Hadoop 企业优化

6.1 MapReduce 跑的慢的原因

Mapreduce 程序效率的瓶颈在于两点：

1）计算机性能

CPU、内存、磁盘健康、网络

2）I/O 操作优化

（1）数据倾斜

（2）map 和 reduce 数设置不合理

（3）map 运行时间太长，导致 reduce 等待过久

（4）小文件过多

（5）大量的不可分块的超大文件

（6）spill 次数过多

（7）merge 次数过多等。

6.2 MapReduce 优化方法

MapReduce 优化方法主要从六个方面考虑：数据输入、Map 阶段、Reduce 阶段、IO 传输、数据倾斜问题和常用的调优参数。

6.2.1 数据输入

（1）合并小文件：在执行 mr 任务前将小文件进行合并，大量的小文件会产生大量的 map 任务，增大 map 任务装载次数，而任务的装载比较耗时，从而导致 mr 运行较慢。

（2）采用 CombineTextInputFormat 来作为输入，解决输入端大量小文件场景。

6.2.2 Map 阶段

1）减少溢写（spill）次数：通过调整 io.sort.mb 及 sort.spill.percent 参数值，增大触发 spill 的内存上限，减少 spill 次数，从而减少磁盘 IO。

2）减少合并（merge）次数：通过调整 io.sort.factor 参数，增大 merge 的文件数目，减少 merge 的次数，从而缩短 mr 处理时间。

3）在 map 之后，**不影响业务逻辑前提下，先进行 combine 处理**，减少 I/O。

6.2.3 Reduce 阶段

1）合理设置 map 和 reduce 数：两个都不能设置太少，也不能设置太多。太少，会导致 task 等待，延长处理时间；太多，会导致 map、reduce 任务间竞争资源，造成处理超时等错误。

2）设置 map、reduce 共存：调整 slowstart.completedmaps 参数，使 map 运行到一定程度后，reduce 也开始运行，减少 reduce 的等待时间。

3）规避使用 reduce：因为 reduce 在用于连接数据集的时候将会产生大量的网络消耗。

4）合理设置 reduce 端的 buffer：默认情况下，数据达到一个阈值的时候，buffer 中的数据就会写入磁盘，然后 reduce 会从磁盘中获得所有的数据。也就是说，buffer 和 reduce 是没有直接关联的，中间多个一个写磁盘->读磁盘的过程，既然有这个弊端，那么就可以通过参数来配置，使得 buffer 中的一部分数据可以直接输送到 reduce，从而减少 IO 开销：mapred.job.reduce.input.buffer.percent，默认为 0.0。当值大于 0 的时候，会保留指定比例的内存读 buffer 中的数据直接拿给 reduce 使用。这样一来，设置 buffer 需要内存，读取数据需要内存，reduce 计算也要内存，所以要根据作业的运行情况进行调整。

6.2.4 I/O 传输

1）采用数据压缩的方式，减少网络 IO 的的时间。安装 Snappy 和 LZO 压缩编码器。

2）使用 SequenceFile 二进制文件。

6.2.5 数据倾斜问题

1）数据倾斜现象

数据频率倾斜——某一个区域的数据量要远远大于其他区域。

数据大小倾斜——部分记录的大小远远大于平均值。

2）如何收集倾斜数据

在 reduce 方法中加入记录 map 输出键的详细情况的功能。

```
public static final String MAX_VALUES = "skew.maxvalues";
private int maxValueThreshold;

@Override
public void configure(JobConf job) {
    maxValueThreshold = job.getInt(MAX_VALUES, 100);
}
@Override
public void reduce(Text key, Iterator<Text> values,
```

```
                OutputCollector<Text, Text> output,
                Reporter reporter) throws IOException {
    int i = 0;
    while (values.hasNext()) {
      values.next();
      i++;
    }

    if (++i > maxValueThreshold) {
      log.info("Received " + i + " values for key " + key);
    }
}
```

3）减少数据倾斜的方法

方法 1：抽样和范围分区

可以通过对原始数据进行抽样得到的结果集来预设分区边界值。

方法 2：自定义分区

基于输出键的背景知识进行自定义分区。例如，如果 map 输出键的单词来源于一本书。且其中某几个专业词汇较多。那么就可以自定义分区将这这些专业词汇发送给固定的一部分 reduce 实例。而将其他的都发送给剩余的 reduce 实例。

方法 3：Combine

使用 Combine 可以大量地减小数据倾斜。在可能的情况下，combine 的目的就是聚合并精简数据。

方法 4：采用 Map Join，尽量避免 Reduce Join。

6.2.6 常用的调优参数

1）资源相关参数

（1）以下参数是在用户自己的 mr 应用程序中配置就可以生效（mapred-default.xml）

配置参数	参数说明
mapreduce.map.memory.mb	一个 Map Task 可使用的资源上限（单位:MB），默认为 1024。如果 Map Task 实际使用的资源量超过该值，则会被强制杀死。
mapreduce.reduce.memory.mb	一个 Reduce Task 可使用的资源上限（单位:MB），默认为 1024。如果 Reduce Task 实际使用的资源量超过该值，则会被强制杀死。
mapreduce.map.cpu.vcores	每个 Map task 可使用的最多 cpu core 数目，默认值: 1

mapreduce.reduce.cpu.vcores	每个 Reduce task 可使用的最多 cpu core 数目，默认值: 1
mapreduce.reduce.shuffle.parallelcopies	每个 reduce 去 map 中拿数据的并行数。默认值是 5
mapreduce.reduce.shuffle.merge.percent	buffer 中的数据达到多少比例开始写入磁盘。默认值 0.66
mapreduce.reduce.shuffle.input.buffer.percent	buffer 大小占 reduce 可用内存的比例。默认值 0.7
mapreduce.reduce.input.buffer.percent	指定多少比例的内存用来存放 buffer 中的数据，默认值是 0.0

（2）应该在 yarn 启动之前就配置在服务器的配置文件中才能生效（yarn-default.xml）

配置参数	参数说明
yarn.scheduler.minimum-allocation-mb 1024	给应用程序 container 分配的最小内存
yarn.scheduler.maximum-allocation-mb 8192	给应用程序 container 分配的最大内存
yarn.scheduler.minimum-allocation-vcores 1	每个 container 申请的最小 CPU 核数
yarn.scheduler.maximum-allocation-vcores 32	每个 container 申请的最大 CPU 核数
yarn.nodemanager.resource.memory-mb 8192	给 containers 分配的最大物理内存

（3）shuffle 性能优化的关键参数，应在 yarn 启动之前就配置好（mapred-default.xml）

配置参数	参数说明
mapreduce.task.io.sort.mb 100	shuffle 的环形缓冲区大小，默认 100m
mapreduce.map.sort.spill.percent 0.8	环形缓冲区溢出的阈值，默认 80%

2）容错相关参数(mapreduce 性能优化)

配置参数	参数说明
mapreduce.map.maxattempts	每个 Map Task 最大重试次数，一旦重试参数超过该值，则认为 Map Task 运行失败，默认值：4。
mapreduce.reduce.maxattempts	每个 Reduce Task 最大重试次数，一旦重试参数超过该值，则认为 Map Task 运行失败，默认值：4。
mapreduce.task.timeout	Task 超时时间，经常需要设置的一个参数，该参数表达的意思为：如果一个 task 在一定时间内没有任何进入，即不会读取新的数据，也没有输出数据，则认为该 task 处于 block 状态，可能是卡住了，也许永远会卡主，为了防止因为用户程序永远 block 住不退出，则强制设置了一个该超时时间（单位毫秒），默认是 600000。如果你的程序对每条输入数据的处理时间过长（比如会访问

	数据库，通过网络拉取数据等），建议将该参数调大，该参数过小常出现的错误提示是“AttemptID:attempt_14267829456721_123456_m_000224_0 Timed out after 300 secsContainer killed by the ApplicationMaster.”。

6.3 HDFS 小文件优化方法

6.3.1 HDFS 小文件弊端

HDFS 上每个文件都要在 namenode 上建立一个索引，这个索引的大小约为 150byte，这样当小文件比较多的时候，就会产生很多的索引文件，一方面会大量占用 namenode 的内存空间，另一方面就是索引文件过大是的索引速度变慢。

6.3.2 解决方案

1）Hadoop Archive:

是一个高效地将小文件放入 HDFS 块中的文件存档工具，它能够将多个小文件打包成一个 HAR 文件，这样就减少了 namenode 的内存使用。

2）Sequence file：

sequence file 由一系列的二进制 key/value 组成，如果 key 为文件名，value 为文件内容，则可以将大批小文件合并成一个大文件。

3）CombineFileInputFormat：

CombineFileInputFormat 是一种新的 inputformat，用于将多个文件合并成一个单独的 split，另外，它会考虑数据的存储位置。

4）开启 JVM 重用

对于大量小文件 Job，可以开启 JVM 重用会减少 45%运行时间。

JVM 重用理解：一个 map 运行一个 jvm，重用的话，在一个 map 在 jvm 上运行完毕后，jvm 继续运行其他 map。

具体设置：mapreduce.job.jvm.numtasks 值在 10-20 之间。