Spark 大数据系统 basic 应用 垃圾回收性能分析

测试方案

1. 测试用例：
   选取 basic 典型的两个例子，GroupByTest, WordCount
2. 测试数据：

GroupByTest 参数设置：numMappers = 100，numKVParis = 900000，KeySize = 1000，numReducers = 36

WordCount 实验数据由程序随机生成字符串，约有两千万左右的数据

3. 测试环境

Application	Cores	Memory Per Node	Job Amount	Stage Amount	Task Amount
GroupByTest	42	3.0GB	2	3	236
WordCount	42	3.0GB	1	2	30

测试发现概况

1. 对于 GroupByTest，G1 算法和 CMS 算法表现不相上下（G1/CMS：20min/19min, 17min、16min）；对于 WordCount 而言，虽然这个应用本身的执行时间并不长，但是 Parallel GC 算法的性能是最优的（Parallel GC/CMS/G1: 1.2min/1.4min/2.1min）
2. GroupByTest：
   2.1 当使用 Parallel GC 时，应用没有办法顺利执行下去，会在 4min 左右自行中断，首先是 Java Heap Space 空间不能满足应用顺利正常执行下去，然后会出现 GC OverHead,在 task 不断丢失的情况下，应用就会 aborted
   2.2 虽然当 GroupByTest 使用 G1 和 CMS 性能不相上下，甚至 CMS 还要比 G1 快一点点，当使用 G1 算法时，会出现 task failed 然后 task 重做的现象，主要原因就是 Java Heap Space 不能满足当前应用运行，导致 OOM（OutOfMemory）
3. WordCount：
   3.1 如果我们设置 Rate = Task GC/Task Duration, 可以发现，当应用使用 G1 算法时，各个 Task 的 Rate 都不是很高，甚至 Total Task GC/Total Task Duration 都要比使用 CMS 或者 Parallel GC 时低，但是应用运行的总时间 G1 算法下却是最高的
   3.2 虽然 G1 算法下，从比值来看，各个 task 的 GC 都不是非常严重，且比较平均，但是，从真正的时长来看，正是因为使用了 G1 算法，使得 task 的执行时间变长（这与本身 G1 算法的性能有关）
   3.3 虽然 WordCount 这个应用比较小，但是小应用的好处就是任何一个细微的变化都会非常明显，由此应用也可能看到，G1 算法虽然理论上是最优的，但是还是要分情况使用，不是每种应用都适合使用 G1 作为垃圾回收

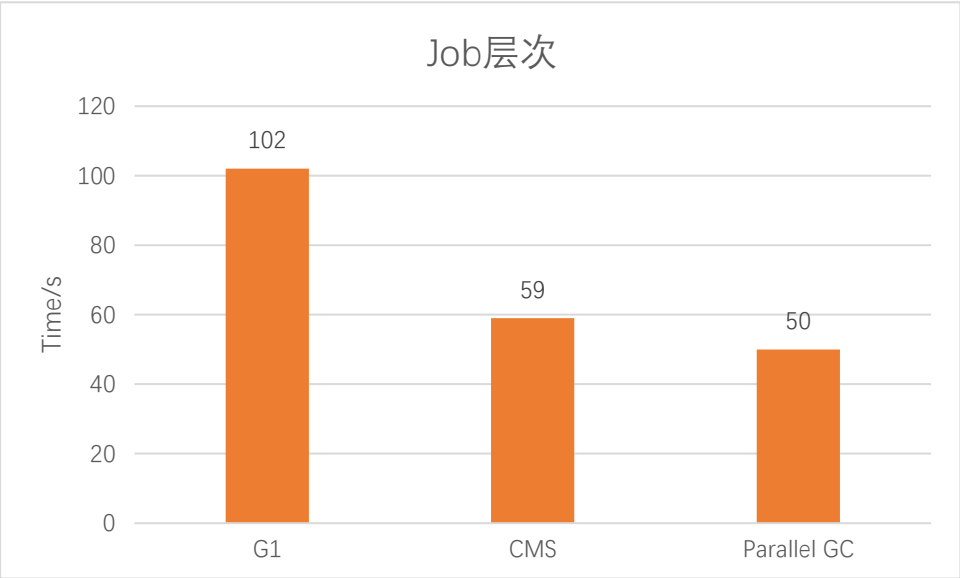
数据统计

WordCount

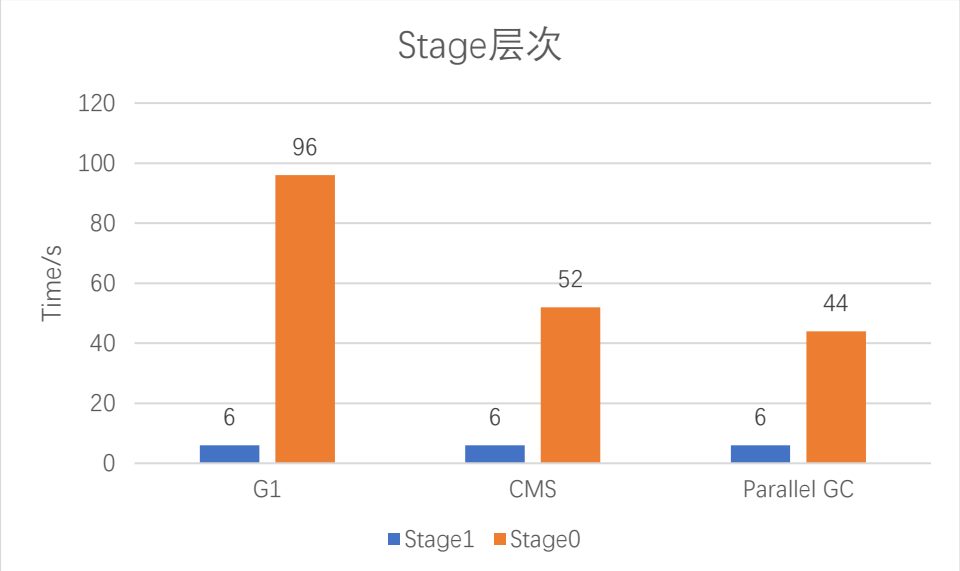
1. 应用采用不同 GC 算法，执行情况统计

GC Type	Job	Stage	Task	Duration
G1	1	2	30	2.1min
CMS	1	2	30	1.4min
Parallel GC	1	2	30	1.2min

Job 层次：



Stage 层次：



2. 因为 WordCount 的 job 和 stage 都相对较少，我们从 Task 层次详细对比一下

GC Type	Task Amount	Total GC Time/Total Time	Total Duration/Total Time

G1	30	0.149	0.850
CMS	30	0.224	0.776
Parallel GC	30	0.210	0.789

（其中，Total Time = Total Task Duration + Total GC Time）

对于每一个 task，我们计算了 Rate = Task GC Time /Task Duration，做出如下统计

GC Type	Rate>0.3	0.3>Rate>0.2	0.2>Rate
	Task Amount/ Total Task Amount		
G1	0	8/30	22/30
CMS	8/30	6/30	16/30
Parallel GC	8/30	9/30	13/30

因为 WordCount 只有两个 stage，相对来说，GC 相对严重的 Task 都在 Stage0 阶段，即完成 map at WordCount.

数据统计

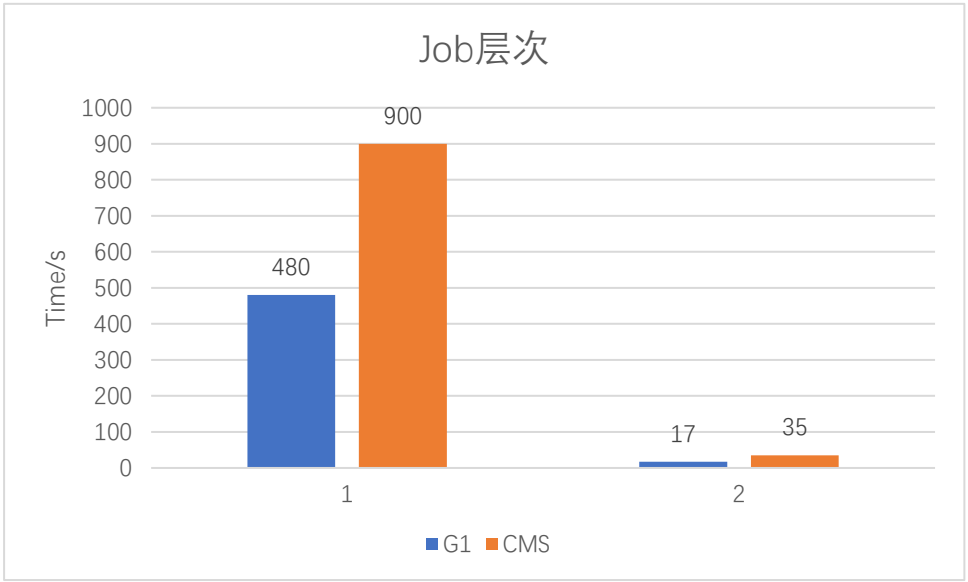
GroupByTest

因为 GroupByTest 在使用 Parallel GC 情况下，应用不会执行完全，所以在数据统计分析时，我们这里只分析在使用 G1 和 CMS 两种算法的情况下：

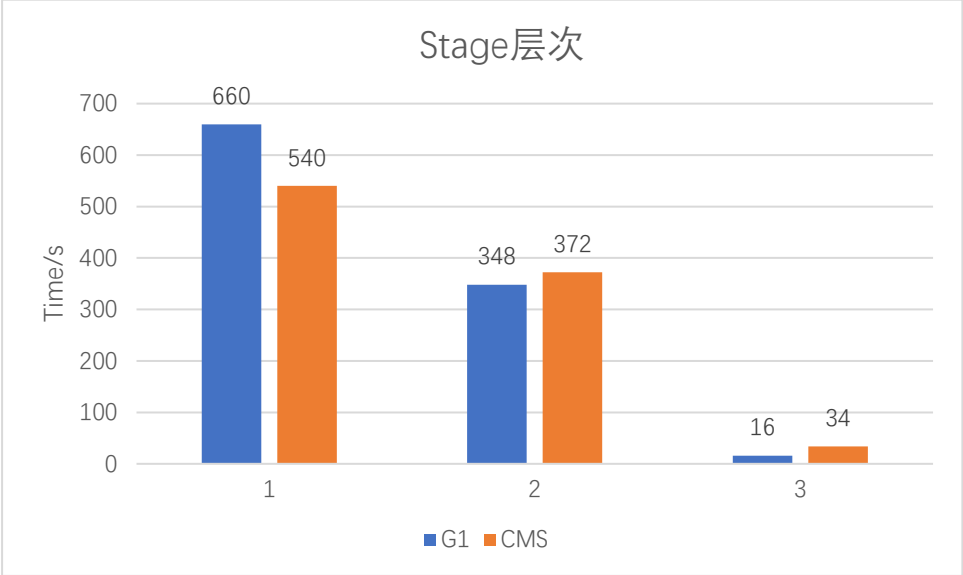
1. 应用采用不同 GC 算法，执行情况统计

GC Type	Job	Stage	Task	Duration
G1	2	3	236	17min
CMS	2	3	236	16min

Job 层次：



Stage 层次：



2. 因为 WordCount 的 job 和 stage 都相对较少，我们从 Task 层次详细对比一下

GC Type	Task Amount	Total GC Time/Total Time	Total Duration/Total Time
G1	238	0.016	0.983
CMS	236	0.032	0.967

（其中，Total Time = Total Task Duration + Total GC Time）

对于每一个 task，我们计算了 Rate = Task GC Time /Task Duration，做出如下统计

GC Type	Rate>0.4	0.4>Rate>0.2	0.2>Rate
	Task Amount/ Total Task Amount		
G1	1/238	35/238	202/238
CMS	2/236	94/236	140/236

各种算法下，gc 相对严重的 task 所在的 stage

其中 gc 相对严重是 Rate = Task GC Time /Task Duration 相对较大，或者 GC Time 相对较长

当使用 G1 算法时，Rate 相对较大的 task 都在 stage0，GC 时间相对较长的 task 都在 stage1

当使用 CMS 算法时，Rate 相对较大的 task 都在 stage0，GC 时间相对较长的 task 都在 stage1

其中 stage0 完成的工作是 count at GroupByTest

Stage1 完成的工作是 flatMap at GroupByTest.

测试中出现的问题

1. 理论上 G1 算法是最优的，而且数据表明当应用在使用 G1 算法时，GC 普遍较低或者与对应的 task 的 duraion 相比，相对较低，但是总体上来看，使用 G1 算法时，会使得应用执行时间变长

   主要原因是：Parallel GC 关注的是吞吐量，即运行用户代码的时间/（运行用户代码时间+垃圾收集时间），所以在小应用情况下，本身 GC 并不是非常明显的时候，使用 Parallel GC 算法，会使得吞吐量有保障

   而 CMS 算法和 G1 算法的关注点在于尽可能缩短垃圾收集时间，即 GC 时间的缩短是以吞吐量减小来换取的；

   同时，CMS 和 G1 的不同点在于，理论上来说，CMS 是最大限度的保障最短回收停顿时间的算法，而 G1 算法是要同时兼顾吞吐量和来及垃圾回收停顿时间两方面，而且从实验数据的角度来说，G1 算法的 GC 性能确实最优，但是涉及到 task 层次的执行时间时，总会比 CMS 弱一点点，合理怀疑是，G1 在宏观上采用的是标记整理算法，但是局部来看，是采用复制的策略，而 G1 算法采用的不是分区，而是将 java 堆分成一个一个小的 region，可能是没有一个合适的 region 让其复制，所以会产生较多次的 Full GC 现象；而 CMS 采用的原始分区方式，即 young eden – old eden，且采用的是标记-整理这种就地处理的方式，所以相对效果会好一点。