GC Tuning & Troubleshooting Crash Course

Ram Lakshmanan

Architect: Gceasy.io, fastThread.io, HeapHero.io

What is garbage?

Java 11+:
How many Garbage Collectors?

12

HotSpot

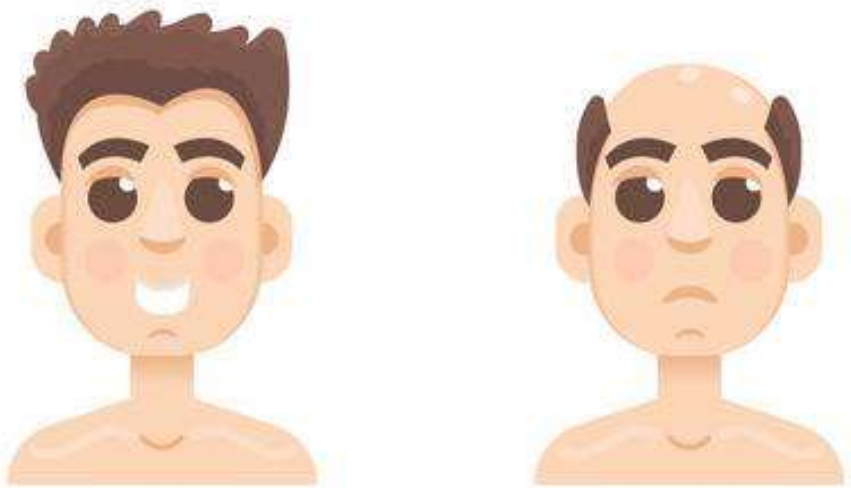
1. Serial
2. Parallel
3. CMS
4. G1
5. Shenandoah
6. Z GC
7. Epsilon

OpenJ9

1. Balanced
2. Concurrent
3. Mentronome
4. Pause Time
5. Throughput
6. Epsilon

How many GC/Memory related JVM arguments are available?

600+



Trying to understand GC

Overwhelming?

- **You are not alone!**

Don’t Fear!
Simple techniques

Key Performance Indicators

You can't optimize, what you can't measure

Avg Pause GC Time	34.7 ms
Max Pause GC Time	990 ms

Duration (ms) 200 ms Change	No. of GCs	Percentage
0 - 200	769	99.23%
200 - 400	1	0.13%
400 - 600	2	0.26%
600 - 800	1	0.13%
800 - 1,000	2	0.26%

99.994%

Memory: 2GB
CPU: 45%

1. Latency

GC Event's Pause Time

2. Throughput

Percentage of time spent in processing customer transactions vs time spent in GC activity. i.e. productive work vs non-productive work

3. Footprint

Memory and CPU consumption of the application

.

How to source these KPIs?

Till Java 8

-XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:<file-path>

Enable GC logs (always)

From Java 9

-Xlog:gc*:file=<file-path>

Vanilla Format

2016-08-31T01:09:19.397+0000: 1.606: [GC (Metadata GC Threshold) [PSYoungGen: 545393K->18495K(2446848K)] 545393K->18519K(8039424K), 0.0189376 secs] [Times: user=0.15 sys=0.01, real=0.02 secs]

2016-08-31T01:09:19.416+0000: 1.625: [Full GC (Metadata GC Threshold) [PSYoungGen: 18495K->0K(2446848K)] [ParOldGen: 24K->17366K(5592576K)] 18519K->17366K(8039424K), [Metaspace: 20781K->20781K(1067008K)], 0.0416162 secs] [Times: user=0.38 sys=0.03, real=0.04 secs]

2016-08-31T01:18:19.288+0000: 541.497: [GC (Metadata GC Threshold) [PSYoungGen: 1391495K->18847K(2446848K)] 1408861K->36230K(8039424K), 0.0568365 secs] [Times: user=0.31 sys=0.75, real=0.06 secs]

2016-08-31T01:18:19.345+0000: 541.554: [Full GC (Metadata GC Threshold) [PSYoungGen: 18847K->0K(2446848K)] [ParOldGen: 17382K->25397K(5592576K)] 36230K->25397K(8039424K), [Metaspace: 34865K->34865K(1079296K)], 0.0467640 secs] [Times: user=0.31 sys=0.08, real=0.04 secs]

2016-08-31T02:33:20.326+0000: 5042.536: [GC (Allocation Failure) [PSYoungGen: 2097664K->11337K(2446848K)] 2123061K->36742K(8039424K), 0.3298985 secs] [Times: user=0.00 sys=9.20, real=0.33 secs]

2016-08-31T03:40:11.749+0000: 9053.959: [GC (Allocation Failure) [PSYoungGen: 2109001K->15776K(2446848K)] 2134406K->41189K(8039424K), 0.0517517 secs] [Times: user=0.00 sys=1.22, real=0.05 secs]

2016-08-31T05:11:46.869+0000: 14549.079: [GC (Allocation Failure) [PSYoungGen: 2113440K->24832K(2446848K)] 2138853K->50253K(8039424K), 0.0392831 secs] [Times: user=0.02 sys=0.79, real=0.04 secs]

2016-08-31T06:26:10.376+0000: 19012.586: [GC (Allocation Failure) [PSYoungGen: 2122496K->25600K(2756096K)] 2147917K->58149K(8348672K), 0.0371416 secs] [Times: user=0.01 sys=0.75, real=0.04 secs]

2016-08-31T07:50:03.442+0000: 24045.652: [GC (Allocation Failure) [PSYoungGen: 2756096K->32768K(2763264K)] 2788645K->72397K(8355840K), 0.0709641 secs] [Times: user=0.16 sys=1.39, real=0.07 secs]

2016-08-31T09:04:21.406+0000: 28503.616: [GC (Allocation Failure) [PSYoungGen: 2763264K->32768K(2733568K)] 2802893K->83469K(8326144K), 0.0789178 secs] [Times: user=0.12 sys=1.59, real=0.08 secs]

GC Log Format varies

JVM Vendor
Oracle
HP
IBM
Azul
OpenJDK
…

Java Version
1.4
5
6
7
8
9
10
11
12

GC algorithm
Serial
Parallel
CMS
G1
Shennandoh
Z GC

Arguments
-XX:+PrintGC
-XX:+PrintGCDateStamps
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+PrintPromotionFailure
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintClassHistogram
-XX:PrintFLSStatistics=1
-XX:+PrintCodeCache

G1 GC Format

```
2015-09-14T11:58:55.131-0700: 0.519: [GC pause (G1 Evacuation Pause) (young), 0.0096438 secs]
   [Parallel Time: 7.9 ms, GC Workers: 8]
      [GC Worker Start (ms): Min: 519.4, Avg: 519.6, Max: 520.6, Diff: 1.3]
      [Ext Root Scanning (ms): Min: 0.0, Avg: 2.9, Max: 7.3, Diff: 7.3, Sum: 23.4]
      [Update RS (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
         [Processed Buffers: Min: 0, Avg: 0.0, Max: 0, Diff: 0, Sum: 0]
      [Scan RS (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
      [Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.1, Diff: 0.1, Sum: 0.1]
      [Object Copy (ms): Min: 0.0, Avg: 4.2, Max: 7.2, Diff: 7.2, Sum: 34.0]
      [Termination (ms): Min: 0.0, Avg: 0.2, Max: 0.4, Diff: 0.4, Sum: 1.7]
         [Termination Attempts: Min: 1, Avg: 7.9, Max: 18, Diff: 17, Sum: 63]
      [GC Worker Other (ms): Min: 0.0, Avg: 0.0, Max: 0.1, Diff: 0.1, Sum: 0.4]
      [GC Worker Total (ms): Min: 6.4, Avg: 7.4, Max: 7.7, Diff: 1.3, Sum: 59.6]
      [GC Worker End (ms): Min: 527.0, Avg: 527.1, Max: 527.1, Diff: 0.1]
   [Code Root Fixup: 0.0 ms]
   [Code Root Purge: 0.0 ms]
   [Clear CT: 0.5 ms]
   [Other: 1.3 ms]
      [Choose CSet: 0.0 ms]
      [Ref Proc: 0.7 ms]
      [Ref Enq: 0.0 ms]
      [Redirty Cards: 0.3 ms]
      [Humongous Register: 0.0 ms]
      [Humongous Reclaim: 0.0 ms]
      [Free CSet: 0.0 ms]
   [Eden: 24.0M(24.0M)->0.0B(34.0M) Survivors: 0.0B->3072.0K Heap: 24.0M(252.0M)->3338.0K(252.0M)]
 [Times: user=0.06 sys=0.00, real=0.01 secs]
```

CMS GC Format

Before GC:

Statistics for BinaryTreeDictionary:

Total Free Space: 2524251

Max Chunk Size: 2519552

Number of Blocks: 13

Av. Block Size: 194173

Tree Height: 8

2016-05-03T04:27:37.503+0000: 30282.678: [ParNew

Desired survivor size 214728704 bytes, new threshold 1 (max 1)

- age 1: 85782640 bytes, 85782640 total

: 3510063K->100856K(3774912K), 0.0516290 secs] 9371816K->6022161K(14260672K)After GC:

Statistics for BinaryTreeDictionary:

Total Free Space: 530579346

Max Chunk Size: 332576815

Number of Blocks: 7178

Av. Block Size: 73917

Tree Height: 44

After GC:

Statistics for BinaryTreeDictionary:

Total Free Space: 2524251

Max Chunk Size: 2519552

Number of Blocks: 13

Av. Block Size: 194173

Tree Height: 8

, 0.0552970 secs] [Times: user=0.67 sys=0.00, real=0.06 secs]

IBM GC Format

```
<af type="tenured" id="4" timestamp="Jun 16 11:28:22 2016" intervalms="5633.039">
  <minimum requested_bytes="56" />
  <time exclusiveaccessms="0.010" meanexclusiveaccessms="0.010" threads="0" lastthreadtid="0xF6B1C400" />
  <refs soft="7232" weak="3502" phantom="9" dynamicSoftReferenceThreshold="30" maxSoftReferenceThreshold="32" />
  <tenured freebytes="42949632" totalbytes="1073741824" percent="3" >
    <soa freebytes="0" totalbytes="1030792192" percent="0" />
    <loa freebytes="42949632" totalbytes="42949632" percent="100" />
  </tenured>
  <pending-finalizers finalizable="0" reference="0" classloader="0" />
  <gc type="global" id="6" totalid="6" intervalms="3342.687">
    <classunloading classloaders="0" classes="0" timevmquiescems="0.000" timetakenms="1.200" />
    <finalization objectsqueued="75" />
    <timesms mark="28.886" sweep="1.414" compact="0.000" total="31.571" />
    <tenured freebytes="1014673616" totalbytes="1073741824" percent="94" >
      <soa freebytes="982461648" totalbytes="1041529856" percent="94" />
      <loa freebytes="32211968" totalbytes="32211968" percent="100" />
    </tenured>
  </gc>
  <tenured freebytes="1014608080" totalbytes="1073741824" percent="94" >
    <soa freebytes="982396112" totalbytes="1041529856" percent="94" />
    <loa freebytes="32211968" totalbytes="32211968" percent="100" />
  </tenured>
  <refs soft="7020" weak="2886" phantom="9" dynamicSoftReferenceThreshold="30" maxSoftReferenceThreshold="32" />
  <pending-finalizers finalizable="75" reference="15" classloader="0" />
  <time totalms="33.852" />
</af>
```

IBM GC another Format

```
<gc-op id="139" type="scavenge" timems="335.610" contextid="136"
timestamp="2016-06-15T15:51:10.128">
  <scavenger-info tenureage="4" tenuremask="7ff0" tiltratio="58" />
  <memory-copied type="nursery" objects="11071048" bytes="448013400"
bytesdiscarded="88016" />
  <memory-copied type="tenure" objects="286673" bytes="9771936"
bytesdiscarded="320608" />
  <copy-failed type="nursery" objects="286673" bytes="9771936" />
  <finalization candidates="112" enqueued="16" />
  <ownableSynchronizers candidates="8111" cleared="11" />
  <references type="soft" candidates="1256" cleared="0" enqueued="0"
dynamicThreshold="32" maxThreshold="32" />
  <references type="weak" candidates="2953" cleared="0" enqueued="0" />
  <references type="phantom" candidates="142406" cleared="142335"
enqueued="142335" />
</gc-op>
```

Android Dalvik GC Format

07-01 15:56:20.035: I/Choreographer(30615): Skipped 141 frames! The application may be doing too much work on its main thread.

07-01 15:56:20.275: D/dalvikvm(30615): GC_FOR_ALLOC freed 4774K, 45% free 49801K/89052K, paused 168ms, total 168ms

07-01 15:56:20.295: I/dalvikvm-heap(30615): Grow heap (frag case) to 56.900MB for 4665616-byte allocation

07-01 15:56:21.315: D/dalvikvm(30615): GC_FOR_ALLOC freed 1359K, 42% free 55045K/93612K, paused 95ms, total 95ms

07-01 15:56:21.965: D/dalvikvm(30615): GC_CONCURRENT freed 6376K, 40% free 56861K/93612K, paused 16ms+8ms, total 126ms

07-01 15:56:21.965: D/dalvikvm(30615): WAIT_FOR_CONCURRENT_GC blocked 111ms

07-01 15:56:21.965: D/dalvikvm(30615): WAIT_FOR_CONCURRENT_GC blocked 97ms

Android ART GC Format

07-01 16:00:44.690: I/art(801): Explicit concurrent mark sweep GC freed 65595(3MB) AllocSpace objects, 9(4MB) LOS objects, 810% free, 38MB/58MB, paused 1.195ms total 87.219ms

07-01 16:00:46.517: I/art(29197): Background partial concurrent mark sweep GC freed 74626(3MB) AllocSpace objects, 39(4MB) LOS objects, 1496% free, 25MB/32MB, paused 4.422ms total 1.371747s

07-01 16:00:48.534: I/Choreographer(29197): Skipped 30 frames! The application may be doing too much work on its main thread.

07-01 16:00:48.566: I/art(29197): Background sticky concurrent mark sweep GC freed 70319(3MB) AllocSpace objects, 59(5MB) LOS objects, 825% free, 49MB/56MB, paused 6.139ms total 52.868ms

07-01 16:00:49.282: I/Choreographer(29197): Skipped 33 frames! The application may be doing too much work on its main thread.

‘Free’ GC Log analysis tools

Freely available Garbage collection log analysis tools

01

GCeasy

http://gceasy.io/

02

GC Viewer

https://github.com/chewiebug/GCViewer

03

IBM GC & Memory visualizer

https://developer.ibm.com/javasdk/tools/

04

HP Jmeter

https://h20392.www2.hpe.com/portal/swdepot/displayProductInfo.do?productNumber=HPJMETER

05

Google Garbage cat (cms)

https://code.google.com/archive/a/eclipselabs.org/p/garbagecat

GC tuning tips & tricks

https://blog.gceasy.io/2016/11/22/reduce-long-gc-pauses/

1. Start from scratch

Remove all GC JVM arguments and start tuning from scratch

Student of my training program

Presented in 2019 DevOps/Jenkins world conference

https://tinyurl.com/jwperformance

Ryan Smith
Senior Developer Support Engineer, CloudBees

Real world data from Big Bank

Key Performance Indicators

(Important section of the report. To learn more about KPIs, click here)

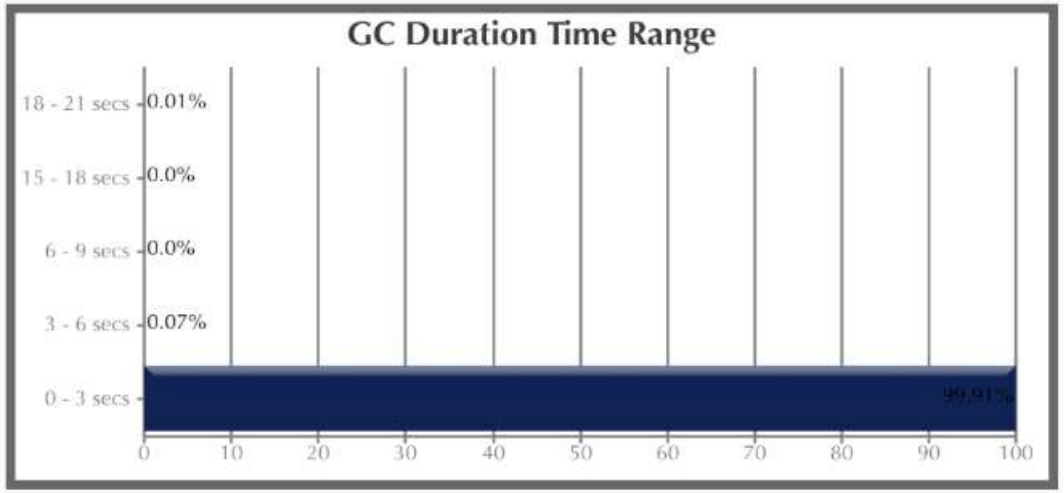
1 Throughput : 92.078%

2 Latency:

Avg Pause GC Time	265 ms
Max Pause GC Time	20 sec 600 ms

GC **Pause** Duration Time Range:

Duration (secs)	No. of GCs	Percentage
0 - 3	41742	99.911%
3 - 6	31	99.986%
6 - 9	1	99.988%
15 - 18	2	99.993%
18 - 21	3	100.0%



After removing all GC parameters

RESOLUTION: From ~92% to 99% Application Throughput

Key Performance Indicators

(Important section of the report. To learn more about KPIs, click here)

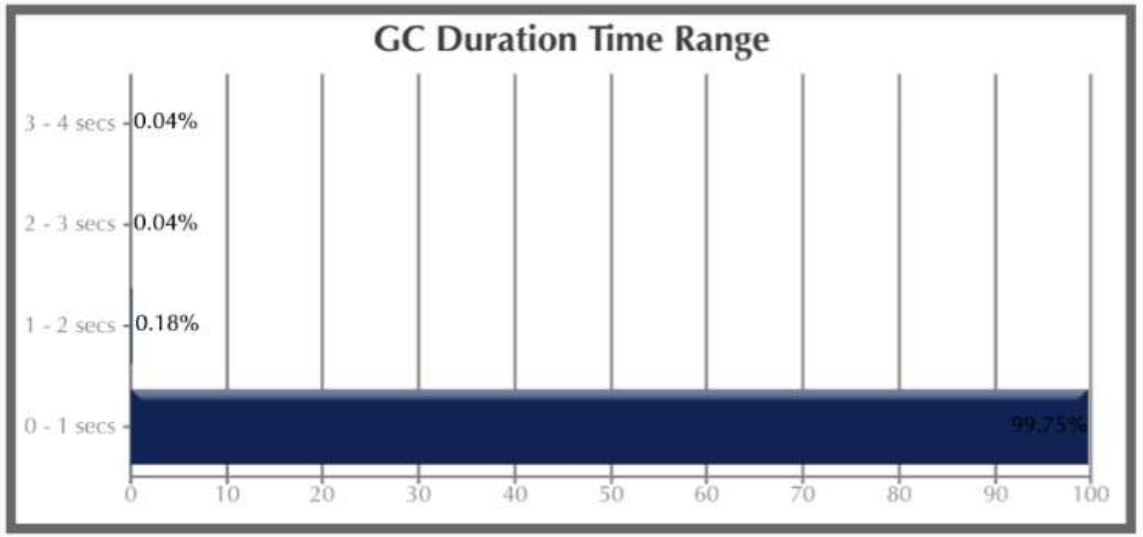
1 Throughput : 99.468%

2 Latency:

Avg Pause GC Time	127 ms
Max Pause GC Time	3 sec 250 ms

GC Pause Duration Time Range:

Duration (secs)	No. of G	Percentage
0 - 1	2839	99.754%
1 - 2	5	99.93%
2 - 3	1	99.965%
3 - 4	1	100.0%

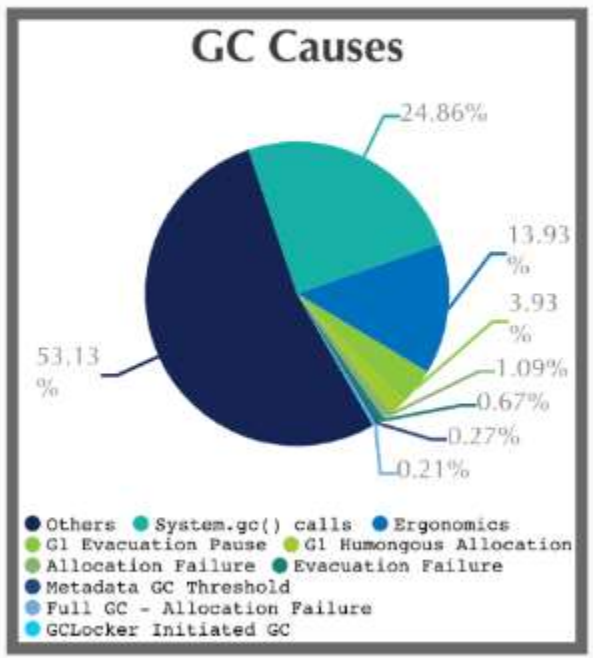


2. Study GC Causes

GC Causes

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

Cause	Count	Avg Time	Max Time	Total Time	Time %
Others	n/a	n/a	n/a	24 min 41 sec 829 ms	53.14%
System.gc() calls	104	6 sec 666 ms	11 sec 176 ms	11 min 33 sec 284 ms	24.86%
Ergonomics	12609	30.8 ms	5 sec 846 ms	6 min 28 sec 324 ms	13.93%
G1 Evacuation Pause	5896	18.6 ms	1 sec 837 ms	1 min 49 sec 667 ms	3.93%
G1 Humongous Allocation	3642	13.3 ms	2 sec 908 ms	48 sec 417 ms	1.74%
Allocation Failure	27	1 sec 124 ms	5 sec 846 ms	30 sec 354 ms	1.09%
Evacuation Failure	25	744 ms	1 sec 837 ms	18 sec 596 ms	0.67%
Metadata GC Threshold	438	16.9 ms	160 ms	7 sec 417 ms	0.27%
Full GC - Allocation Failure	1	5 sec 846 ms	5 sec 846 ms	5 sec 846 ms	0.21%
GCLocker Initiated GC	217	21.8 ms	1 sec 105 ms	4 sec 721 ms	0.17%
Total	22959	n/a	n/a	46 min 28 sec 455 ms	100.01%



3. System.gc()

System.gc() or Runtime.getRuntime().gc()

Always causes stop-the-world Full GCs. Try to avoid it. Because you are intervening with GC's ergnomics

May not always originate from your source code

- 3rd party libraries, frameworks, sometimes even application servers
- External tools (like VisualVM) through use of JMX.
- RMI.

-XX:+DisableExplicitGC

This JVM argument disables System.gc() globally in your JVM.

Real World Data from BIG SHIPPING COMPANY

RESOLUTION: 3500% PERFORMANCE INCREASE

- 99% Throughput
- From 12-23 second pauses to 660ms
- 89ms avg

Key Performance Indicators

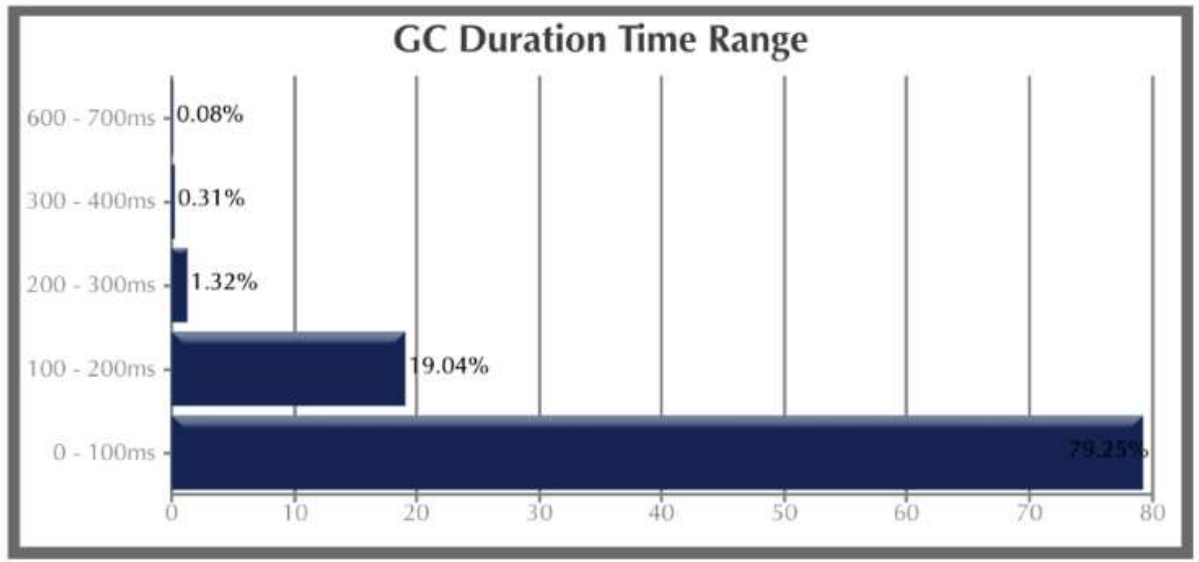
(Important section of the report. To learn more about KPIs, click here)

1. Throughput : 99.834%
2. Latency:

Avg Pause GC Time	89.1 ms
Max Pause GC Time	660 ms

GC **Pause** Duration Time Range:

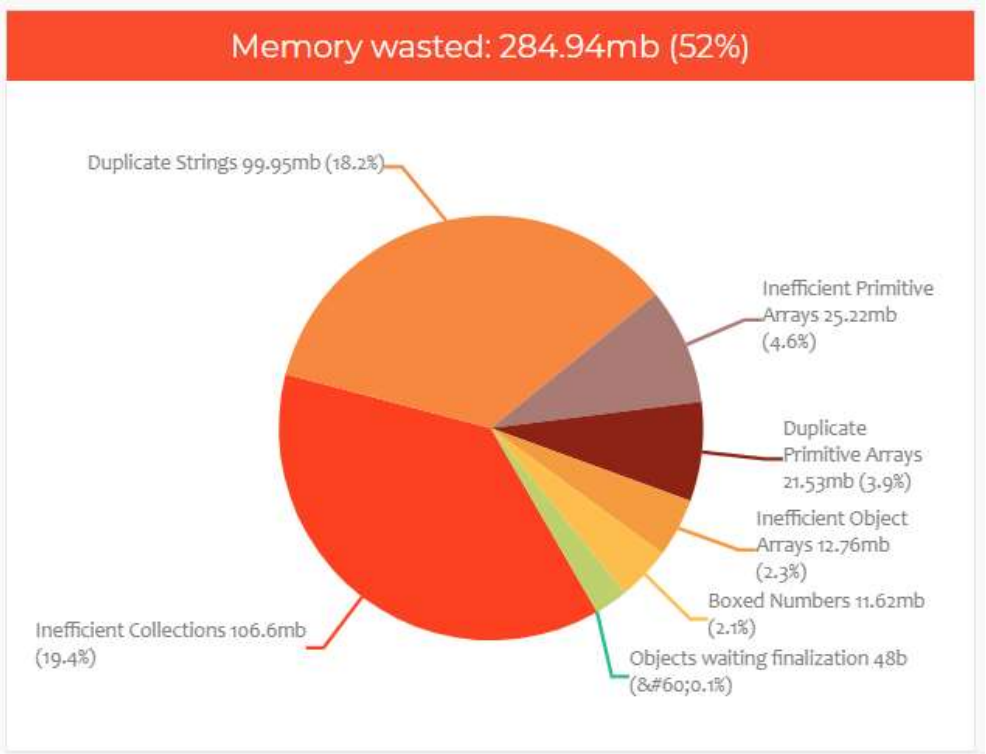
Duration (ms) 100 ms Change	No. of GCs	Percentage
0 - 100	1020	79.25%
100 - 200	245	19.04%
200 - 300	17	1.32%
300 - 400	4	0.31%
600 - 700	1	0.08%



4. High object creation rate

Total created bytes	298.29 gb
Total promoted bytes	70.44 gb
Avg creation rate	618.81 mb/sec
Avg promotion rate	146.13 mb/sec

30 – 90% memory wasted
due to inefficient programming



https://tinyurl.com/y3q5j5oj

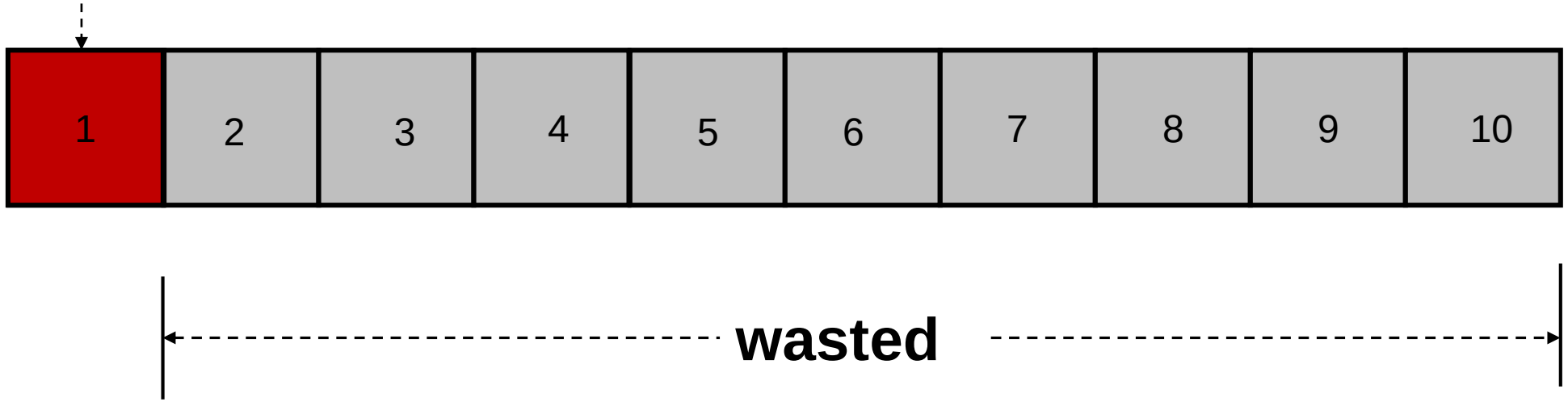
Do I need to care about memory?

Inefficiency in Collections

```
List<User> users = new ArrayList<>();
users.add(user);
```

ArrayList underlyingly maintains

Object[]

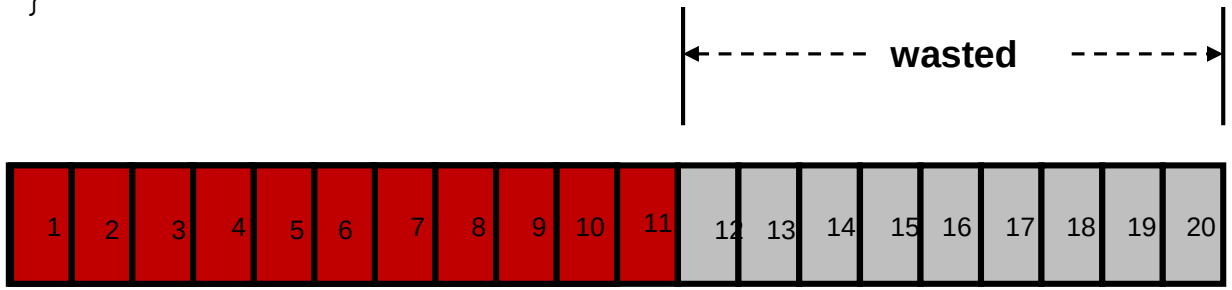


initial capacity = 10

Inefficiency in Collections

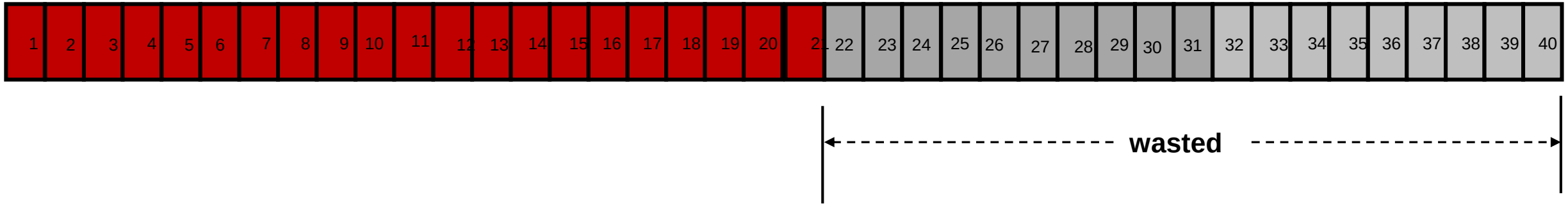
```
List<User> users = new ArrayList<>();
for (int counter = 1; counter <= 11; ++counter) {

        users.add(user);
}
```




```
for (int counter = 1; counter <= 21; ++counter) {

        users.add(user);
}
```



Recommendation 1: use capacity

```
new ArrayList<>();
```

```
new ArrayList<>(3);
```

Recommendation 2: Lazy Initialization

```
private List<User> users = new ArrayList<>();

public void addUser(User user) {

        users.add(user);
}
```

```
private List<User> users;

public void addUser(User user) {

    if (users == null) {
        users = new ArrayList<>();
    }

    users.add(user);
}
```

Recommendation 3: avoid clear()

```
List<User> users = new ArrayList<>();
users.clear();
```

```
List<User> users = new ArrayList<>();
users = null;
```

How all memory is wasted?

1. Duplicate Strings: https://heaphero.io/heap-recommendations/duplicate-strings.html
2. Wrong memory size settings
3. Inefficient Collections: http://heaphero.io/heap-recommendations/inefficient-collections.html
4. Duplicate Objects: https://heaphero.io/heap-recommendations/duplicate-objects.html
5. Duplicate arrays: https://heaphero.io/heap-recommendations/duplicate-primitive-arrays.html
6. Inefficient arrays: https://heaphero.io/heap-recommendations/inefficient-primitive-arrays.html
7. Objects waiting for finalization: https://heaphero.io/heap-recommendations/objects-waiting-finalization.html
8. Boxed numbers: https://heaphero.io/heap-recommendations/boxed-numbers.html
9. Object Headers: https://heaphero.io/heap-recommendations/object-headers.html

5. Choice of GC Algorithm

Choice GC algorithm plays a key role in GC performance.

01

Serial

http://gceasy.io/

02

Parallel

https://github.com/chewiebug/GCViewer

03

CMS

https://developer.ibm.com/javasdk/tools/

04

G1

https://h20392.www2.hpe.com/portal/swdepot/displayProductInfo.do?productNumber=HPJMETER

05

Shenandoah

https://code.google.com/archive/a/eclipselabs.org/p/garbagecat

06

Z GC

https://code.google.com/archive/a/eclipselabs.org/p/garbagecat

GC Time

[Times: user=11.53 sys=1.38, real=1.03 secs]

Real: Wall clock time

Pattern: Sys > User Time

System Time greater than User Time

In **149 GC event(s)**, 'sys' time is greater than 'usr' time. It's not a healthy sign.

Timestamp	User Time (secs)	Sys Time (secs)	Real Time (secs)
2016-08-31T01:18:19	0.31	0.75	0.06
2016-08-31T05:11:46	0.02	0.79	0.04
2016-08-31T06:26:10	0.01	0.75	0.04
2016-08-31T07:50:03	0.16	1.39	0.07
2016-08-31T09:04:21	0.12	1.59	0.08
2016-08-31T11:33:31	0.39	1.25	0.08
2016-08-31T14:31:18	0.31	2.08	0.11
2016-08-31T15:46:17	0.4	2.72	0.14
2016-08-31T17:01:16	0.56	1.86	0.11
2016-08-31T18:16:15	0.54	2.59	0.14

6. Process swapping

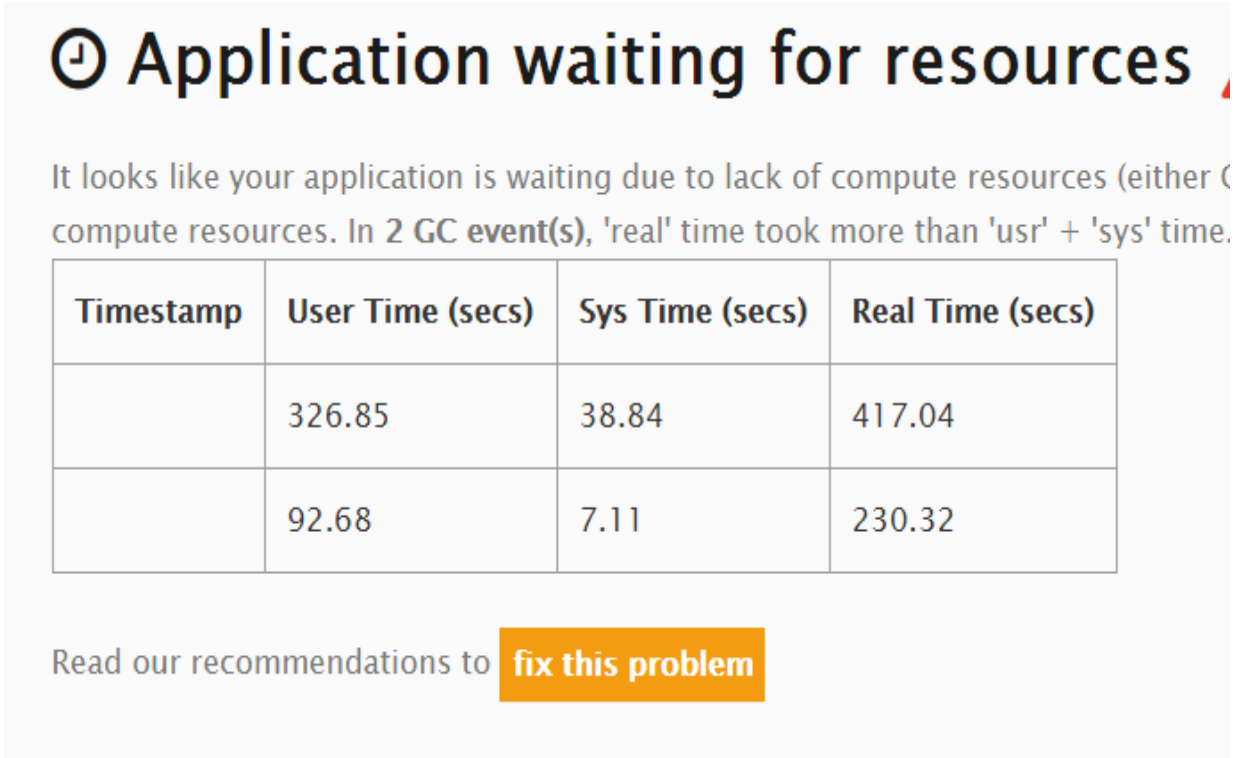
Lack of RAM (Memory)

Sometimes due to lack of memory (RAM), Operating system could be swapping your application from memory.

Process swapping

script will show all the process that are being swapped:
https://blog.gceasy.io/2016/11/22/reduce-long-gc-pauses/

Pattern: Real Time > User Time + Sys Time



7. Background I/O traffic

- If there is a heavy file system I/O activity (i.e. lot of reads and writes are happening) it can also cause long GC pauses.

Tit-bit: How to monitor I/O activity?

sar -d -p 1

‘System Activity Report’ command reports read/write activity made every 1 second

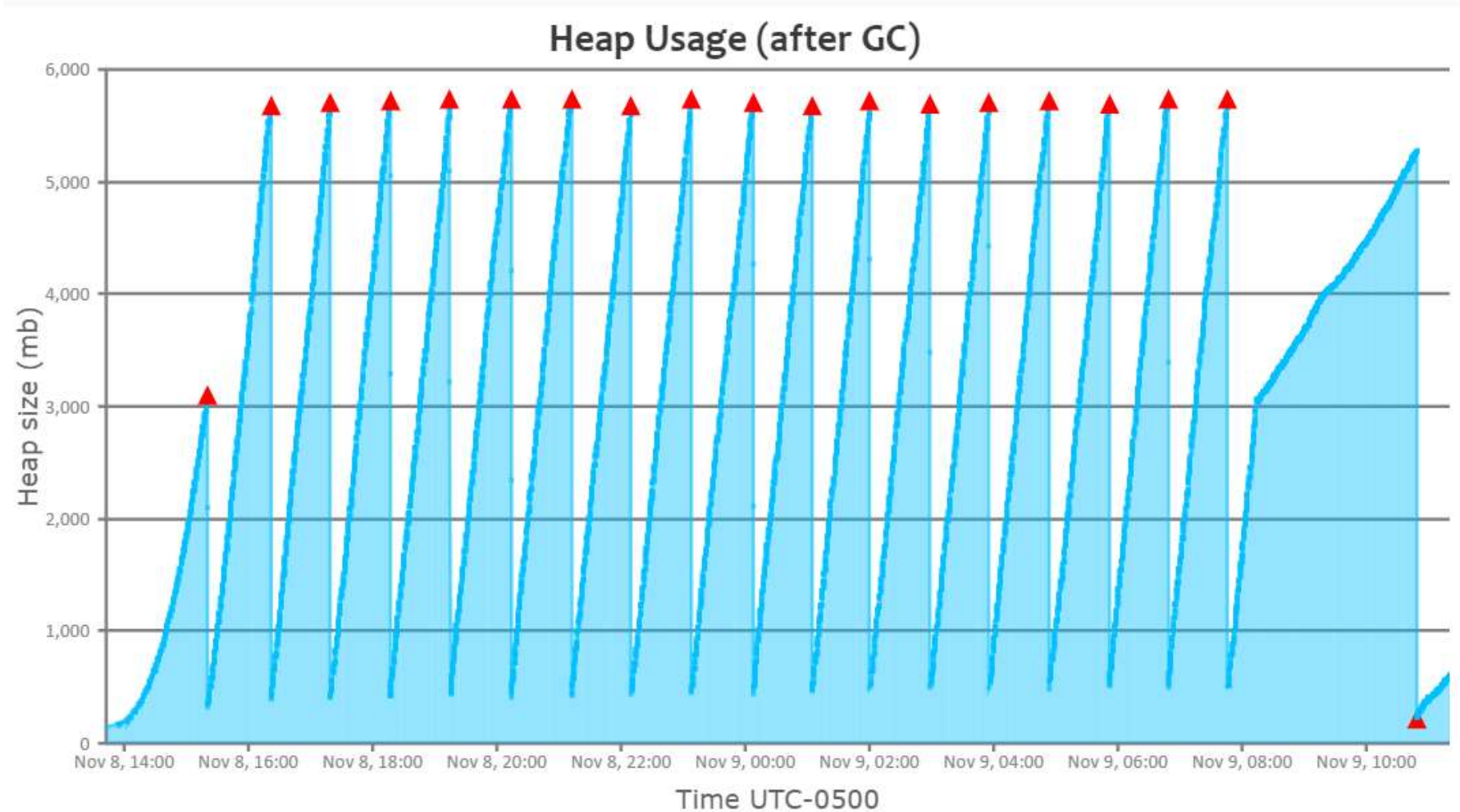
8. Less GC Threads

- WARNING: Adding too many GC threads will consume a lot of CPU and takes away a resource from your application. Thus you need to conduct thorough testing before increasing the GC thread count.

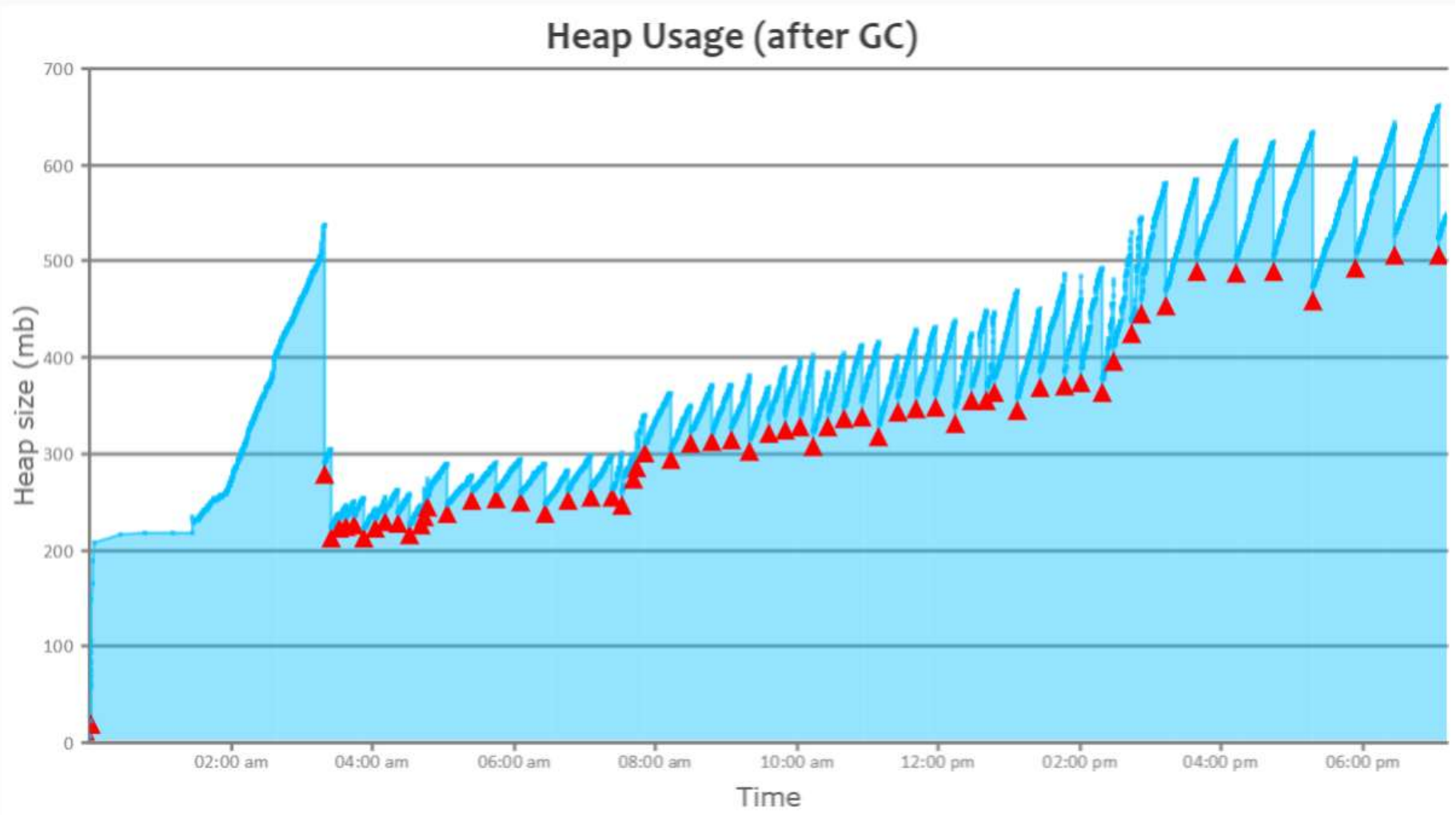
9. Wrong Heap/Generation size settings

Troubleshooting

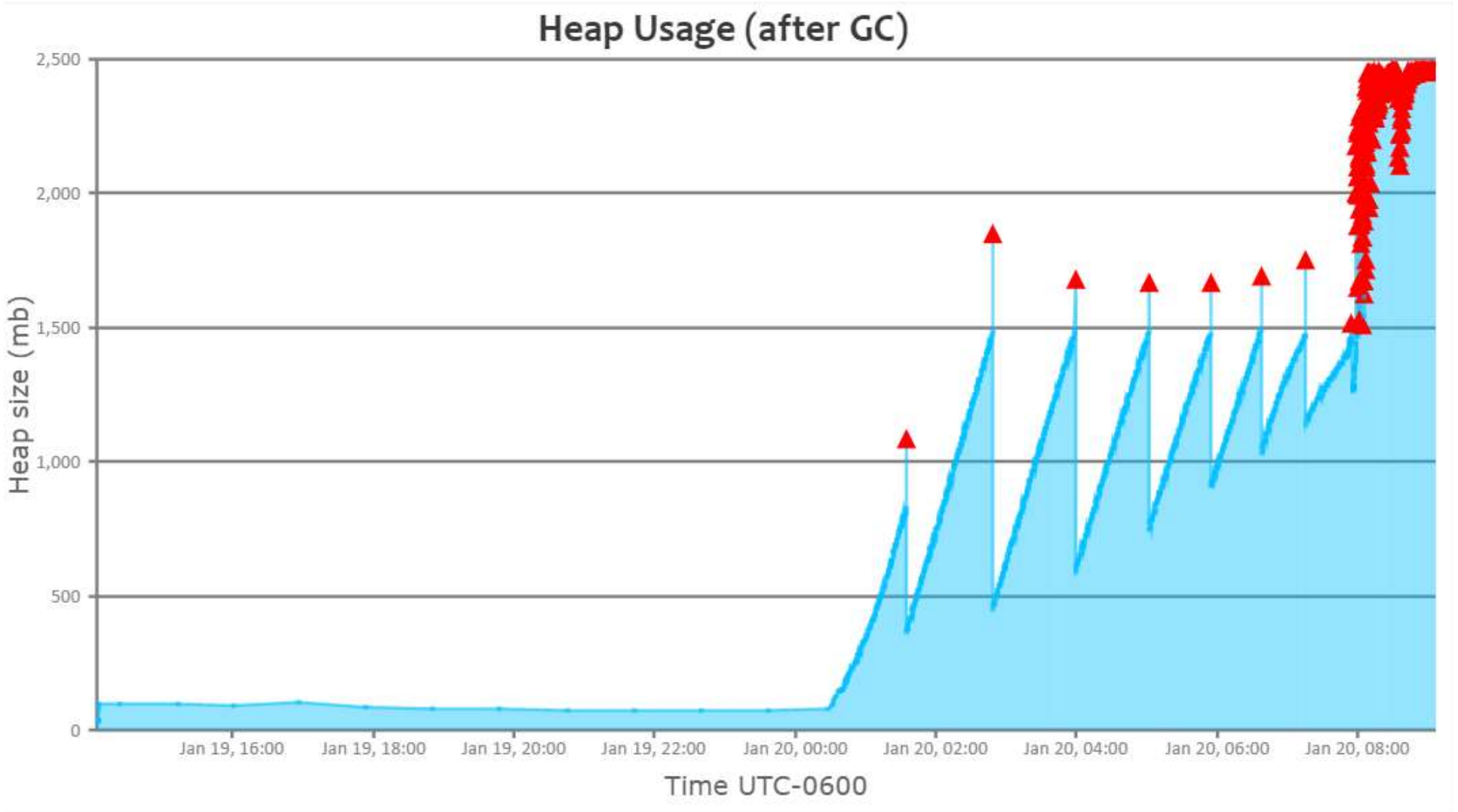
Heap usage graph



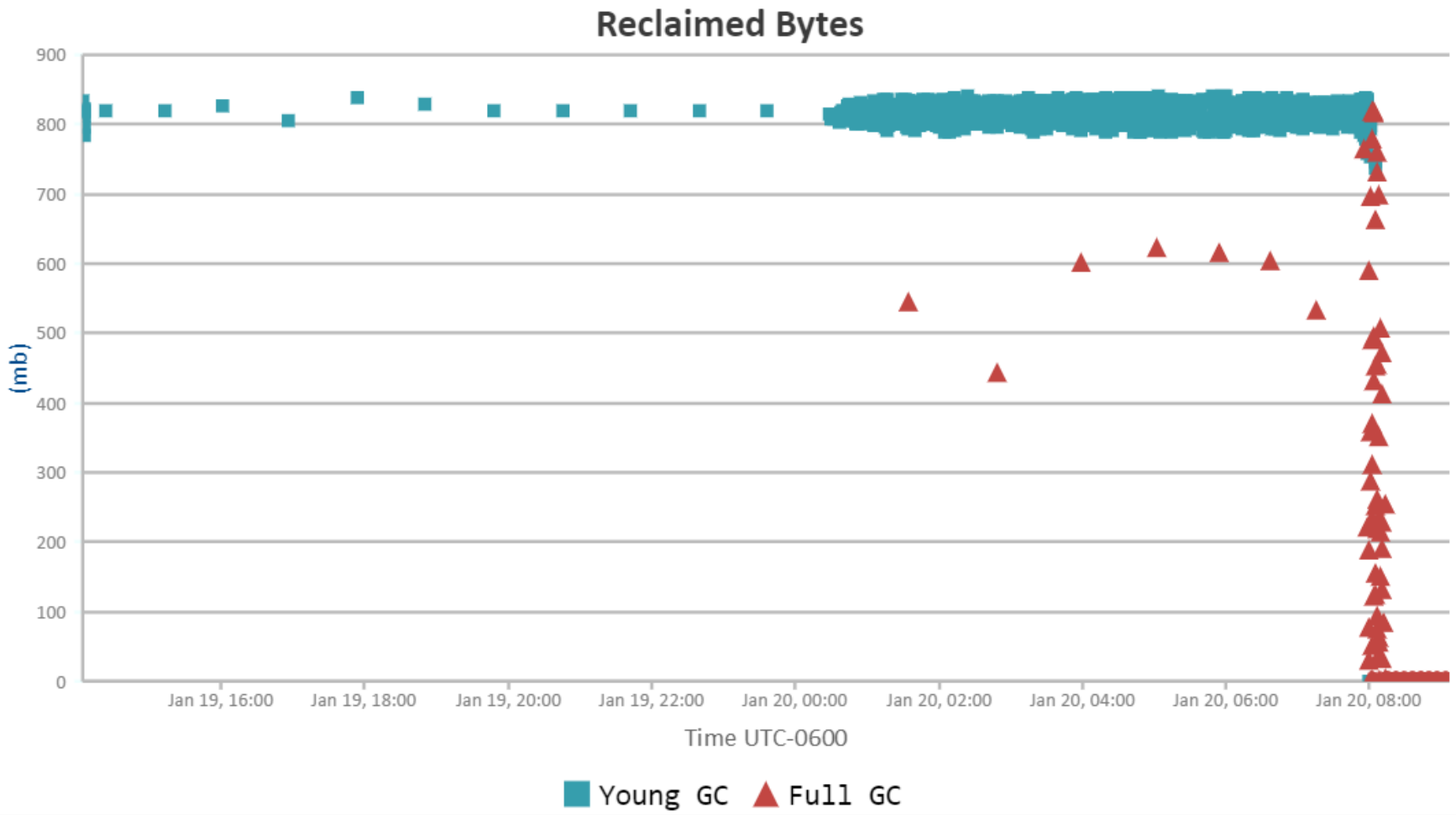
What is your observation?



Memory Problem



Corresponding – Reclaimed bytes chart



Micrometrics – Forecast problems

https://blog.gceasy.io/2019/03/13/micrometrics-to-forecast-application-performance/

GC Throughput

Amount time application spends in processing customer transactions vs amount of time application spend in doing GC

GC Latency

If pause time starts to increase, then it's an indication that app is suffering from memory problems

Object Reclamation rate

If number of objects created in unit time

File Descriptors

File descriptor is a handle to access: File, Pipe, Network Connections. If count grows it's a lead indicator that application isn't closing resources properly.

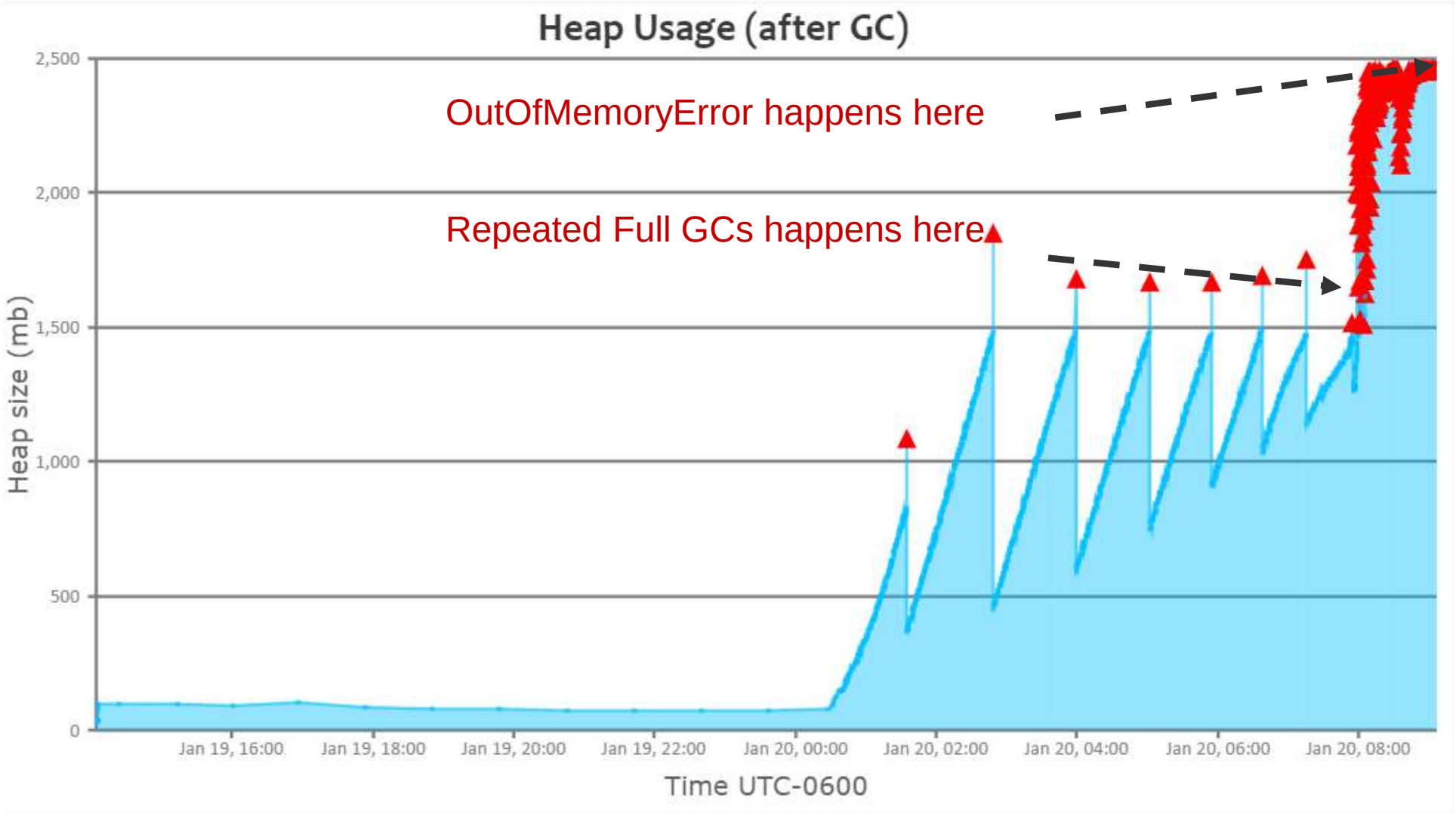
Thread States

If BLOCKED thread state count grows, it's an early indication that your application has potential to become unresponsive

Few more…

TCP/IP States, Hosts count, IOPS, ..

Micro-metrics: Early Indicators



How to diagnose memory leak?

Capture heap dumps

jmap -dump:live,file=<file-path> <pid>
Example: jmap -dump:live,file=/opt/tmp/AddressBook-heapdump.bin 37320

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/logs/heapdump

Eclipse MAT, HeapHero

Two good tools to analyze memory leaks

What is garbage?

Everything is garbage!

Thank you my friends!

Ram Lakshmanan

ram@tier1app.com

@tier1app

https://www.linkedin.com/company/gceasy

Slides will be available in: https://blog.gceasy.io