

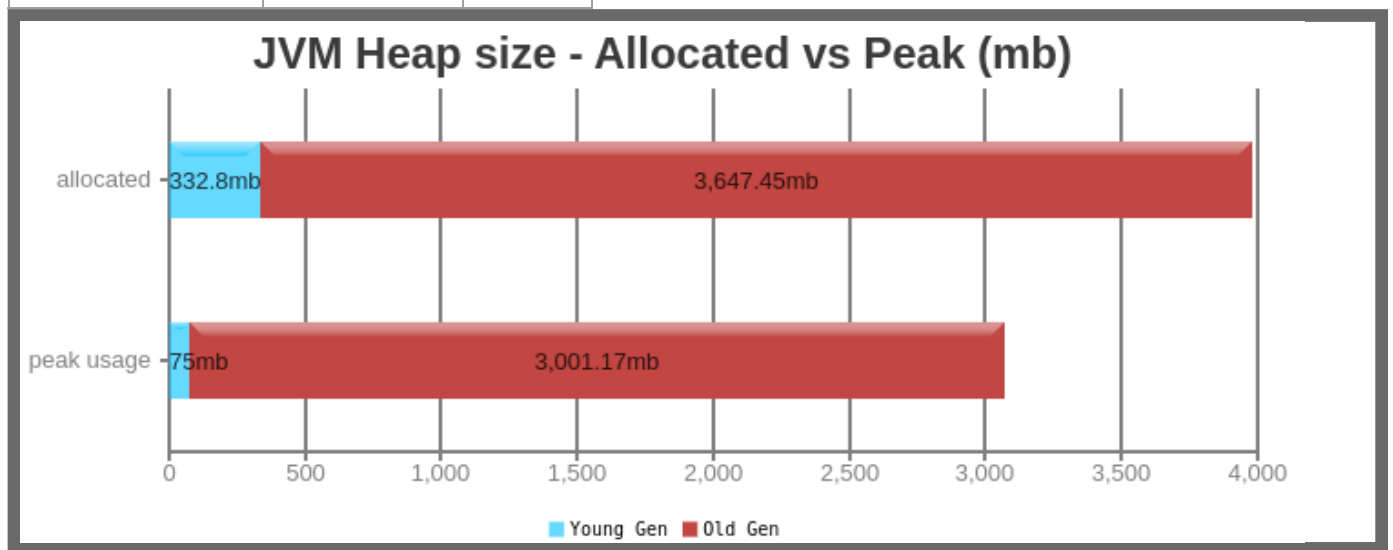


Analysis Report

JVM Heap Size

Generation	Allocated 	Peak 
Young Generation	332.8 mb	75 mb
Old Generation	3.56 gb	2.93 gb
Total	3.89 gb	3 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/) (https://blog.gceasy.io/2016/10/01/garbage-collection-kpi/))

1 **Throughput**  : 58.385%

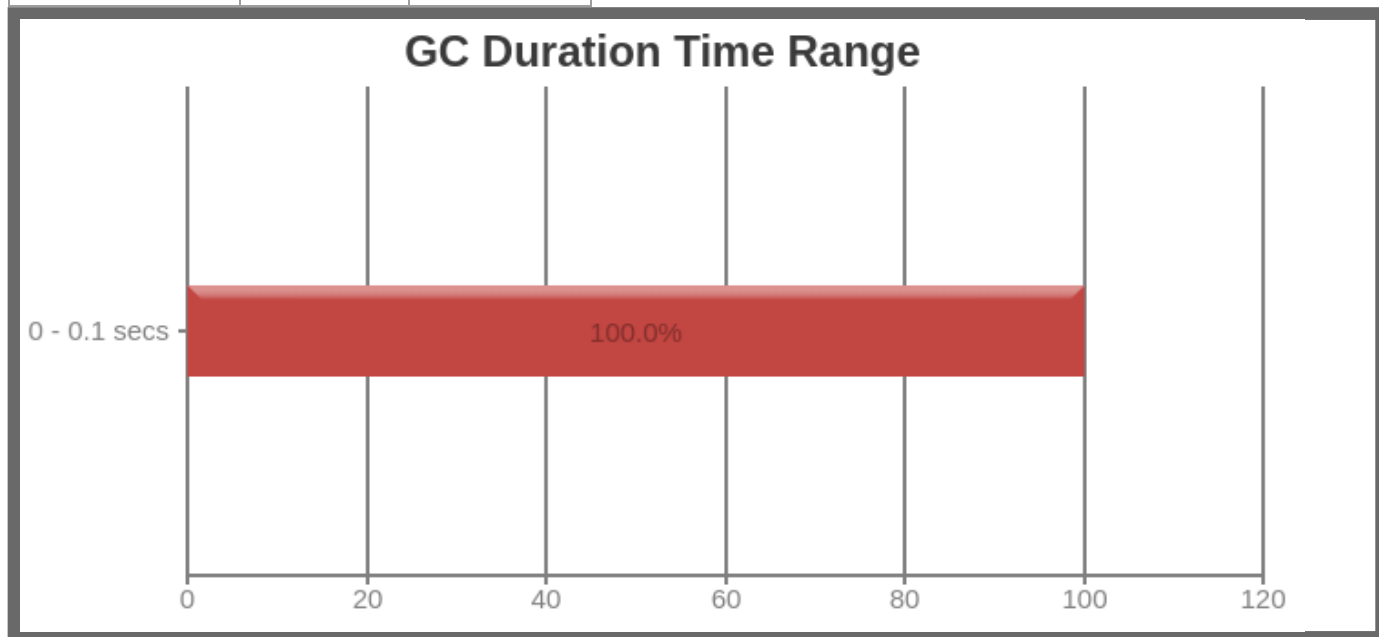
2 **Latency:**

Avg Pause GC Time 	16 ms
Max Pause GC Time 	30 ms

GC **Pause** Duration Time Range 

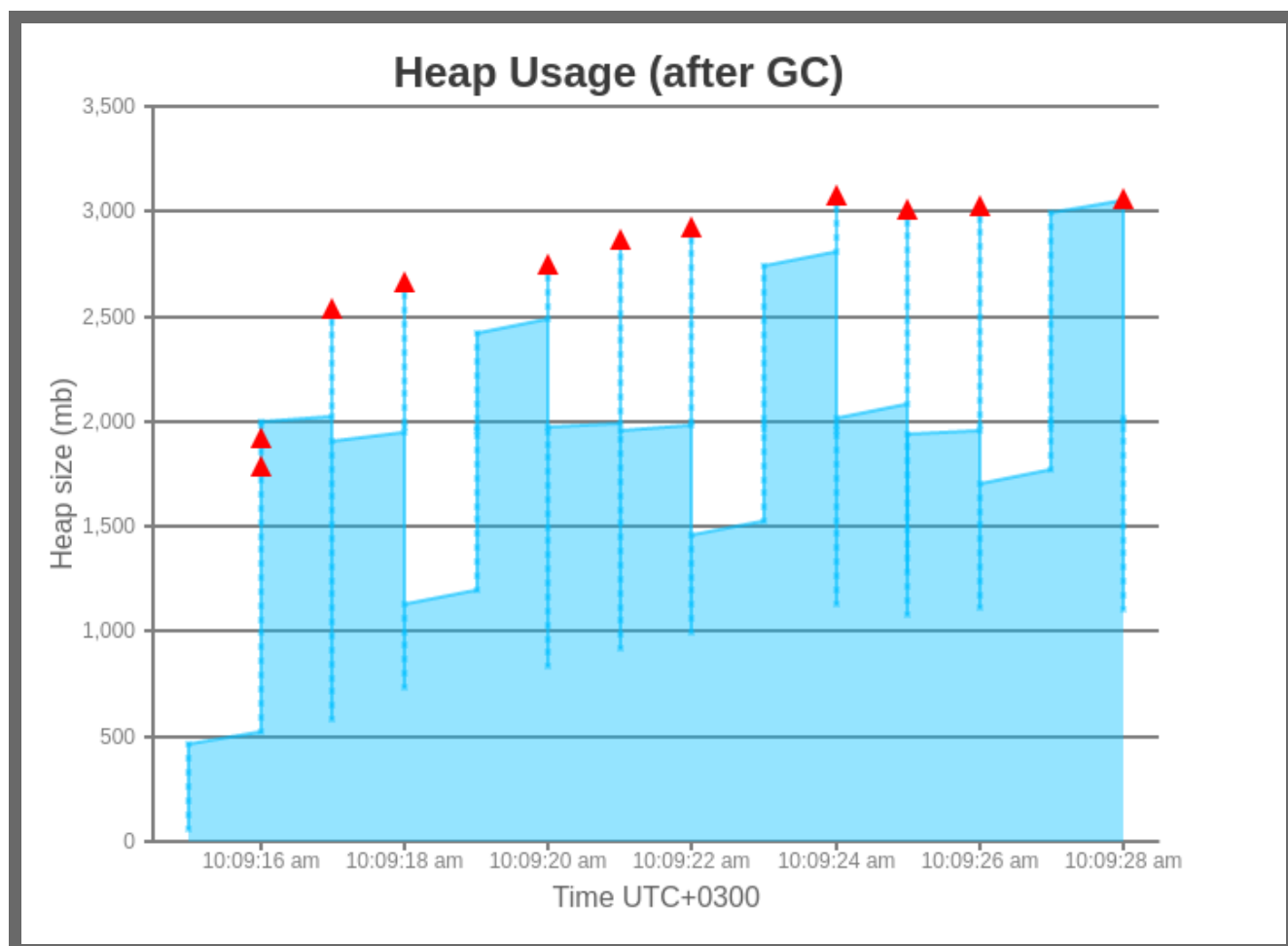
Duration (secs)	No. of GCs	Percentage
-----------------	------------	------------

0 - 0.1	318	100.0%
---------	-----	--------

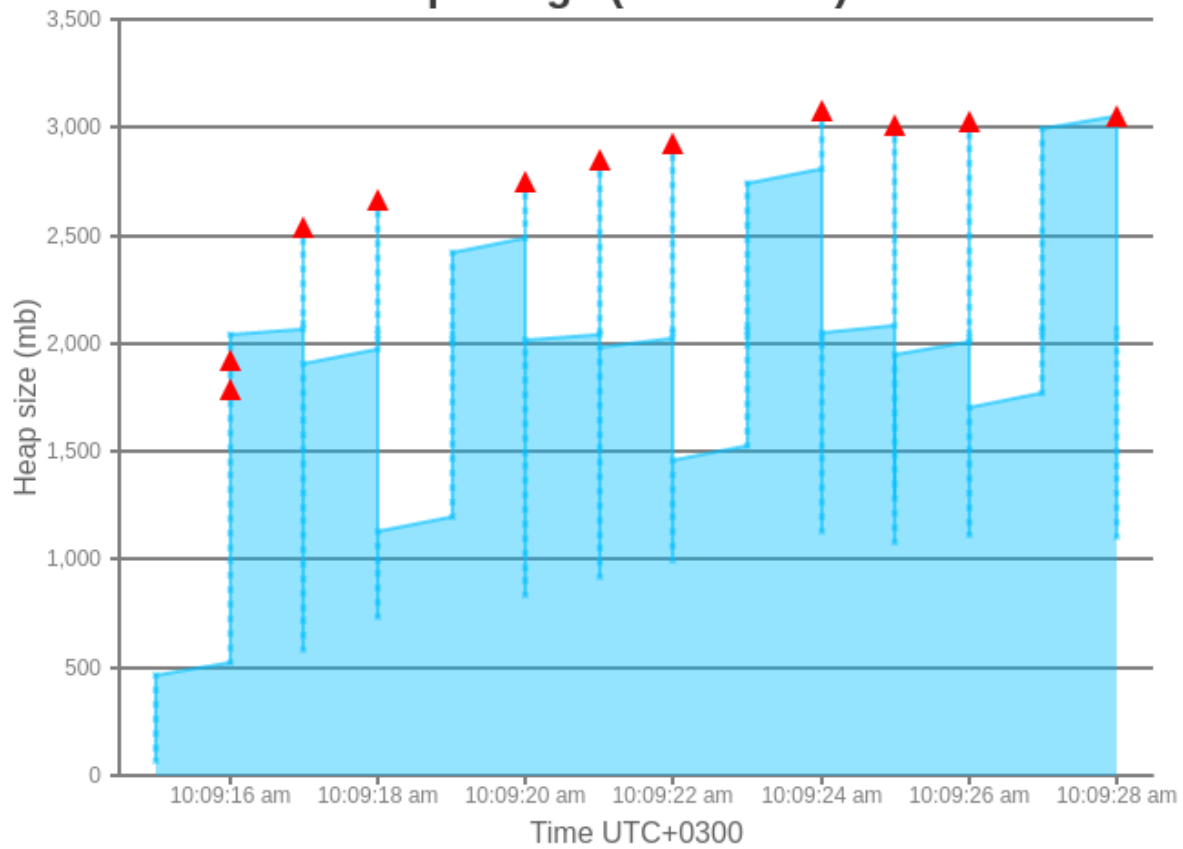


Interactive Graphs

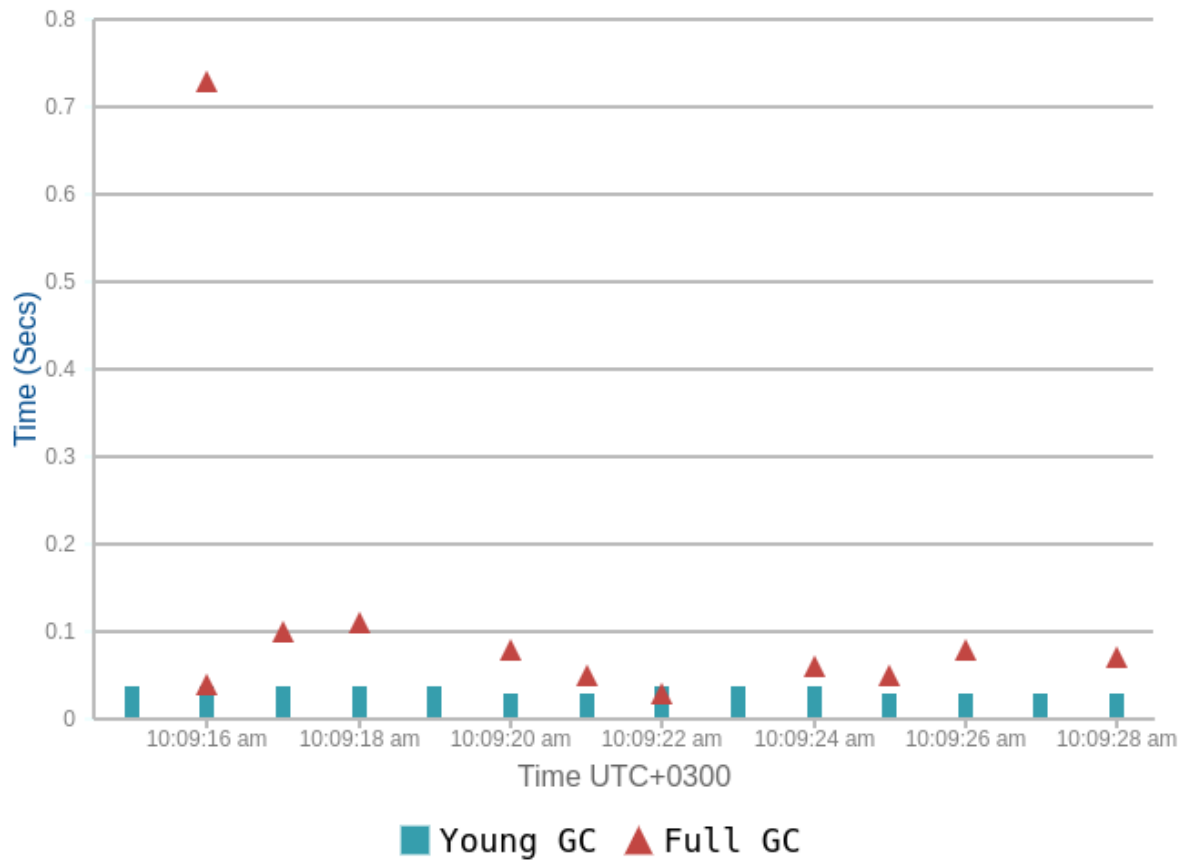
(All graphs are zoomable)

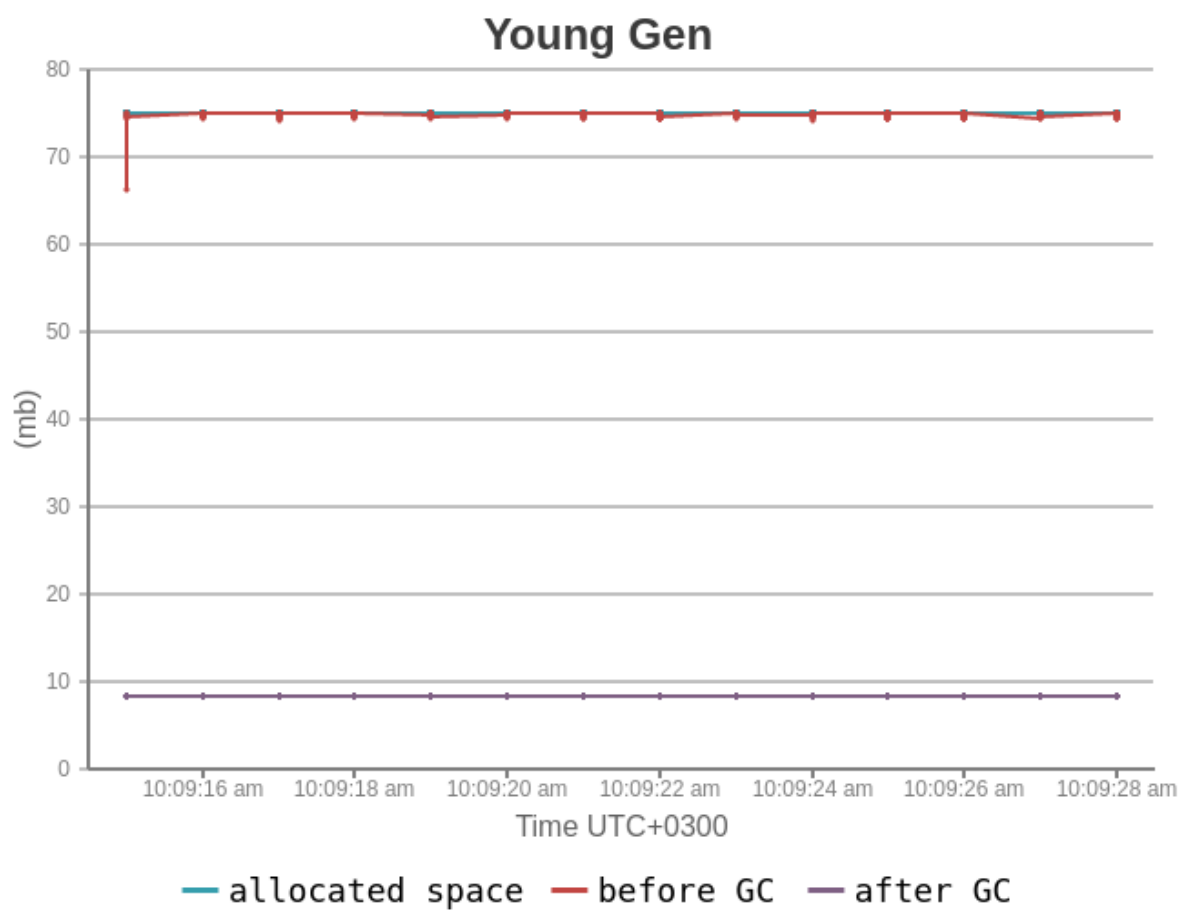
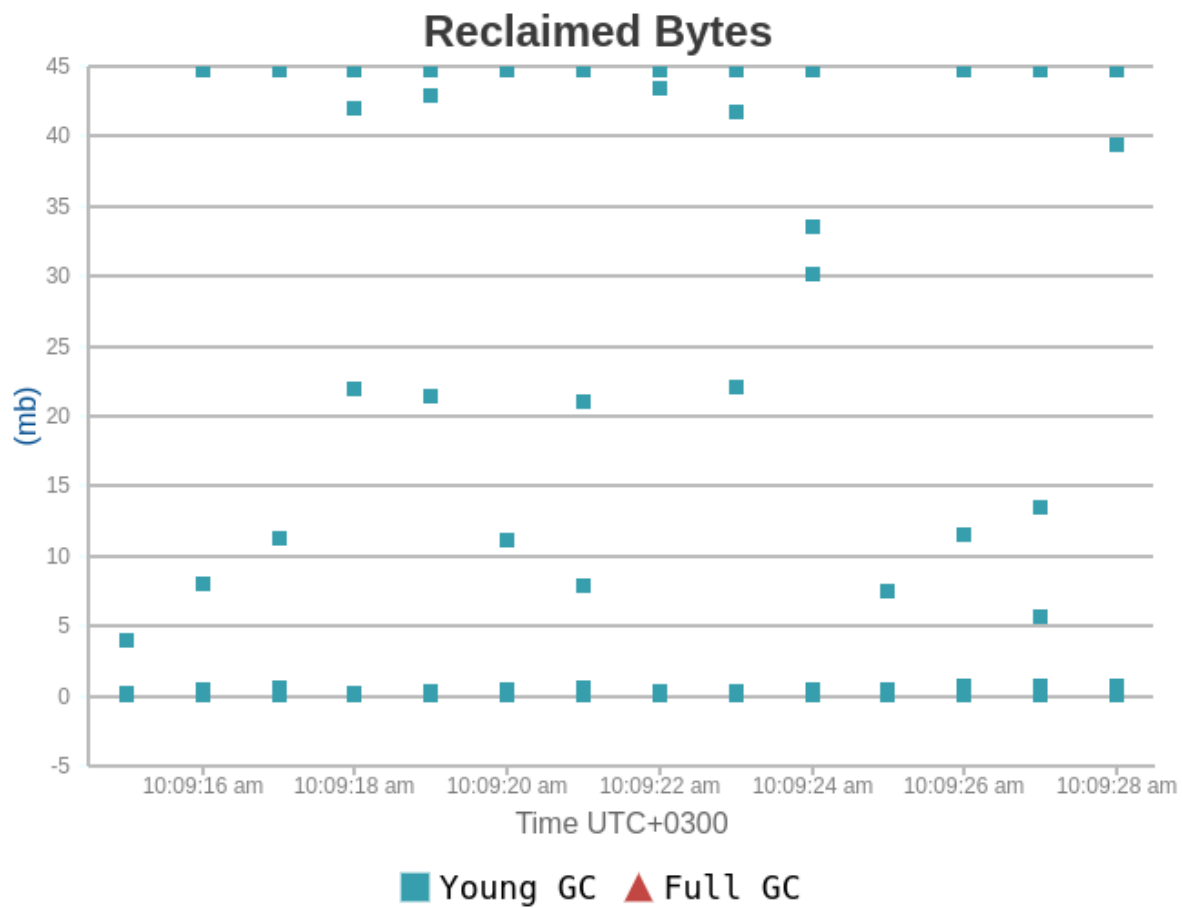


Heap Usage (before GC)

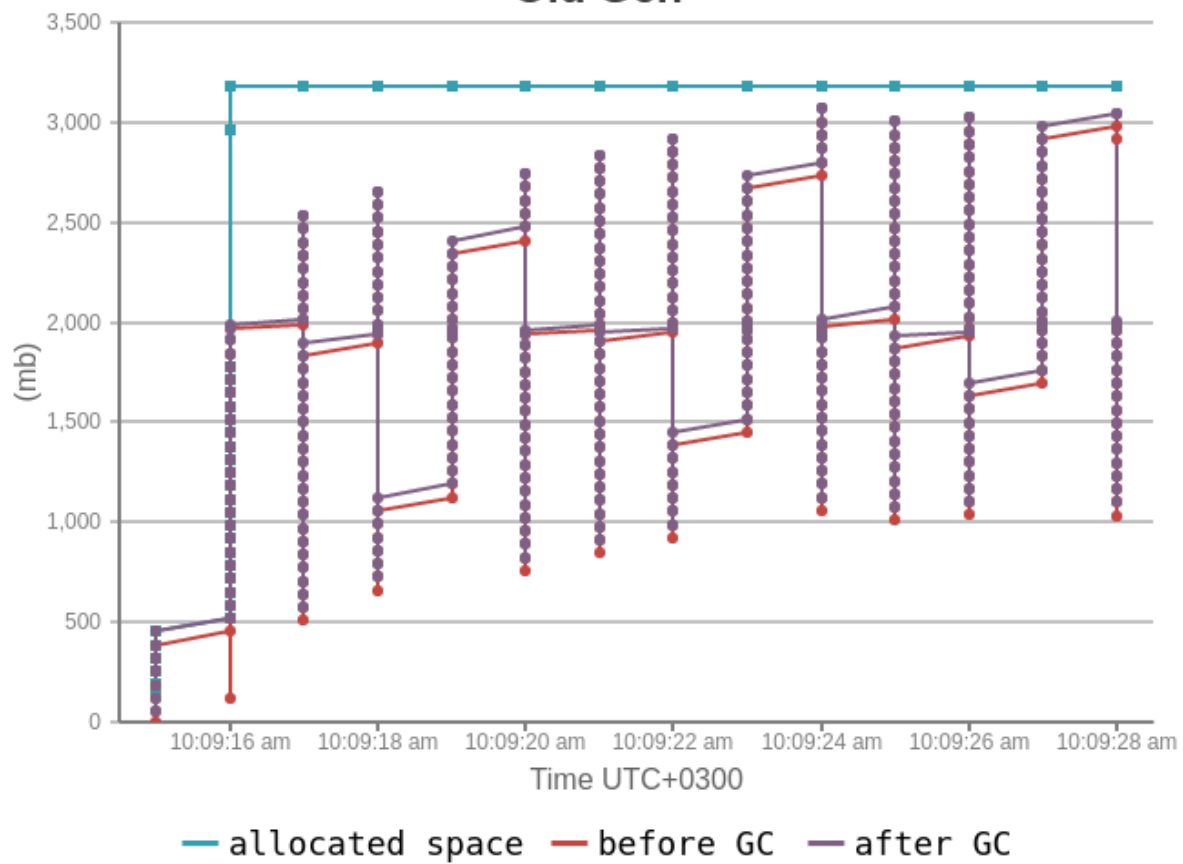


GC Duration Time

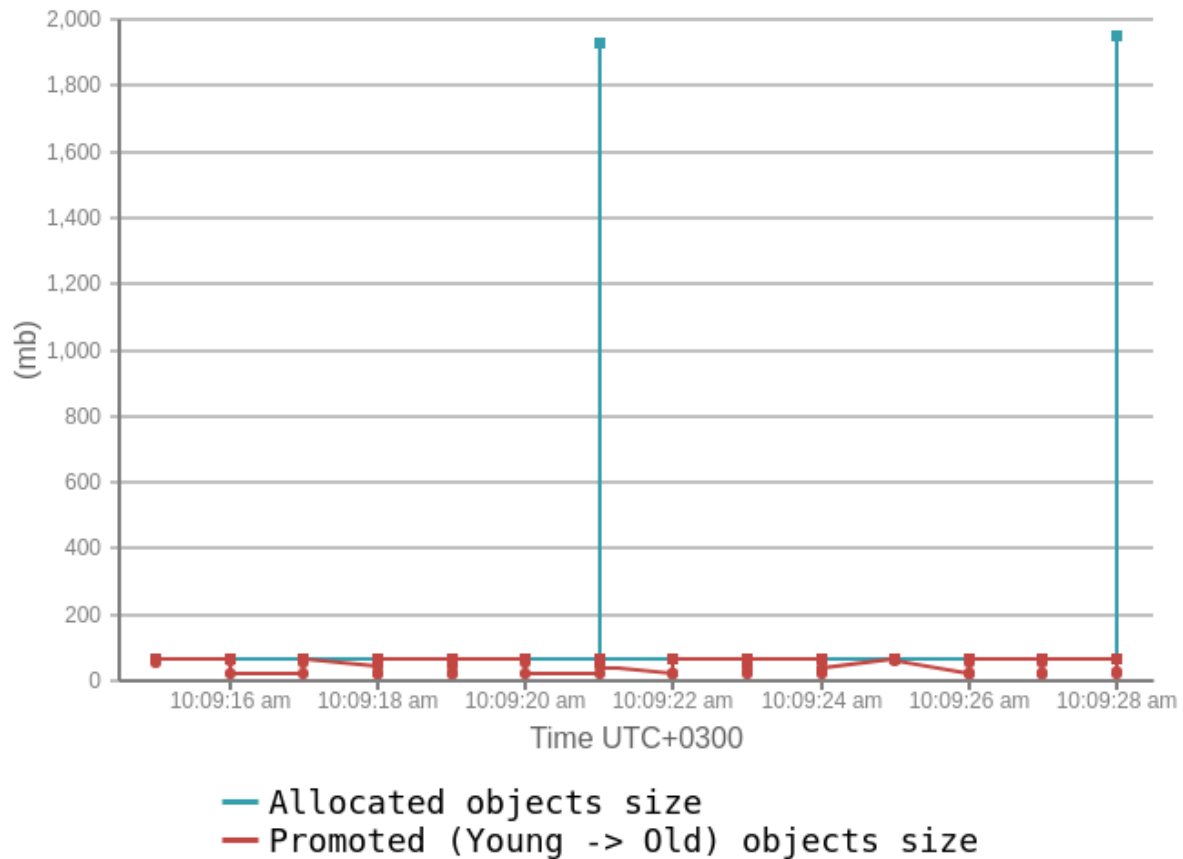




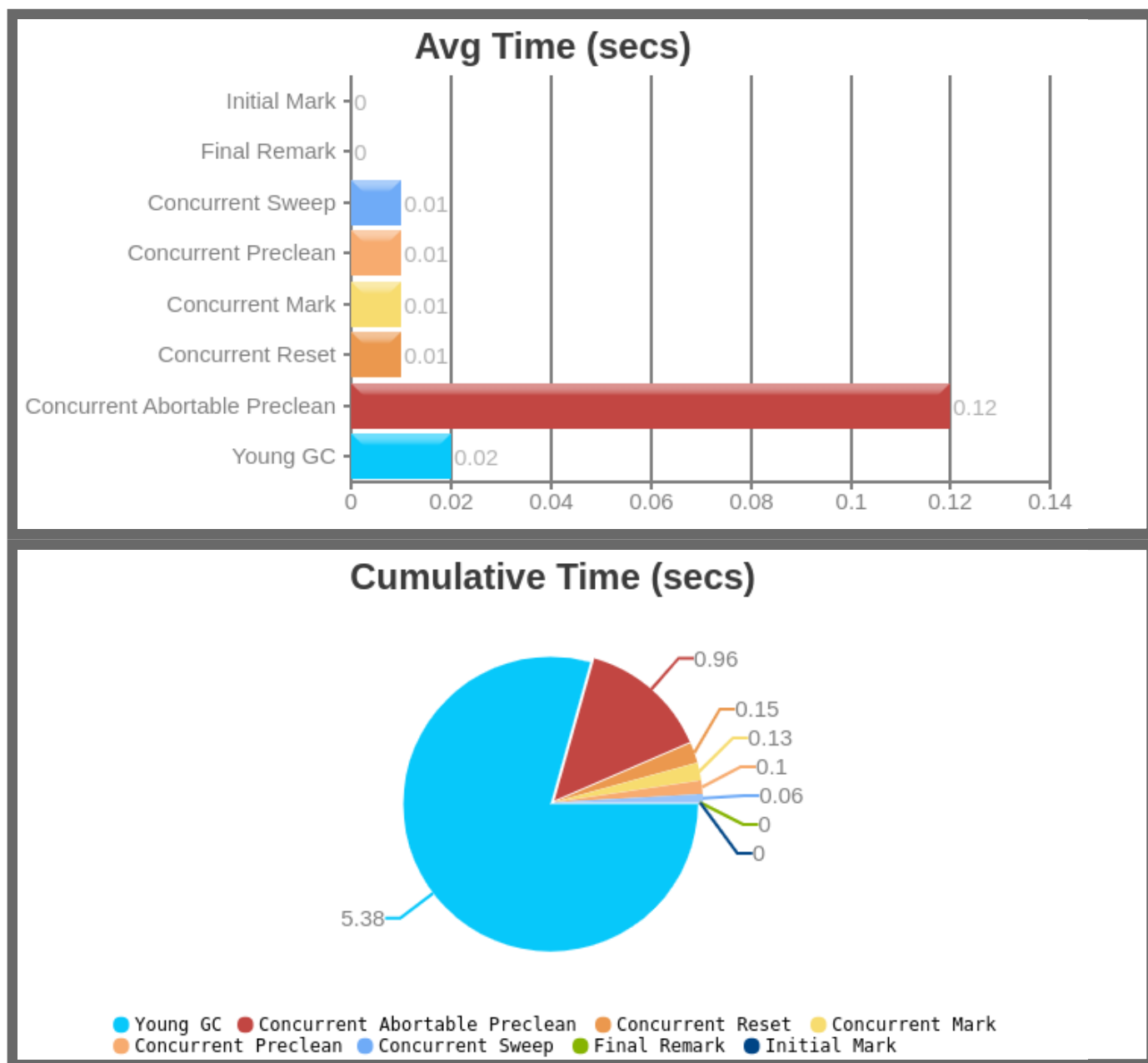
Old Gen



Allocation & Promotion



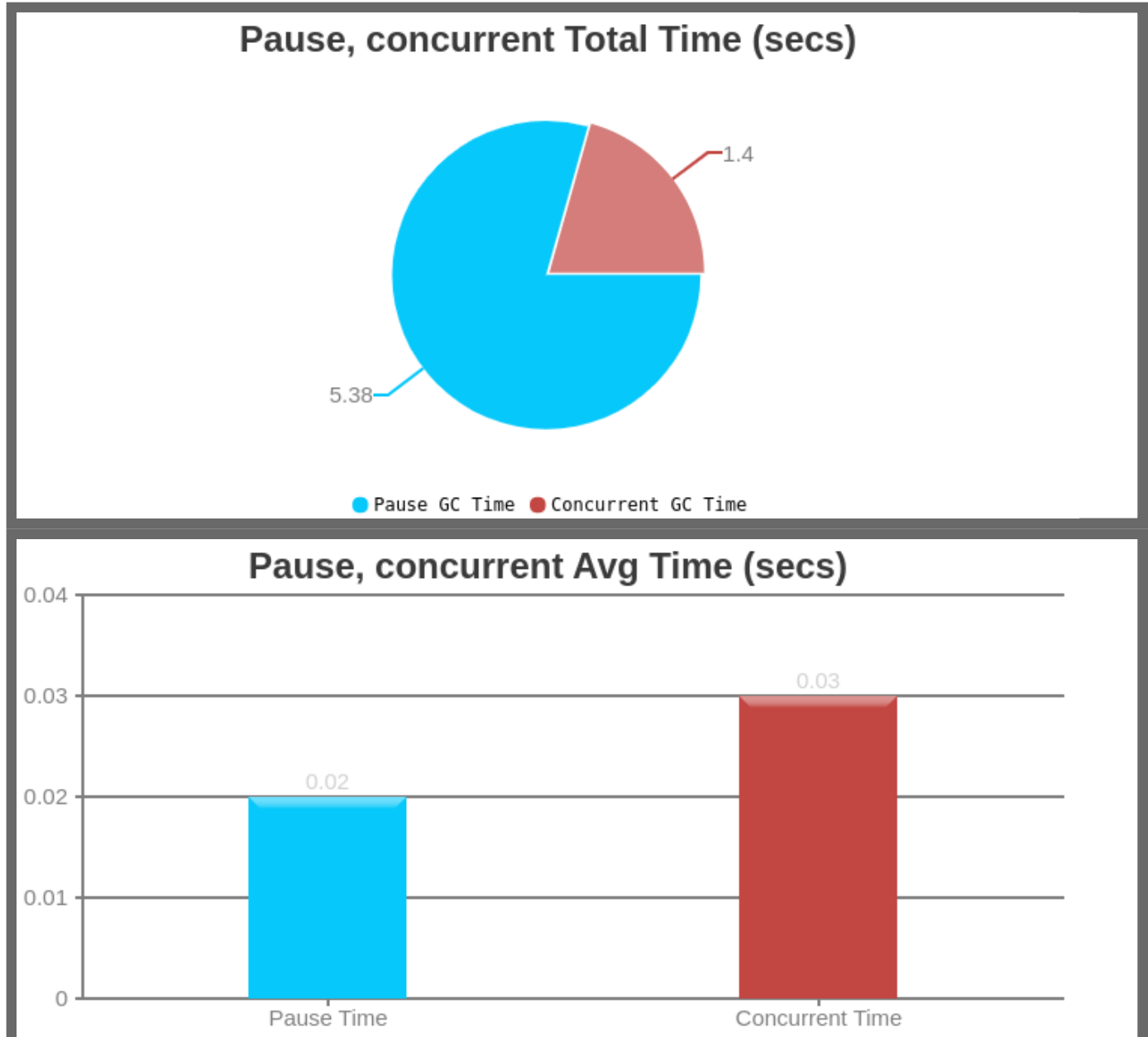
🔗 CMS Collection Phases Statistics



	Young GC 🕒	Concurrent Abortable Preclean	Concurrent Reset	Concurrent Mark	Concurrent Preclean	Concurrent Sweep	Final Remark 🕒	Initial Mark 🕒
Total Time 🕒	5 sec 380 ms	960 ms	150 ms	130 ms	100 ms	60 ms	0	0
Avg Time 🕒	17 ms	120 ms	14 ms	12 ms	9 ms	5 ms	0	0

Std Dev Time	6 ms	224 ms	9 ms	7 ms	5 ms	5 ms	0	0
Min Time ?	10 ms	20 ms	0	0	0	0	0	0
Max Time ?	30 ms	710 ms	30 ms	30 ms	20 ms	10 ms	0	0
Count ?	318	8	11	11	11	11	11	11

🔍 CMS GC Time



Pause Time ?

Total Time	5 sec 380 ms
Avg Time	16 ms
Std Dev Time	7 ms
Min Time	0
Max Time	30 ms

Concurrent Time ?

Total Time	1 sec 400 ms
Avg Time	27 ms
Std Dev Time	97 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 ?	23.71 gb
Total promoted bytes ?	19.55 gb
Avg creation rate ?	1.83 gb/sec
Avg promotion rate ?	1.51 gb/sec

💧 Memory Leak ?

No major memory leaks.

(**Note:** there are [8 flavours of OutOfMemoryErrors](https://tier1app.files.wordpress.com/2014/12/outofmemoryerror2.pdf) (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/) (https://blog.gceasy.io/2016/12/22/total-time-for-which-application-threads-were-stopped/))

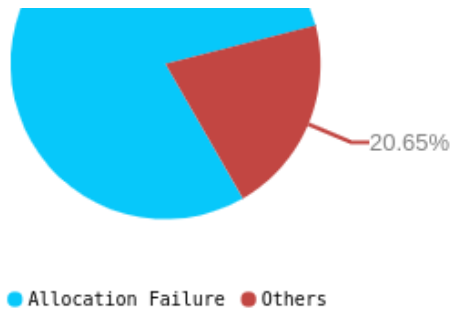
Not Reported in the log.

GC Causes

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

Cause	Count	Avg Time	Max Time	Total Time	Time %
Allocation Failure	318	17 ms	30 ms	5 sec 380 ms	79.35%
Others	11	n/a	n/a	1 sec 400 ms	20.65%
Total	329	n/a	n/a	6 sec 780 ms	100.0%





⌘ Tenuring Summary ⓘ

Not reported in the log.

📄 Command Line Flags ⓘ

-XX:ConcGCThreads=4 -XX:InitialHeapSize=260849920 -XX:MaxHeapSize=4173598720 -XX:MaxNewSize=348966912 -XX:MaxTenuringThreshold=6 -XX:OldPLABSize=16 -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:+UseParNewGC

