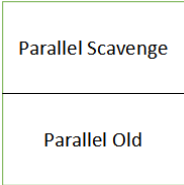
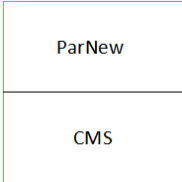
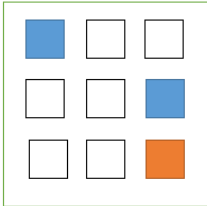


## Spark 大数据系统 垃圾回收性能分析 研究进展

### 测试方案

- 1. 选取典型的测试用例，通过改变不同的 GC 算法，观察性能指标
- 2. 通过 Profiler 收集应用的 Job，Stage，Task 信息，通过数据分析，得到在只改变 GC 算法的情况下，Job，Stage，Task 的执行情况的异同
- 3. GC 参数设置：

算法	图例	参数
Parallel Scavenge  Parallel Old		-XX:UseParallelGC
ParNew  CMS		-XX:UseConcMarkSweepGC
G1		-XX:+UseG1GC

## 测试用例及发现

Type	AppName	Data Source	Findings
Basic	GroupByTest	numMappers =100 numKVPairs=900000 KeySize=1000 numReducers=36	<p>当设置参数足够大时，会出现以下情况：</p> <ol style="list-style-type: none"> <li>1. 在使用 CMS 和 G1 算法时，二者性能不相上下，相差 1min 左右（CMS 略优于 G1）</li> <li>2. 在使用 Parallel GC 算法时，应用不能顺利执行下去，会出现 <b>GC OverHead</b></li> </ol>
Basic	WordCount	Self-Made (Using Random String)	<ol style="list-style-type: none"> <li>1. 从应用的吞吐量<sup>1</sup>角度来看，使用三种算法时，应用吞吐量从高到低为：<b>G1&gt;CMS&gt;Parallel GC</b></li> <li>2. 从本身应用执行情况来看，应用执行时间从长到短为：<b>G1&gt;CMS&gt;Parallel GC</b></li> </ol>
Graphx	TriangleCount	soc-LiveJournal1 (Nodes:4,,847,571 Edges:68,993,773) <a href="http://snap.stanford.edu/data/">http://snap.stanford.edu/data/</a>	<ol style="list-style-type: none"> <li>1. 对于 Graphx 类应用，如果不选用适合的 GC 算法，会频繁出现 <b>OOM 现象</b></li> <li>2. 当采用 Parallel GC 和 G1 算法时，会出现 Stage Retry 现象</li> <li>3. 在使用不同 GC 算法时，应用吞吐量由高到低为 <b>G1&gt;CMS&gt;Parallel GC</b></li> <li>4. 在使用不同 GC 算法时，应用执行时间由短到长为 <b>G1&lt;CMS&lt;Parallel GC</b></li> </ol>
Graphx	PageRank	soc-LiveJournal1 (Nodes:4,,847,571 Edges:68,993,773) <a href="http://snap.stanford.edu/data/">http://snap.stanford.edu/data/</a>	<ol style="list-style-type: none"> <li>1. 当采用 Parallel GC 算法时，应用没有办法顺利执行下去，会长时间执行（经过多次实验，在使用全部集群资源的情况下，72core, 108G 时，应用在执行 <b>3h</b> 以后依然无法执行出结果）</li> <li>2. 当采用 CMS 和 G1 算法时，应用可以在 <b>10min</b> 以内执行出结果</li> </ol>

Mllib	LogisticRegressionWithLBFGS	数据量 24G Partition Num = 13	在采用三种算法的情况下，应用执行时间为： <b>G1&gt;Parallel GC&gt;CMS</b>
-------	-----------------------------	-------------------------------	---

<sup>1</sup> 吞吐量 = 运行用户代码时间 / (运行用户代码时间 + 垃圾收集时间)

ps: 个人认为，这个公式不是特别合理，即在多线程情况下，在使用 CMS 和 G1 时，会出现 GC 和程序代码并行情况，会出现时间重复计算

## 研究计划

初期的研究关注在理论学习，工具准备以及对于应用运行的表层现象的关注，在接下来的研究中，会深入到具体应用的内部，从代码角度，存放内存中的中间数据以及缓存到内存中重用的数据三个方面来分析

- (1) 在单核环境下重跑 basic 实验
- (2) 用默认 GC 算法，改变 `--executor-memory` 看资源竞争对应用的影响
- (3) 在某个阈值时，当资源对应用没有影响时，改变 GC 算法；如果在现有资源下，没能找到一个合适的阈值，则用全部资源跑这个应用
- (4) 在单核条件下，改变 GC 算法后，看 stage 的变化情况（包括 GC 的变化以及其 Duration 的变化）
- (5) 从代码角度，中间数据，缓存数据三个角度重点分析变化的 stage
- (6) 利用监控工具，得到合理的运行数据统计图