Analysis Report

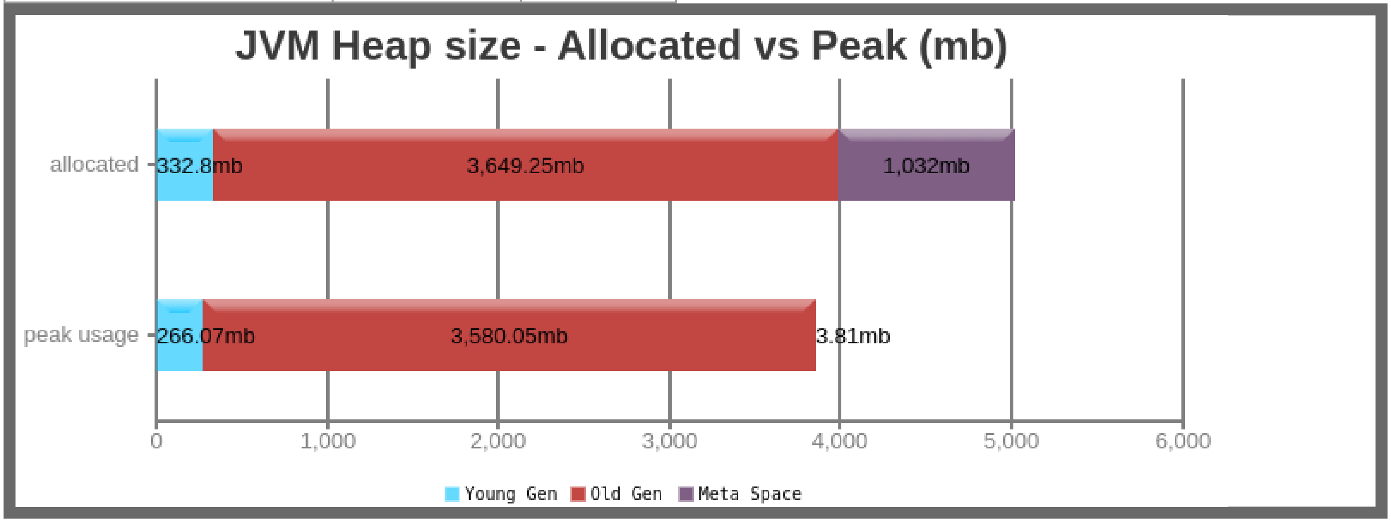
Tips to reduce GC Time

(**CAUTION:** Please do thorough testing before implementing out the recommendations. These are generic recommendations & may not be applicable for your application.)

- ✓ **5.28%** of GC time (i.e 430 ms) is caused by **'Concurrent Mode Failure'**. The CMS collector uses one or more garbage collector threads that run simultaneously with the application threads with the goal of completing the collection of the tenured generation before it becomes full. In normal operation, the CMS collector does most of its tracing and sweeping work with the application threads still running, so only brief pauses are seen by the application threads. However, if the CMS collector is unable to finish reclaiming the unreachable objects before the tenured generation fills up, or if an allocation cannot be satisfied with the available free space blocks in the tenured generation, then the application is paused and the collection is completed with all the application threads stopped. The inability to complete a collection concurrently is referred to as concurrent mode failure and indicates the need to adjust the CMS collector parameters. Concurrent mode failure typically triggers Full GC..
  Solution:
  The concurrent mode failure can either be avoided by increasing the tenured generation size or initiating the CMS collection at a lesser heap occupancy by setting CMSInitiatingOccupancyFraction to a lower value and setting UseCMSInitiatingOccupancyOnly to true. CMSInitiatingOccupancyFraction should be chosen carefuly, setting it to a low value will result in too frequent CMS collections.

JVM Heap Size

Generation	Allocated	Peak
Young Generation	332.8 mb	266.07 mb
Old Generation	3.56 gb	3.5 gb
Meta Space	1.01 gb	3.81 mb
Young + Old + Meta space	4.89 gb	3.57 gb



Key Performance Indicators

(Important section of the report. To learn more about KPIs, click here (https://blog.gceasy.io/2016/10/01/garbage-collection-kpi/))

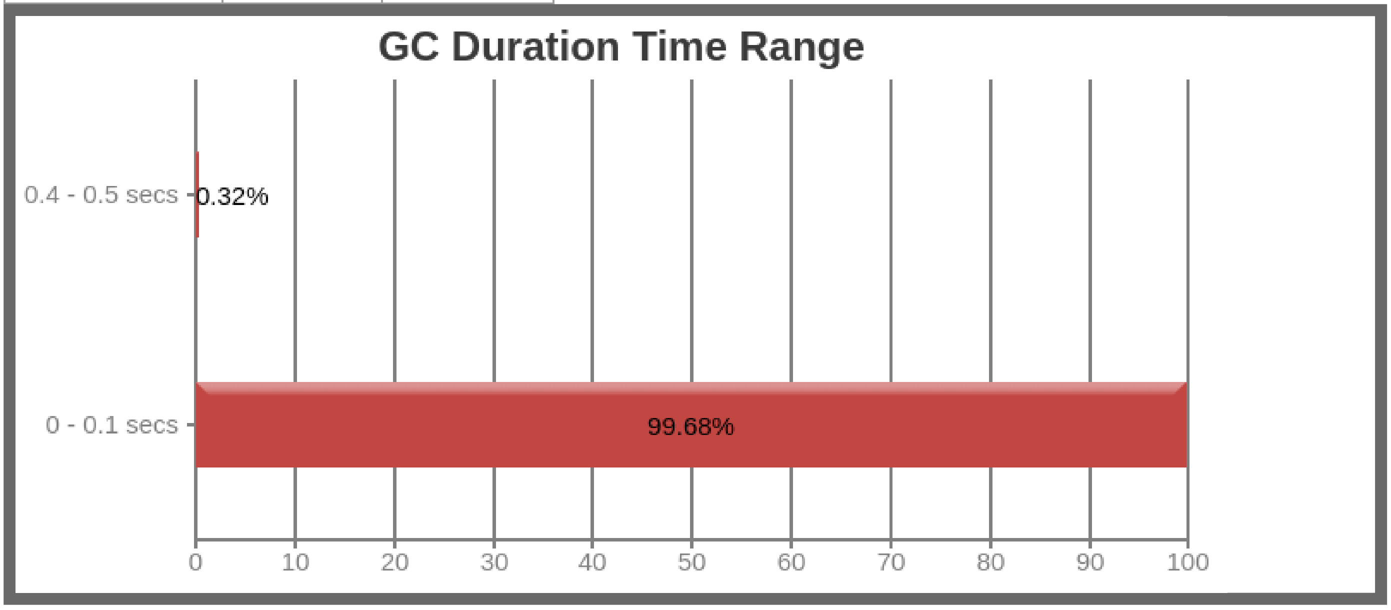
1 **Throughput : 55.228%**

2 **Latency:**

Avg Pause GC Time	**19 ms**
Max Pause GC Time	**430 ms**

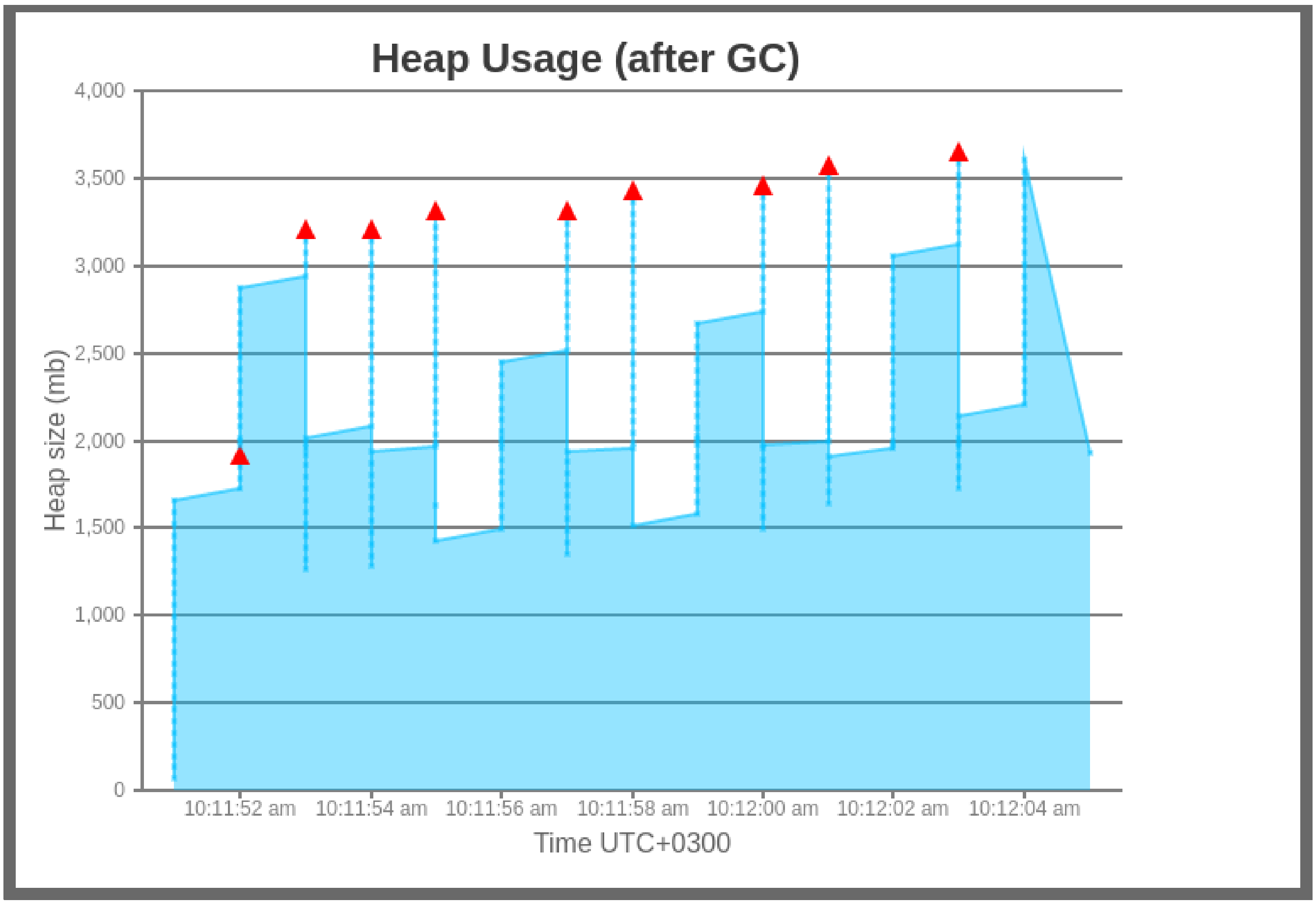
GC **Pause** Duration Time Range :

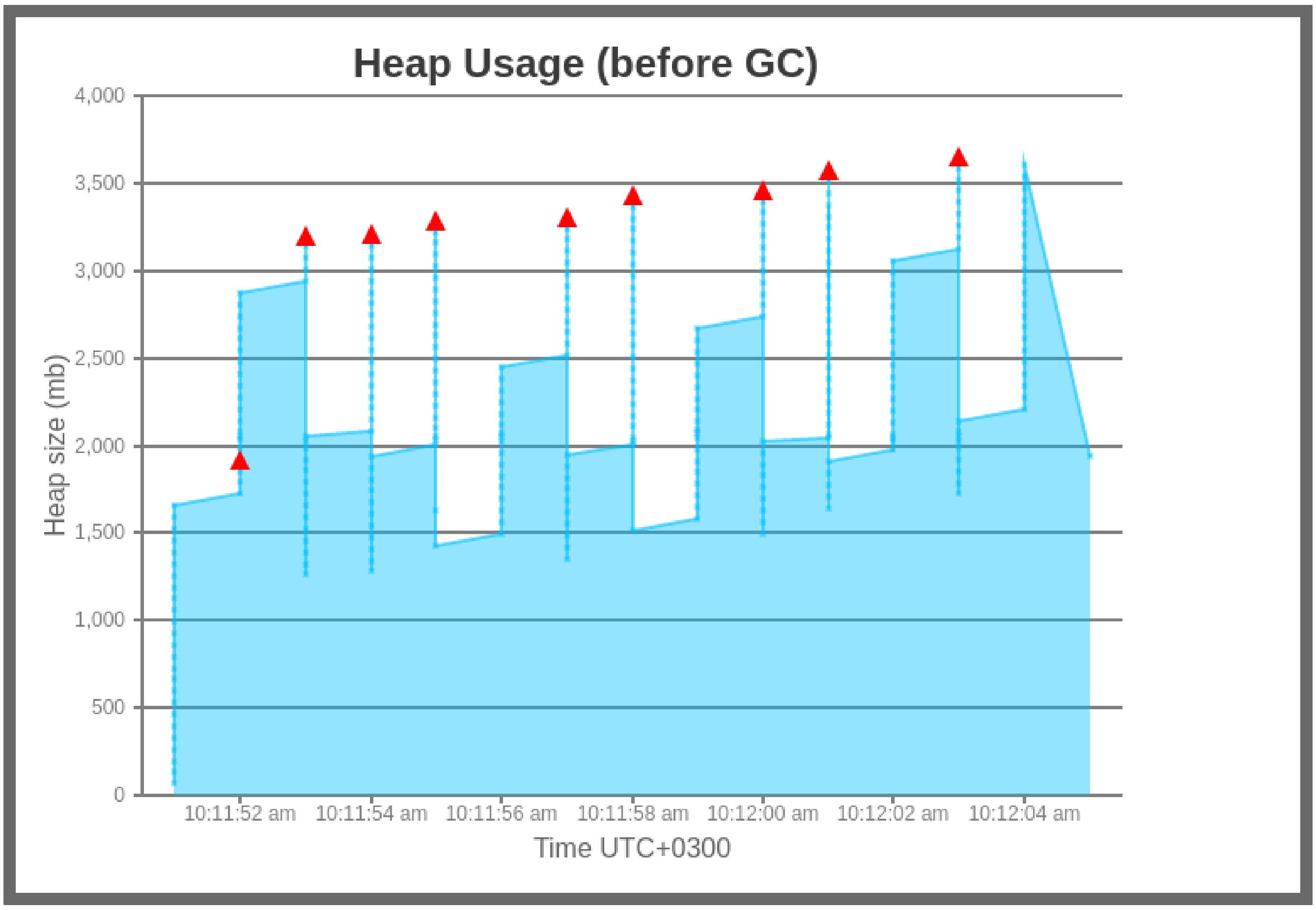
Duration (secs)	No. of GCs	Percentage
0 - 0.1	316	99.685%
0.4 - 0.5	1	100.0%

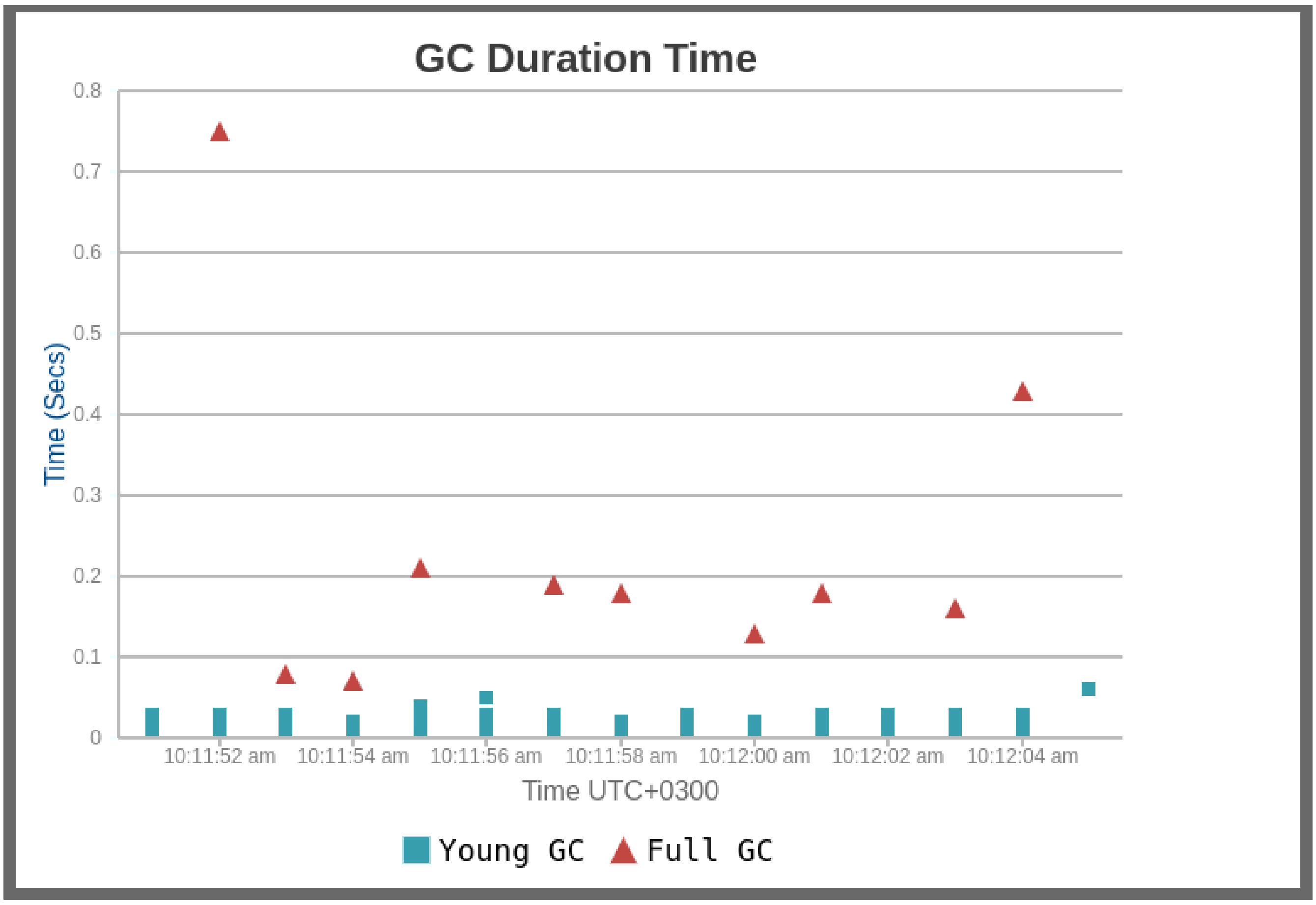


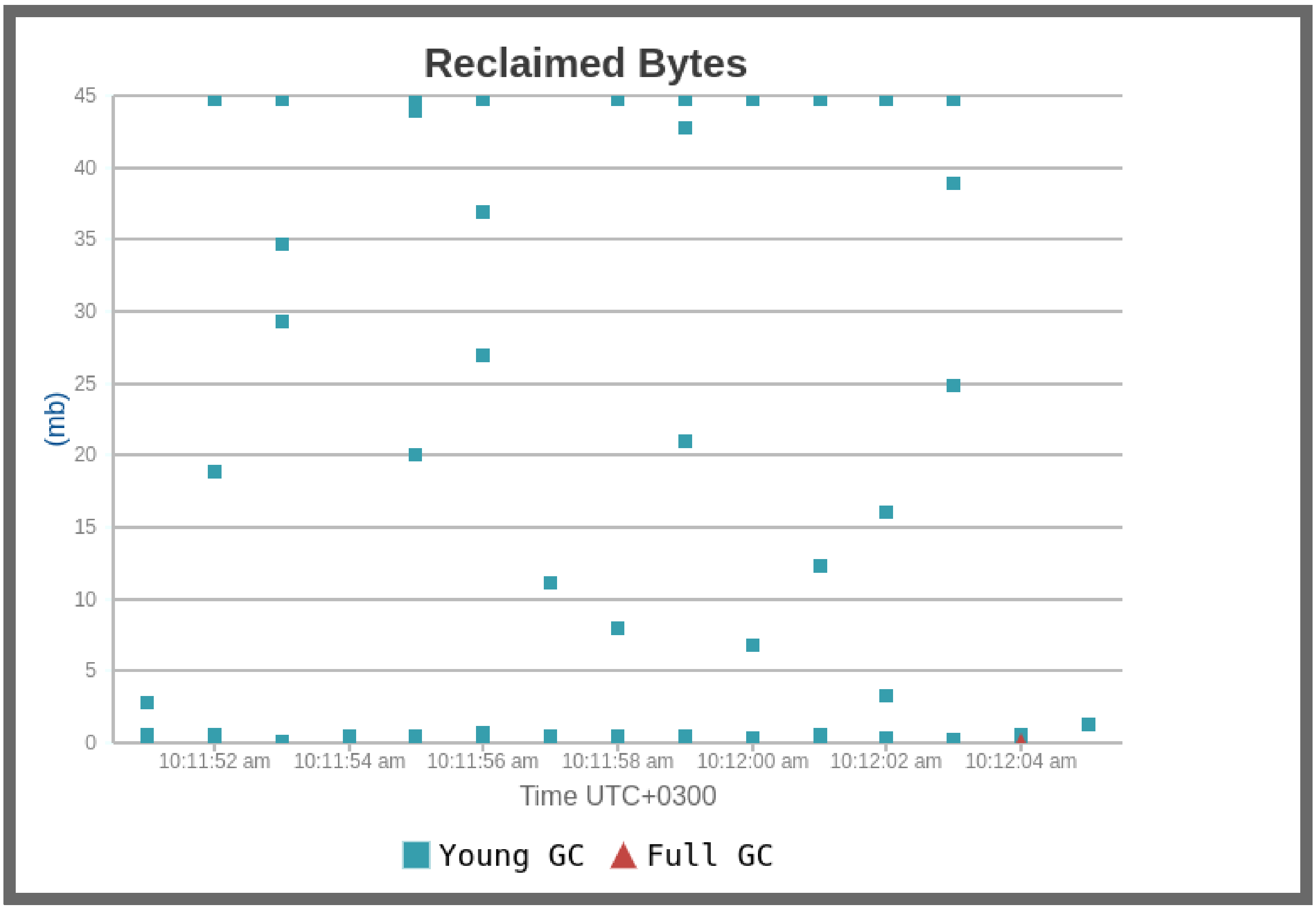
Interactive Graphs

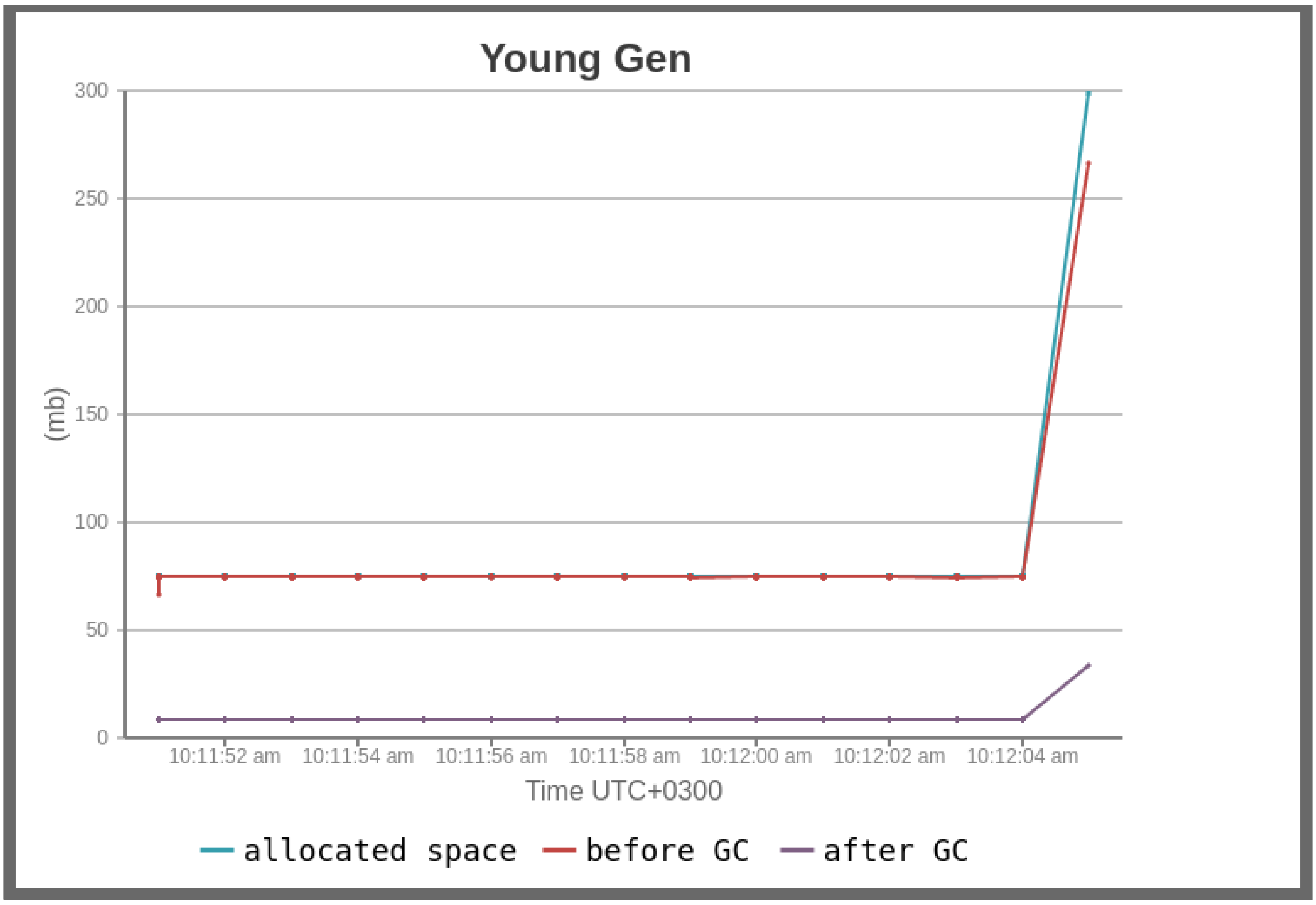
(All graphs are zoomable)

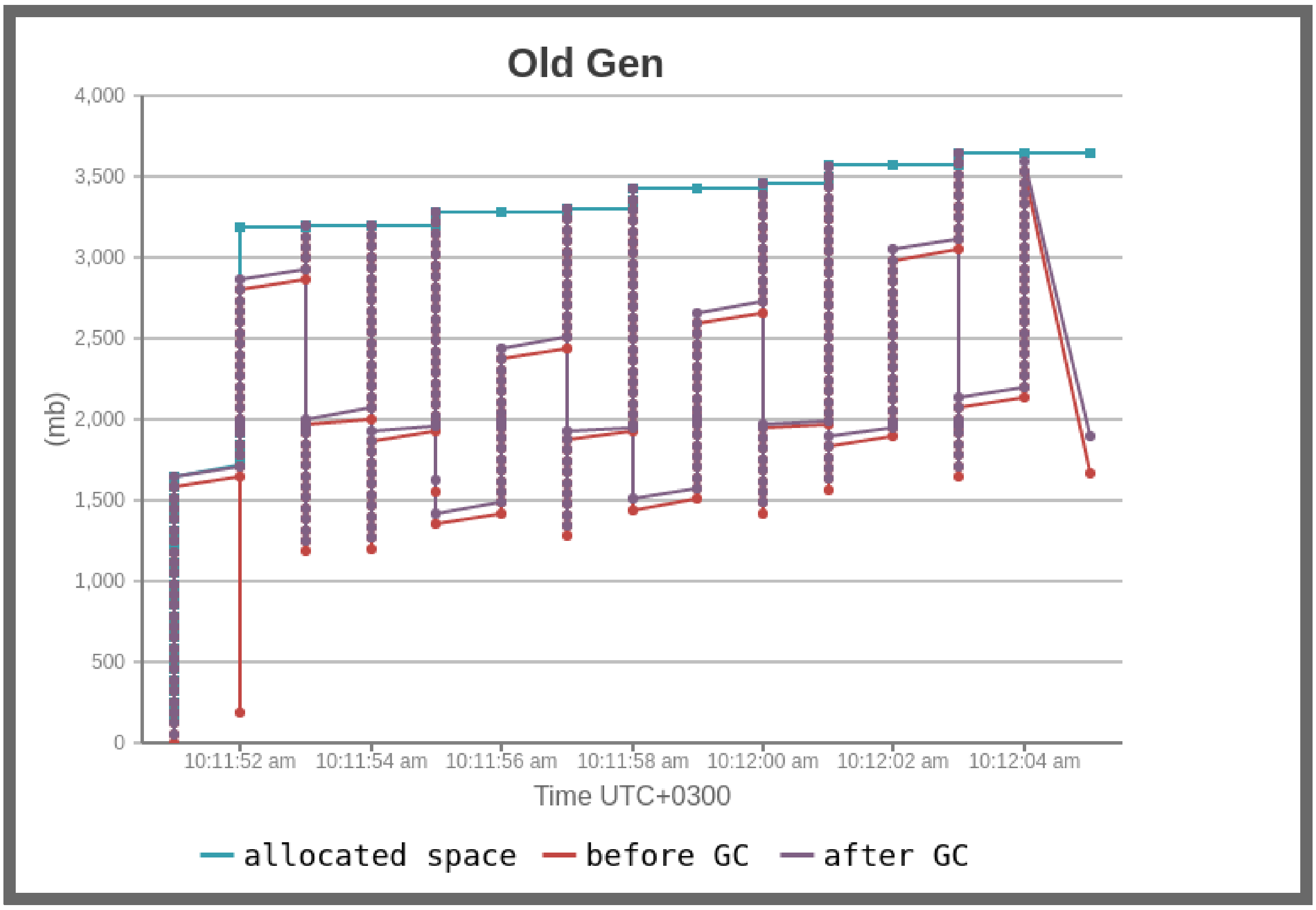

Heap Usage (after GC)
Heap size (mb)
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300

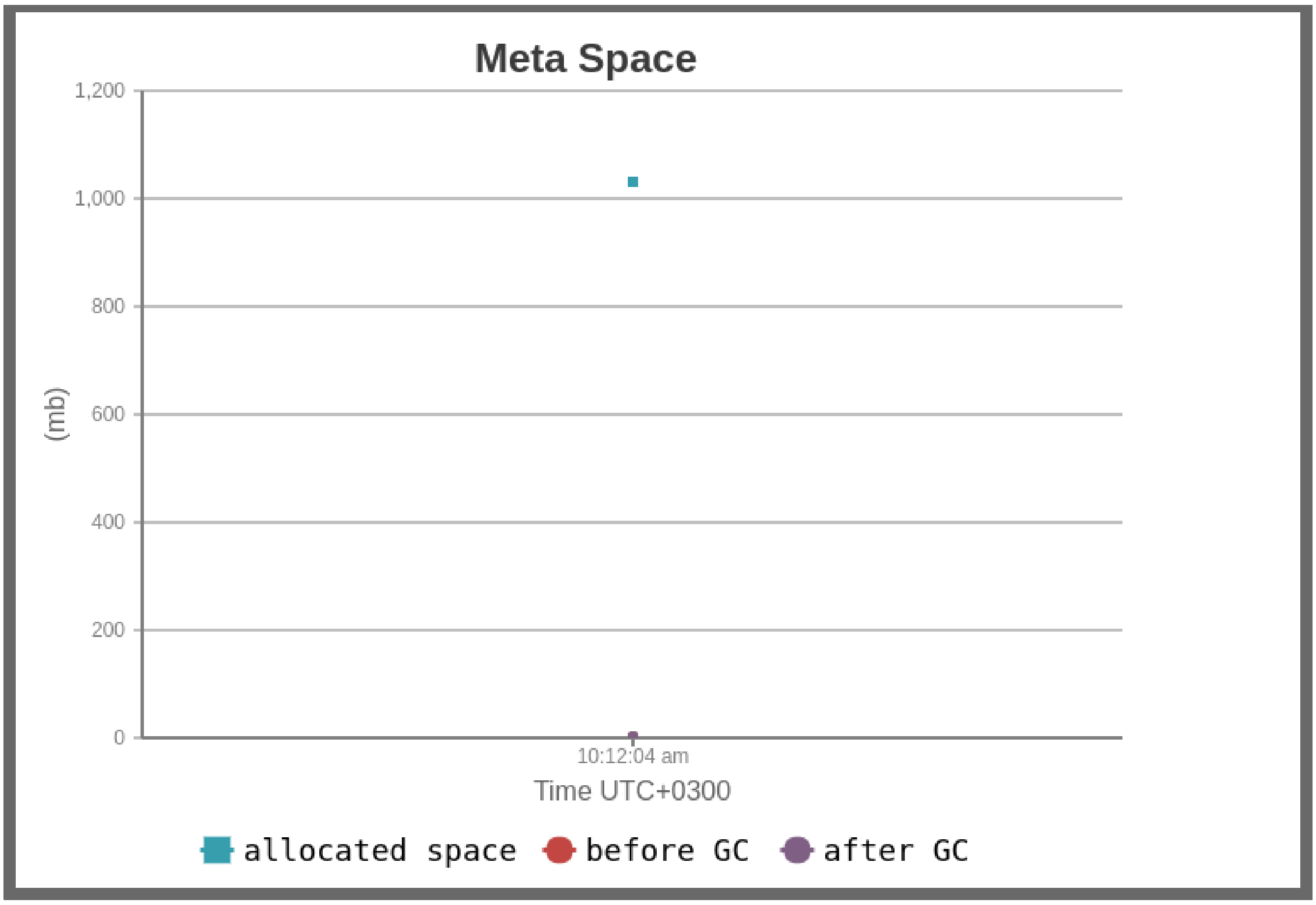

Heap Usage (before GC)
Heap size (mb)
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300

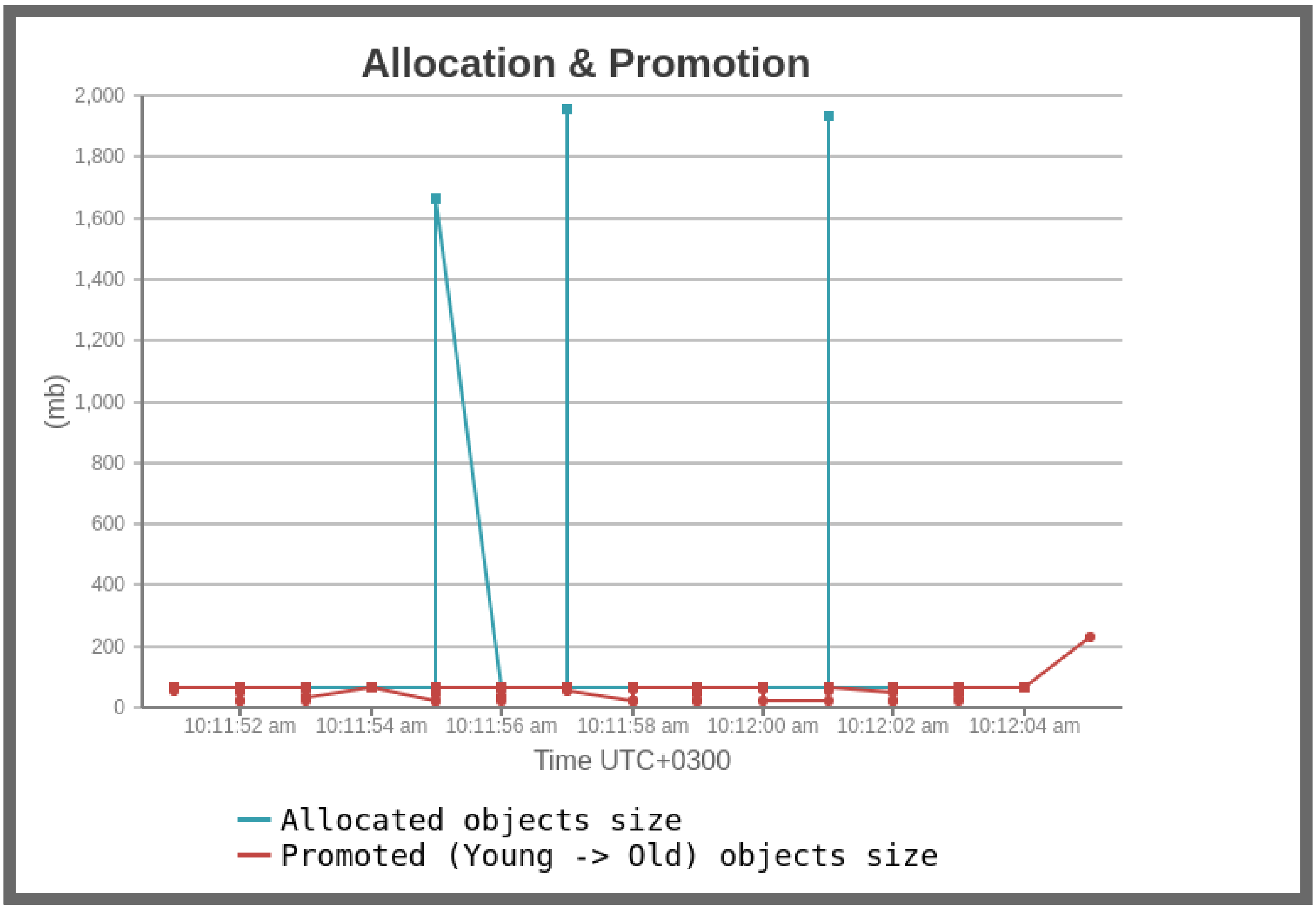

GC Duration Time
Time (Secs)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300
Young GC
Full GC


Reclaimed Bytes
(mb)
45
40
35
30
25
20
15
10
5
0
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300
Young GC
Full GC

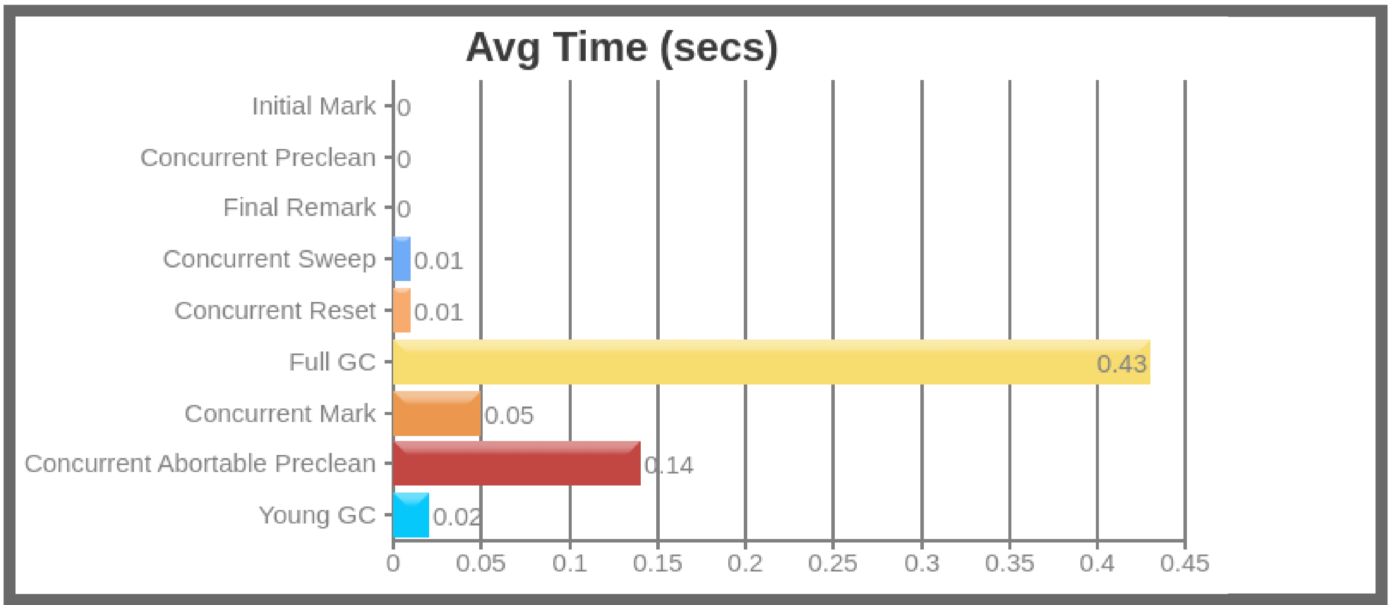

Young Gen
300
250
200
150
100
50
0
(mb)
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300
allocated space
before GC
after GC

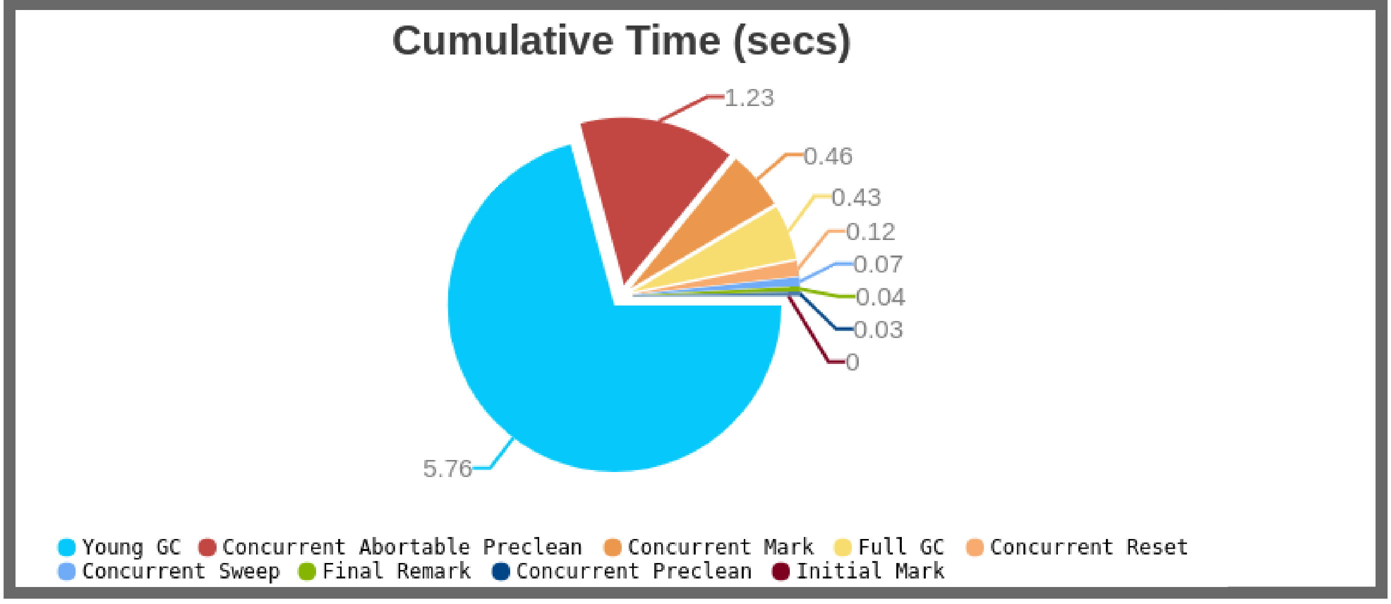

Old Gen
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
(mb)
10:11:52 am
10:11:54 am
10:11:56 am
10:11:58 am
10:12:00 am
10:12:02 am
10:12:04 am
Time UTC+0300
allocated space
before GC
after GC





CMS Collection Phases Statistics

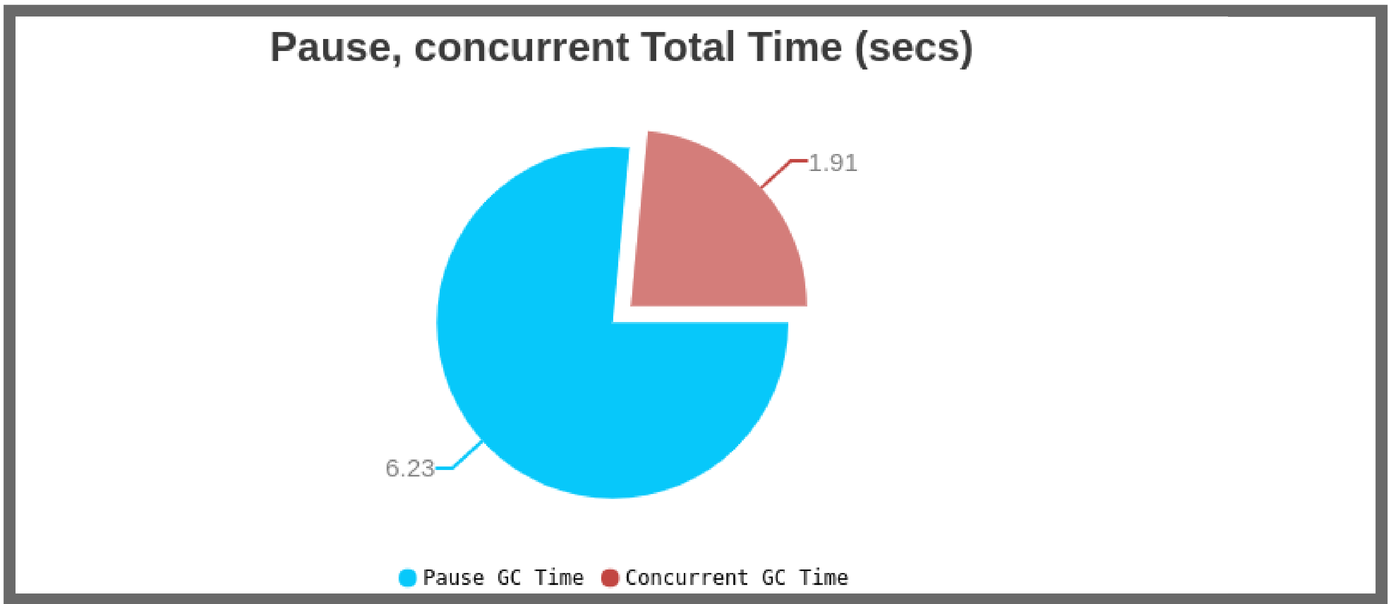


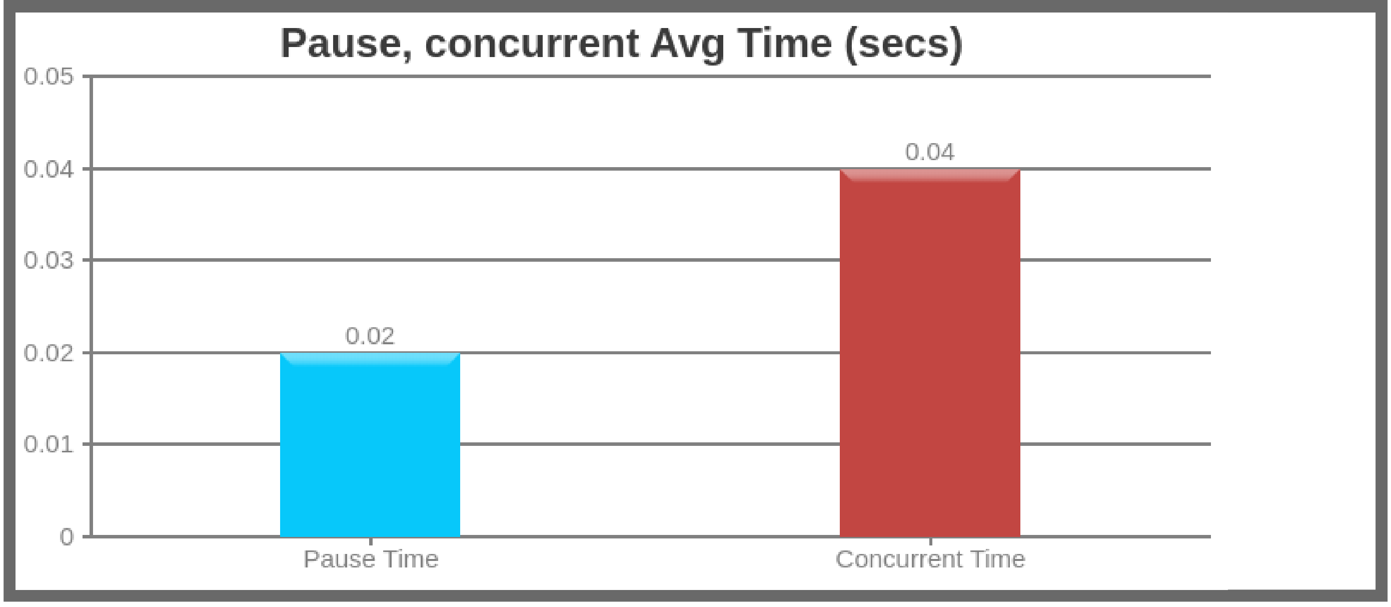


	Young GC	**Concurrent Abortable Preclean**	**Concurrent Mark**	**Full GC**	**Concurrent Reset**	**Concurrent Sweep**	**Final Remark**	**Concurrent Preclean**	**Initial Mark**
Total Time	5 sec 760 ms	1 sec 230 ms	460 ms	430 ms	120 ms	70 ms	40 ms	30 ms	0
Avg Time	18 ms	137 ms	51 ms	430 ms	13 ms	8 ms	4 ms	3 ms	0
Std Dev Time	6 ms	205 ms	15 ms	0	9 ms	9 ms	5 ms	5 ms	0
Min Time	10 ms	20 ms	20 ms	430 ms	0	0	0	0	0

Max Time ❓	60 ms	710 ms	70 ms	430 ms	30 ms	30 ms	10 ms	10 ms	0
Count ❓	312	9	9	1	9	9	9	9	9

CMS GC Time





Pause Time ❓

Total Time	6 sec 230 ms
Avg Time	19 ms
Std Dev Time	24 ms

Min Time	0
Max Time	430 ms

Concurrent Time

Total Time	1 sec 910 ms
Avg Time	42 ms
Std Dev Time	105 ms
Min Time	0
Max Time	710 ms

Object Stats

(These are perfect micro-metrics (https://blog.gceasy.io/2017/05/30/improving-your-performance-reports/) to include in your performance reports)

Total created bytes	24.9 gb
Total promoted bytes	19.41 gb
Avg creation rate	1.79 gb/sec
Avg promotion rate	1.39 gb/sec

Memory Leak

No major memory leaks.

(**Note:** there are 8 flavours of OutOfMemoryErrors (https://tier1app.files.wordpress.com/2014/12/outofmemoryerror2.pdf). With GC Logs you can diagnose only 5 flavours of them(Java heap space, GC overhead limit exceeded, Requested array size exceeds VM limit, Permgen space, Metaspace). So in other words, your application could be still suffering from memory leaks, but need other tools to diagnose them, not just GC Logs.)

Consecutive Full GC

None.

Long Pause

None.

Safe Point Duration

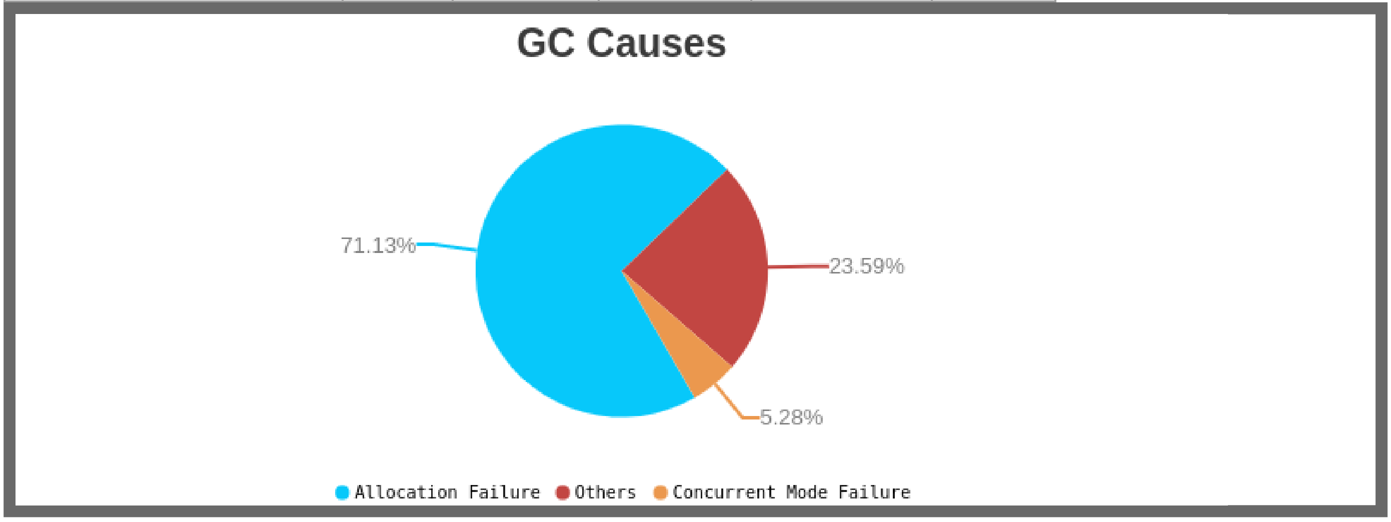
(To learn more about SafePoint duration, click here (https://blog.gceasy.io/2016/12/22/total-time-for-which-application-threads-were-stopped/))

Not Reported in the log.

GC Causes

(What events caused the GCs, how much time it consumed?)

Cause	Count	Avg Time	Max Time	Total Time	Time %
Allocation Failure	313	18 ms	60 ms	5 sec 790 ms	71.13%
Others	8	n/a	n/a	1 sec 920 ms	23.59%
Concurrent Mode Failure	1	430 ms	430 ms	430 ms	5.28%
Total	322	n/a	n/a	8 sec 140 ms	100.0%



Tenuring Summary

Not reported in the log.

Command Line Flags

```
-XX:InitialHeapSize=260849920 -XX:MaxHeapSize=4173598720 -XX:MaxNewSize=348966912 -XX:MaxTenuringThreshold=6 -XX:OldPLABSize=16 -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCMSInitiatingOccupancyOnly -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:+UseParNewGC
```