

# GC Intelligence Report

 eclipse-jee-gc.log

 Duration: 15 min 8 sec 817 ms


 [Download](#)


()

 [Share Report](#)

## Learn key sections

(<https://www.youtube.com/watch?v=dN7S1RoKNYo>)

 Recommendations

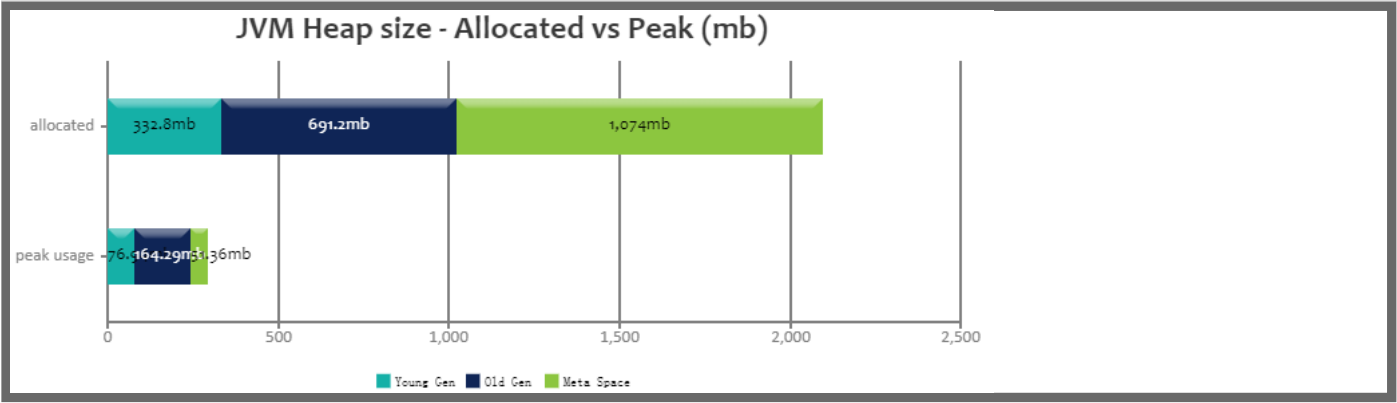


Our Machine Learning algorithms has identified memory optimization recommendations for your application. To see those recommendations

Subscribe (gc-subscription.jsp)

## JVM Heap Size

Generation	Allocated ⓘ	Peak ⓘ
Young Generation	332.8 mb	76.94 mb
Old Generation	691.2 mb	164.29 mb
Meta Space	1.05 gb	51.36 mb
Young + Old + Meta space	2.05 gb	277.49 mb



## 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/>))

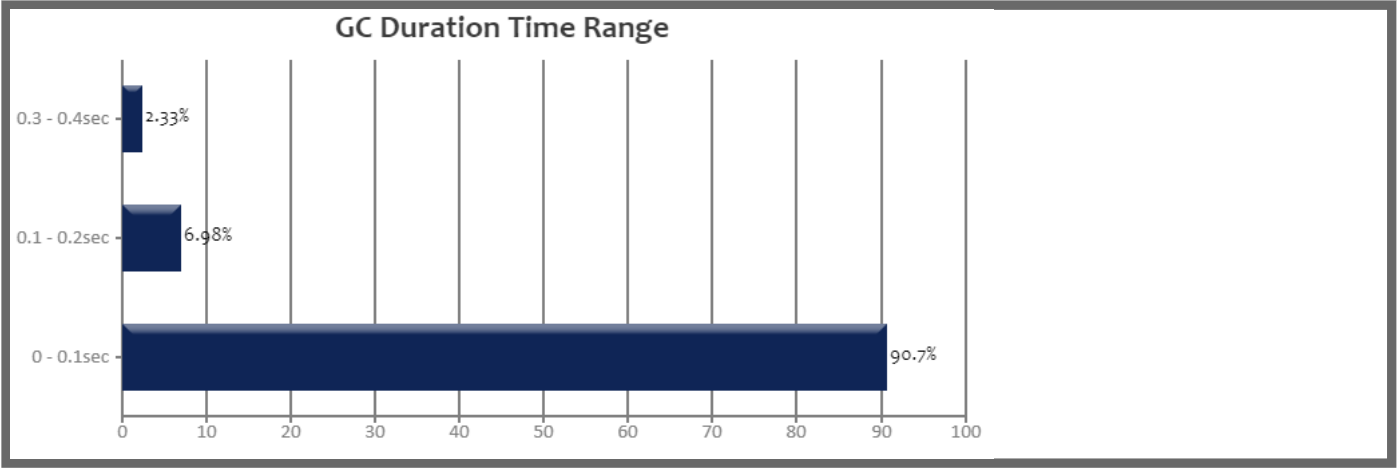
1 Throughput 📉 : 99.73%

2 Latency:

Avg Pause GC Time 📉	57.0 ms
Max Pause GC Time 📉	330 ms

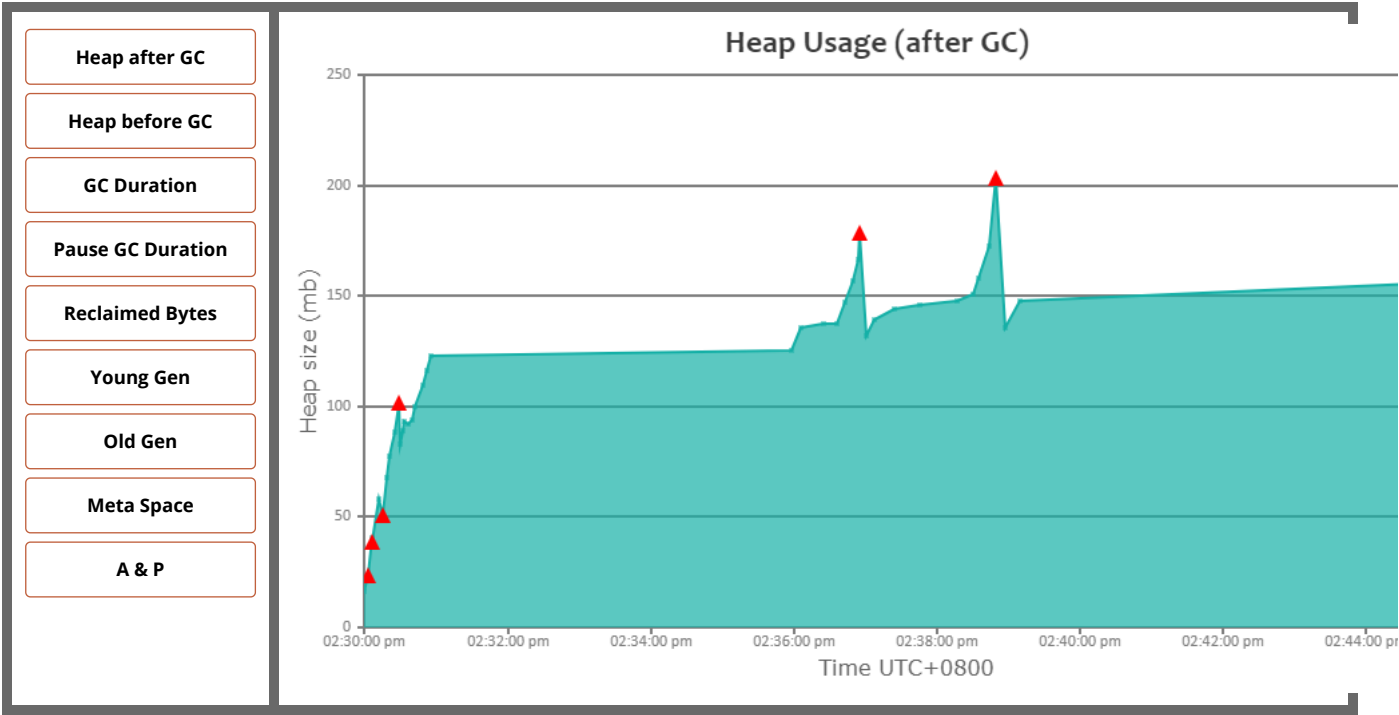
GC Pause Duration Time Range 📉:

Duration (sec)	No. of GCs	Percentage
0.1 sec		
0 - 0.1	39	90.7%
0.1 - 0.2	3	6.98%
0.3 - 0.4	1	2.33%

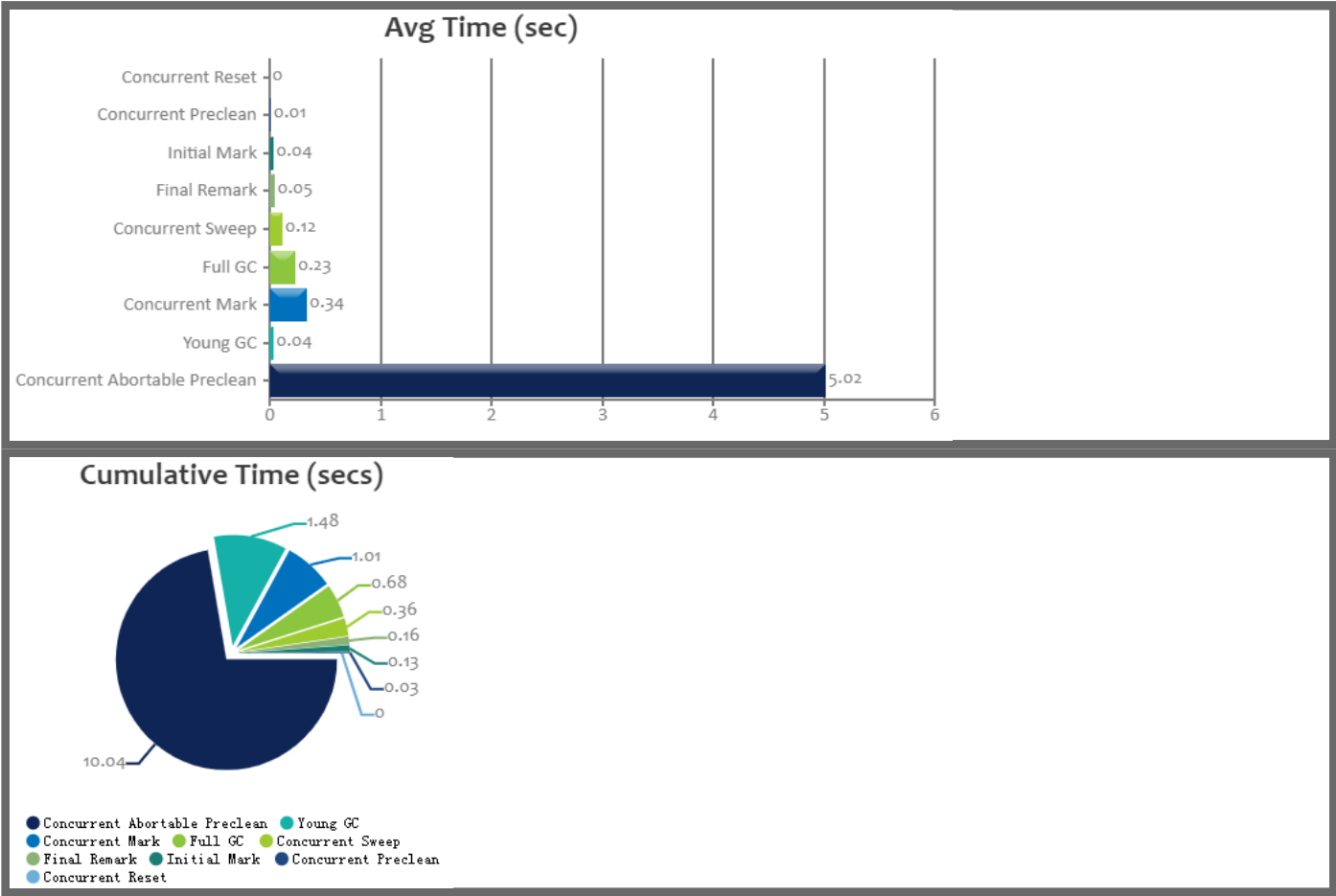


## Interactive Graphs

(All graphs are zoomable)

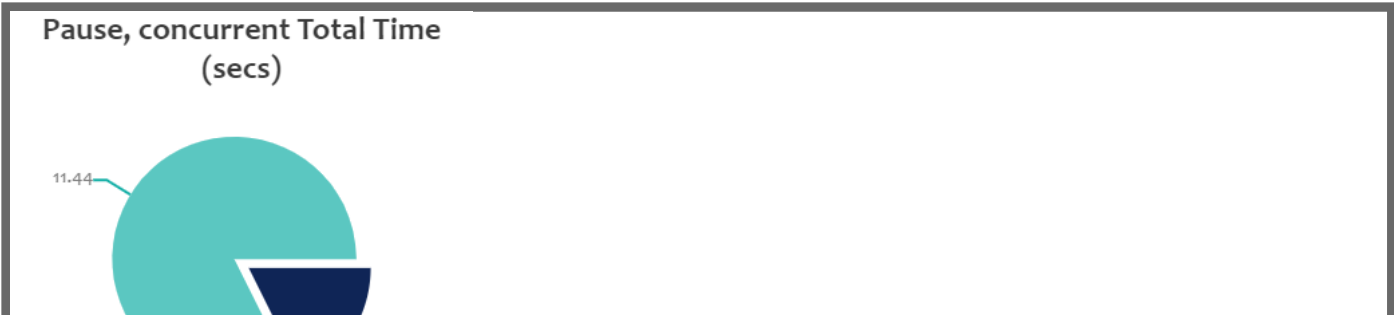


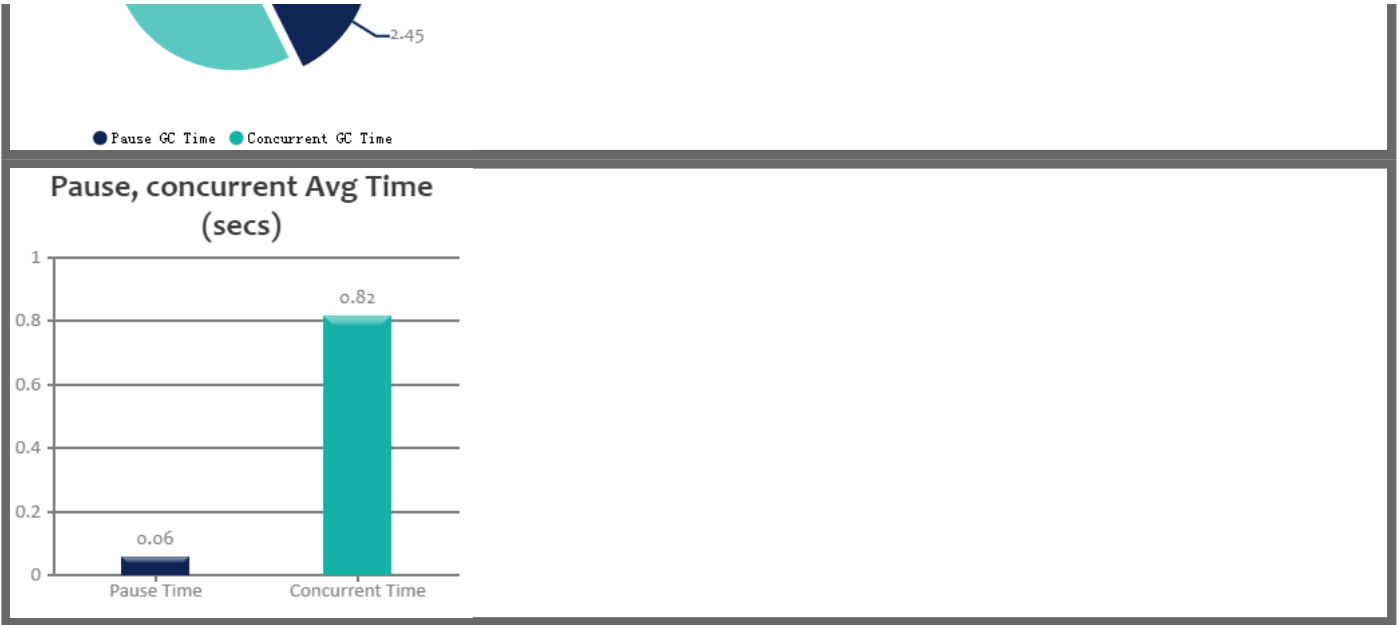
# CMS Collection Phases Statistics



	Concurrent Abortable Preclean	Young GC	Concurrent Mark	Full GC	Concurrent Sweep	Final Remark	Initial Mark	Concurrent Preclean	Concurrent Reset
Total Time	10 sec 40 ms	1 sec 480 ms	1 sec 10 ms	680 ms	360 ms	160 ms	130 ms	30.0 ms	0
Avg Time	5 sec 20 ms	43.5 ms	337 ms	227 ms	120 ms	53.3 ms	43.3 ms	10.0 ms	0
Std Dev Time	20.0 ms	24.1 ms	145 ms	73.2 ms	53.5 ms	17.0 ms	18.9 ms	8.16 ms	0
Min Time	5 sec	20.0 ms	180 ms	170 ms	50.0 ms	30.0 ms	30.0 ms	0	0
Max Time	5 sec 40 ms	140 ms	530 ms	330 ms	180 ms	70.0 ms	70.0 ms	20.0 ms	0
Count	2	34	3	3	3	3	3	3	3

## CMS GC Time





Pause Time ?

Total Time	2 sec 450 ms
Avg Time	57.0 ms
Std Dev Time	55.2 ms
Min Time	20.0 ms
Max Time	330 ms

Concurrent Time ?

Total Time	11 sec 440 ms
Avg Time	817 ms
Std Dev Time	1 sec 722 ms
Min Time	0
Max Time	5 sec 40 ms

⚙️ Object Stats

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

Total created bytes ?	2.23 gb
Total promoted bytes ?	237.14 mb
Avg creation rate ?	2.51 mb/sec
Avg promotion rate ?	267 kb/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 %
Others	3	n/a	n/a	11 sec 730 ms	84.45%
Allocation Failure ⓘ	34	43.5 ms	140 ms	1 sec 480 ms	10.66%
Metadata GC Threshold ⓘ	3	227 ms	330 ms	680 ms	4.9%
Total	40	n/a	n/a	13 sec 890 ms	100.01%



## ⚡ Tenuring Summary ⓘ

Not reported in the log.

## 📄 Command Line Flags ⓘ

-XX:-BytecodeVerificationLocal -XX:-BytecodeVerificationRemote -XX:-CMSClassUnloadingEnabled -XX:CMSInitiatingOccupancyFraction=85 -XX:-ClassUnloading -XX:+DisableExplicitGC -XX:-ExplicitGCInvokesConcurrentAndUnloadsClasses -XX:InitialHeapSize=268435456 -XX:MaxHeapSize=1073741824 -XX:MaxNewSize=348966912 -XX:MaxTenuringThreshold=6 -XX:OldPLABSize=16 -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -

XX:+PrintGCTimeStamps -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:-UseGCOverheadLimit -XX:-UseLargePagesIndividualAllocation -XX:+UseParNewGC -XX:+UseStringDeduplication

## 🏆 Become a DevOps champion in your organization

(Best practises/tools)

- ✓ Use **fastThread.io** (<https://fastthread.io/>) tool to analyze thread dumps, core dumps and hs\_err\_pid files
- ✓ Do proactive Garbage Collection analysis on all your JVMs (not just one or two) [using the GC log analysis API](https://blog.gceasy.io/2016/06/18/garbage-collection-log-analysis-api/) (https://blog.gceasy.io/2016/06/18/garbage-collection-log-analysis-api/)
- ✓ Always [rotate the GC log files](https://blog.gceasy.io/2016/11/15/rotating-gc-log-files/) (https://blog.gceasy.io/2016/11/15/rotating-gc-log-files/) to prevent critical garbage collection data loss
- ✓ Use 'Enterprise' edition (<http://gceasy.io/pricing.jsp>) for 10x fast, unlimited, secure usage

## Do you like this report?



### GCEasy

GCEasy is the industry's first online Garbage collection log analysis tool aided by Machine Learning.

It's used by thousands of enterprises globally to tune & troubleshoot complex memory & GC problems.

### Reach Us

📍 Dublin, CA, USA

☎ +1-415-948-5431

✉ [team@tier1app.com](mailto:team@tier1app.com) (mailto:team@tier1app.com)

### Quick Links

- [Terms & Conditions \(terms.jsp\)](#)
- **fastThread** (sister product) (<https://fastthread.io/>)
- **HeapHero** (sister product) (<https://heaphero.io/>)

### Stay in Touch!

Follow us on social networks!

📘 (<https://www.facebook.com/tier1app>) 🐦 (<https://twitter.com/tier1app>) 💼 (<https://www.linkedin.com/company/gceasy>)

© Tier1App. All Right Reserved

Made by [Tier1app](http://tier1app.com) (<http://tier1app.com>) with 🍷 + soul + intelligence