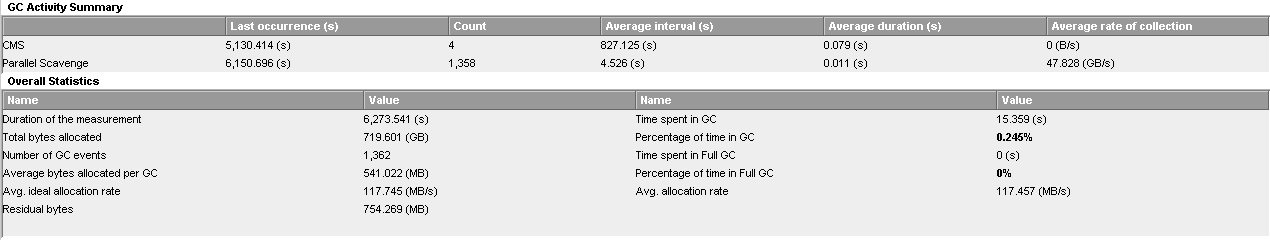
# GC分析

## 一、JVM运行参数

JAVA\_OPTS="-server -Xms1800m -Xmx1800m -Xmn680m -Xss256k -XX:PermSize=240m -XX:MaxPermSize=240m -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:CMSFullGCsBeforeCompaction=5 -XX:+UseCMSCompactAtFullCollection -XX:+CMSClassUnloadingEnabled -XX:+DisableExplicitGC -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Dcom.sun.management.jmxremote.port=9981 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"

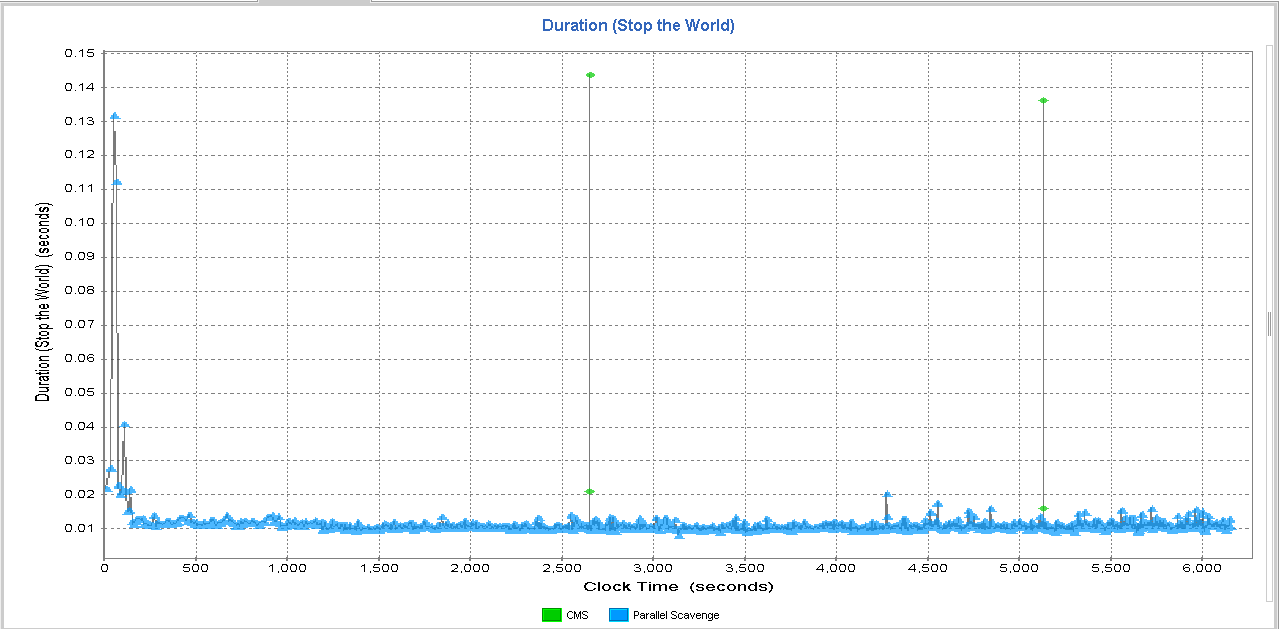
## 二、GC概要



图一

如图一所示，1 hour 42 minute的压测期间(qps：200 -> 1000)：CMS GC发生4次，平均持续时间0.079秒，平均间隔时间827.125秒，ParNew GC发生1358次，平均持续时间0.011秒，平均间隔时间4.526秒。GC负载：0.245%，无Full GC发生。

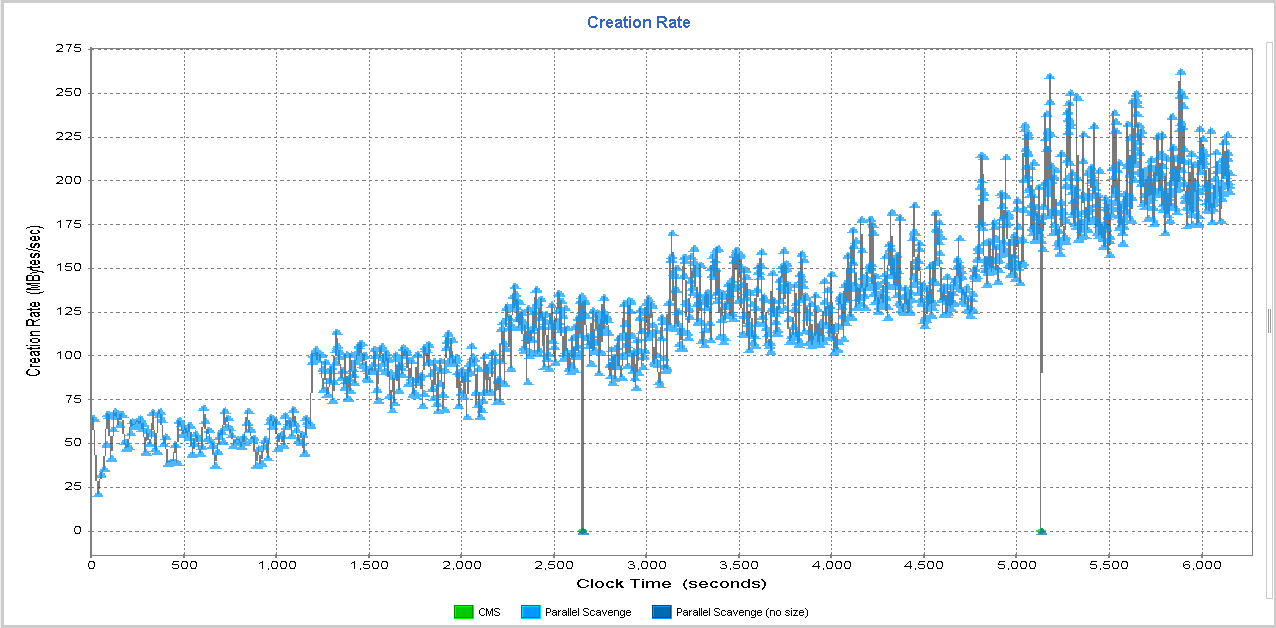
## 三、GC耗时分布



图二

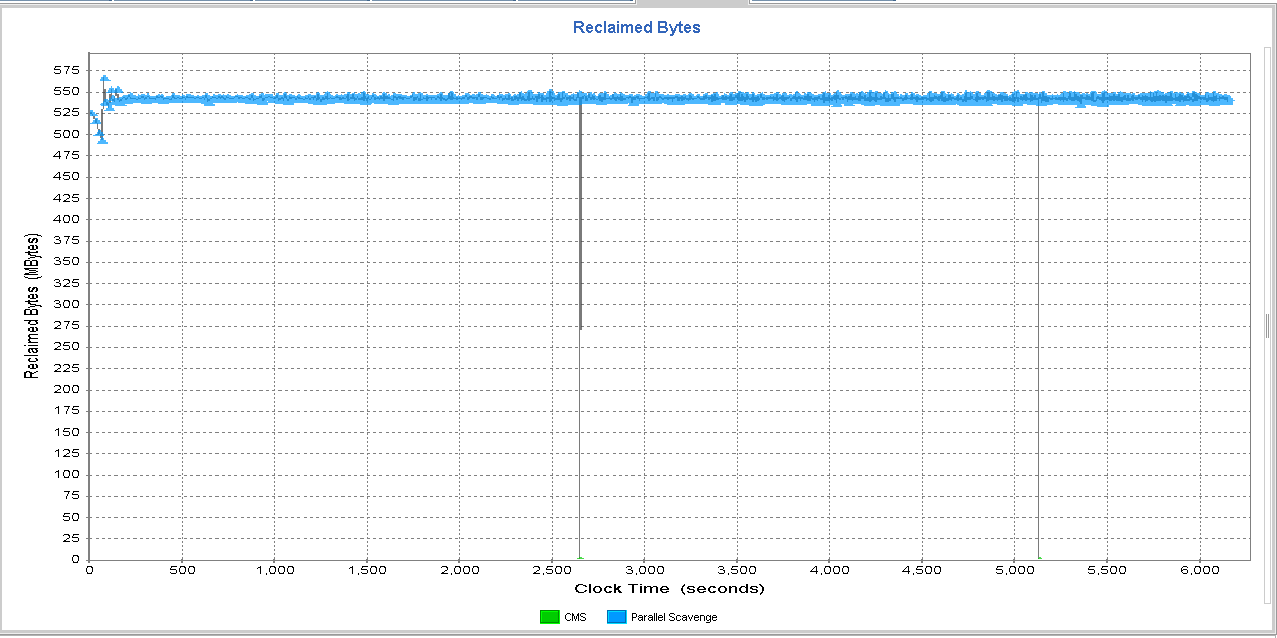
大多数YGC的耗时在0.01-0.02秒之间，CMS耗时最高不超过0.15秒。

## 四、对象创建速率及年龄分布

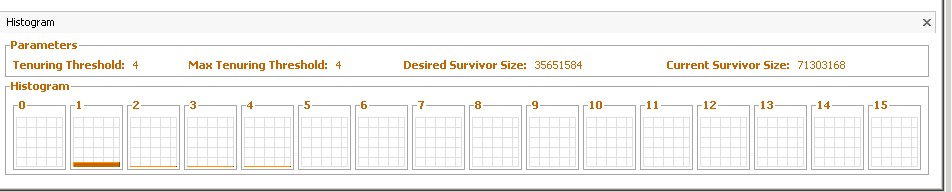


图三

如图三所示，对象创建的速率基本与压测倍数保持正比。另外从gc日志里看到，随着压力的增大，ygc的间隔时间也逐渐变短，从最初的5秒多一次缩短到3秒多一次。但每次ygc的效率都很高，基本都是从570M回收到20M左右，如图四所示。Survivor区对象的年龄分布也很合理，年龄越大的对象越少。最大年龄稳定在4。survivor区的大小略微有点大。如下图五所示。



图四



图五

## 五、对象晋升及old区CMS分析

每次ygc之后晋升到old区的对象大约才2M，所以old区增长比较缓慢。当old区使用率增大到约92%时触发CMS,回收的效率很高，从1030M回收到约140M。

## 六、perm区分析

Perm区利用率比较稳定，在QPS1000的情况下，依然保持在80M以下。

## 六、结论与建议

GC总体表现良好。但仍可进行一些微调，建议如下：

1. survivorratio可调整到10
2. perm区调整到128M
3. 设置CMSInitiatingOccupancyFraction为75%，并且设置UseCMSInitiatingOccupancyOnly=true。