hands-on perf workshop

let us into your performance #BCNJUG

@srvaroa
@duarte_nunes
@gontanon

May 25 2017, Barcelona JUG

hello

agenda

Intro & mechanics

The workshop

- Exercises

Wrap up

first of all

Check out our repository at:

https://github.com/srvaroa/jug-perf-workshop

WIFI details

NR-GUEST - RubyOnRails!

intro and mechanics

Some exercises to let you discover & play with tools

One presenter per exercise.

Presenters will wander around the room supporting

exercise mechanics

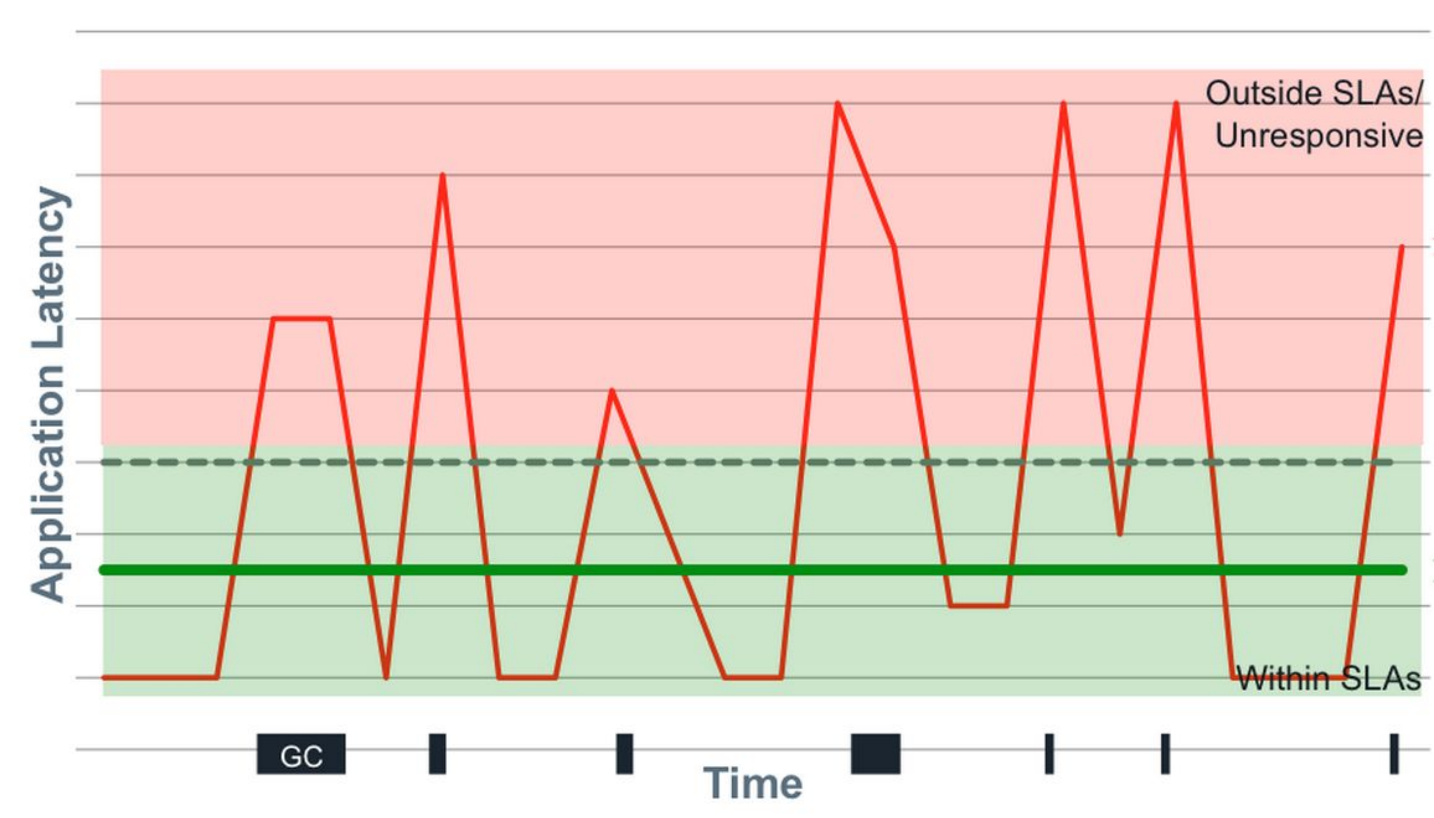
1. Each exercise pretends to be an ‘application’, measure its latency
2. Profile & troubleshoot with tools
3. Make changes
4. Repeat

how to measure latency

latency

latency: the time it takes for an operation to complete, such as an application request, a database query, a file system operation, etc.

hiccups

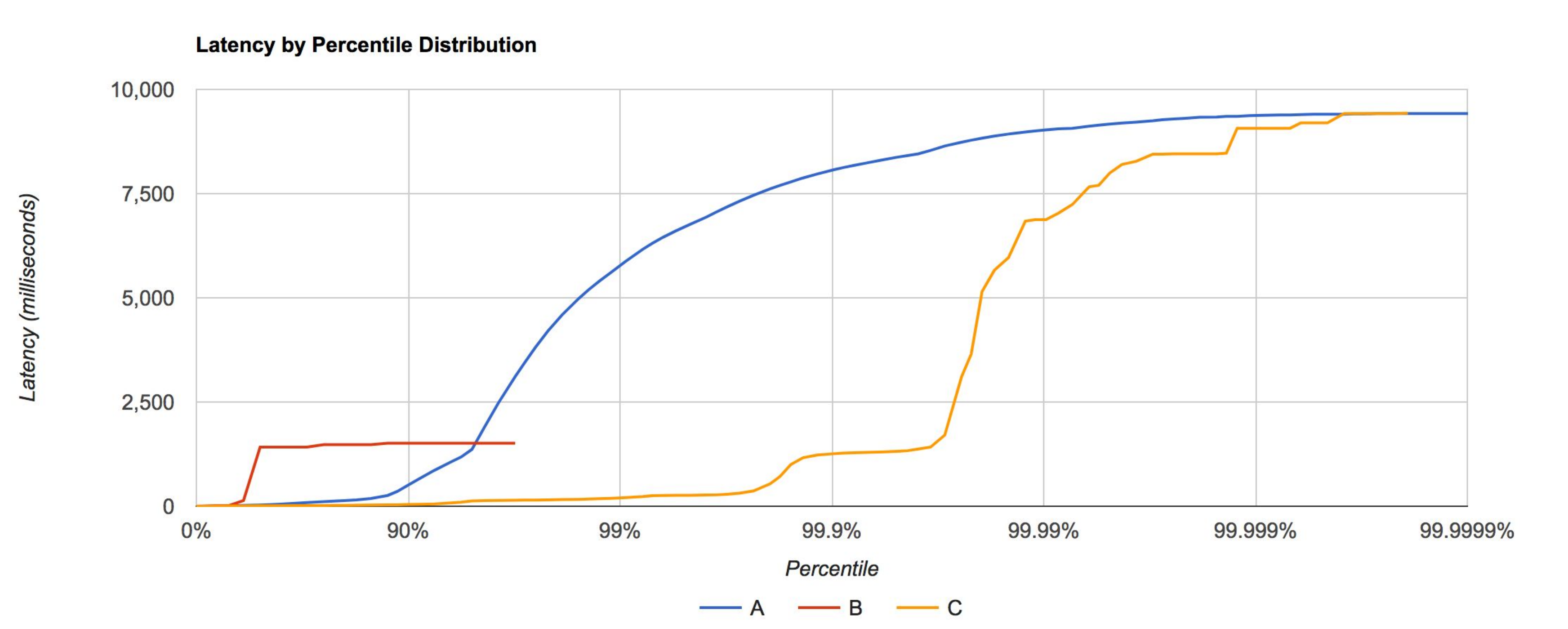

Outside SLAs/
Unresponsive
Application Latency
Within SLAs
GC
Time

percentiles

Measure percentiles (90th, 99th, 99.9th, ...)

- They quantify slowest requests in the population
- Think in SLAs: max latency allowed for a percentile
- Typically measured as a function of some load
- The max is also very meaningful

a latency percentile distribution



coordinated omission

When the loader coordinates with the system under test:

> The loader issues requests one by one at a certain rate, but fails to impose that rate due to backoff (e.g., due to synchronous requests);

or

> Measure latency before start and end of an operation, but delays outside of timing window do not get measured at all (like queuing and garbage collection pauses).

toolset

toolset

Tools list:

- jstack
- gc logs
- visualvm
- jmap
- jit logs
- gcviewer
- gcviewer

verify toolset

```
> ./check_tools.sh
java ok
javac ok
jstat ok
jvisualvm ok
jstack ok
jmap ok
Java is HotSpot ok
Java compiler is HotSpot ok
Java is 1.8 ok
Java compiler is 1.8 ok
```

how to run an exercise

Take latency measurements (20 secs):

```
    > ./run.sh <exercise_number>
    Latencies in usec recorded to build/Ex<number>-1495666400.hist
```

Open charts/plotFiles.html in browser, load histogram

For longer profiling / troubleshooting runs (e.g., 5 minutes):

```
    > ./run.sh <exercise_number> 300000
```

Exercise 1

A garbage generator

Look at: profiler & GC

Exercise 1

A linked list, storing pairs of strings that get added and removed

1_0

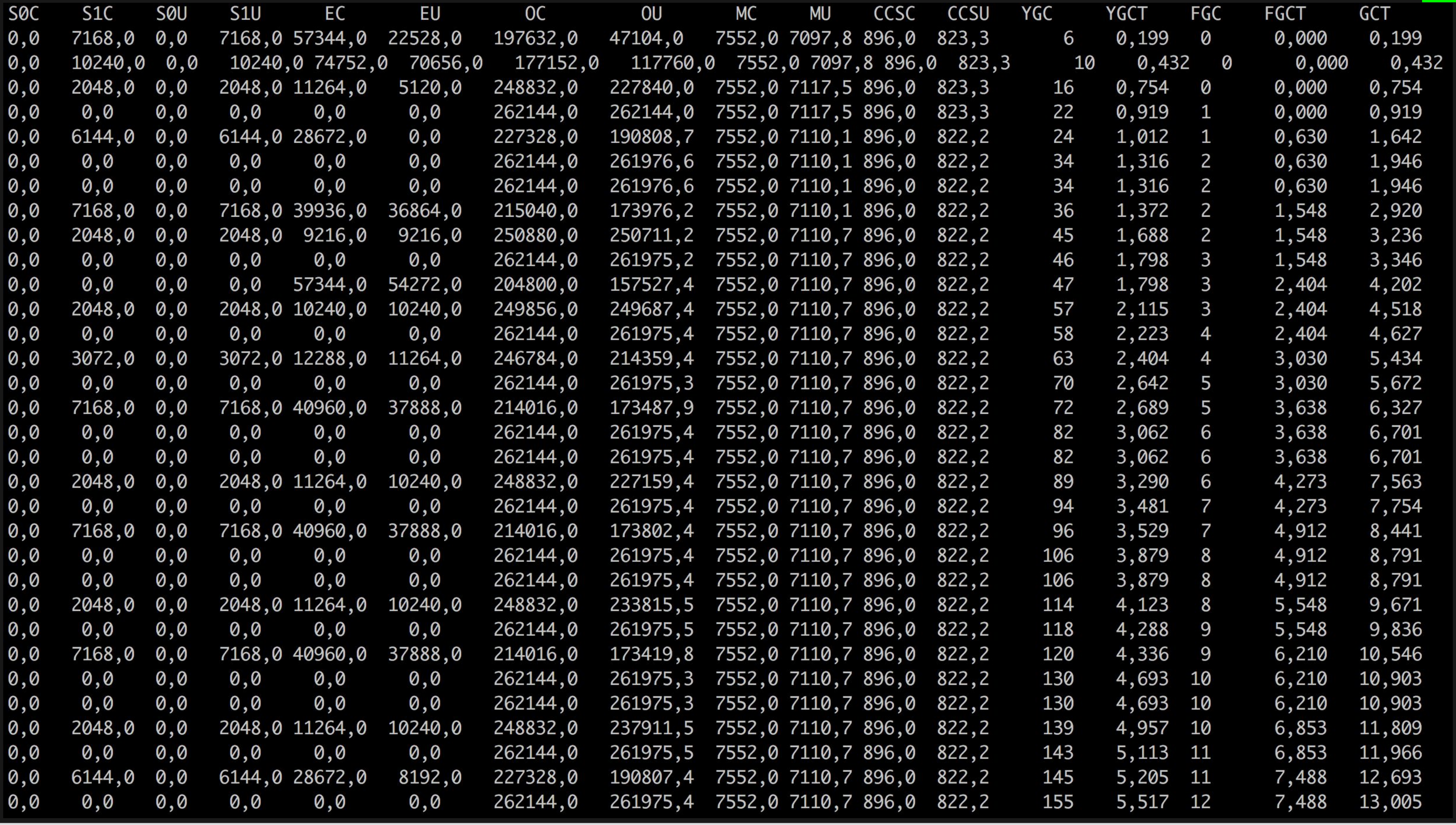
1_0 → base example

Let’s execute and see behaviour

1_0

```
jstat -gc -h80 $(jps | grep Ex | cut -f1 -d' ') 500
```

1_0

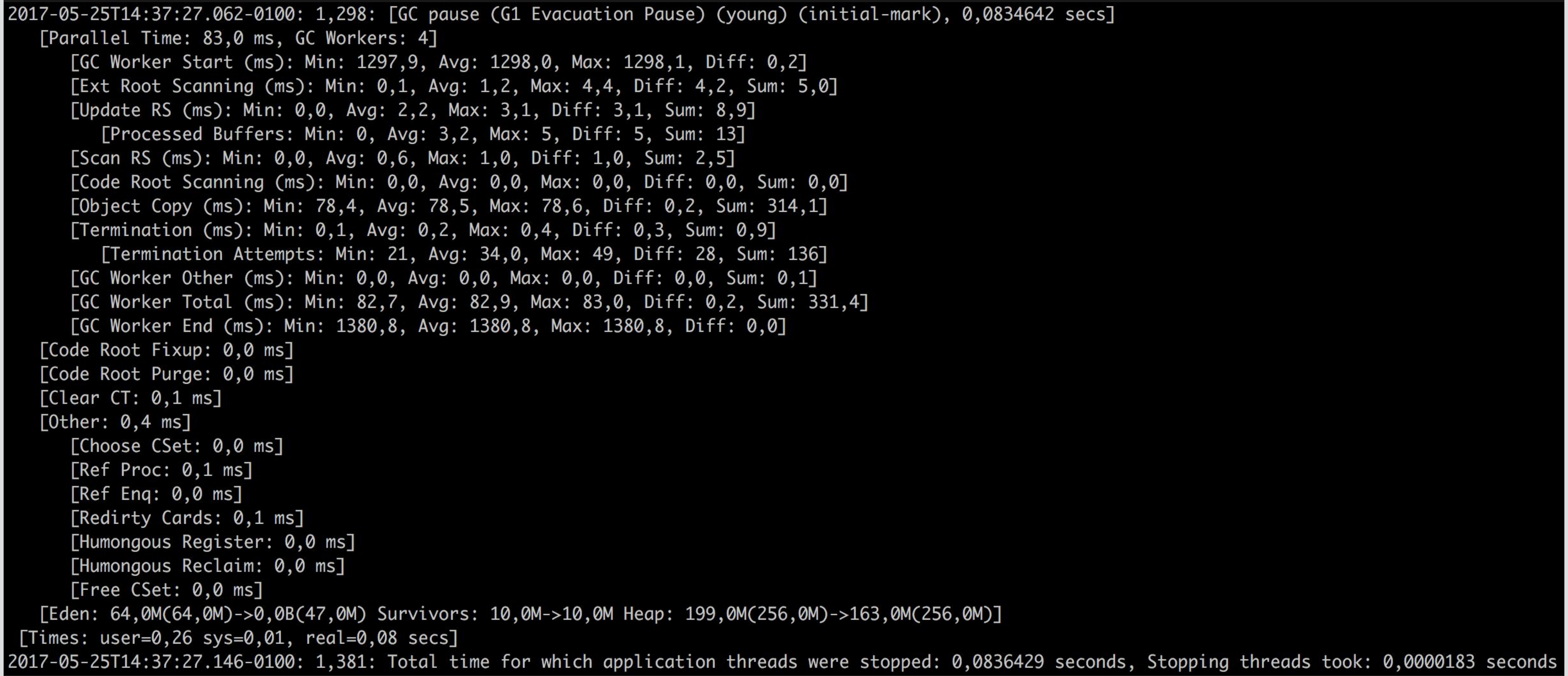


```
S0C      S1C   S0U      S1U       EC       EU         OC         OU       MC      MU   CCSC   CCSU    YGC     YGCT  FGC      FGCT      GCT
0,0   7168,0   0,0   7168,0  57344,0  22528,0   197632,0    47104,0   7552,0  7097,8  896,0  823,3      6    0,199    0     0,000    0,199
0,0  10240,0   0,0  10240,0  74752,0  70656,0   177152,0   117760,0   7552,0  7097,8  896,0  823,3     10    0,432    0     0,000    0,432
0,0   2048,0   0,0   2048,0  11264,0   5120,0   248832,0   227840,0   7552,0  7117,5  896,0  823,3     16    0,754    0     0,000    0,754
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   262144,0   7552,0  7117,5  896,0  823,3     22    0,919    1     0,000    0,919
0,0   6144,0   0,0   6144,0  28672,0      0,0   227328,0   190808,7   7552,0  7110,1  896,0  822,2     24    1,012    1     0,630    1,642
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261976,6   7552,0  7110,1  896,0  822,2     34    1,316    2     0,630    1,946
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261976,6   7552,0  7110,1  896,0  822,2     34    1,316    2     0,630    1,946
0,0   7168,0   0,0   7168,0  39936,0  36864,0   215040,0   173976,2   7552,0  7110,1  896,0  822,2     36    1,372    2     1,548    2,920
0,0   2048,0   0,0   2048,0   9216,0   9216,0   250880,0   250711,2   7552,0  7110,7  896,0  822,2     45    1,688    2     1,548    3,236
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,2   7552,0  7110,7  896,0  822,2     46    1,798    3     1,548    3,346
0,0      0,0   0,0      0,0  57344,0  54272,0   204800,0   157527,4   7552,0  7110,7  896,0  822,2     47    1,798    3     2,404    4,202
0,0   2048,0   0,0   2048,0  10240,0  10240,0   249856,0   249687,4   7552,0  7110,7  896,0  822,2     57    2,115    3     2,404    4,518
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2     58    2,223    4     2,404    4,627
0,0   3072,0   0,0   3072,0  12288,0  11264,0   246784,0   214359,4   7552,0  7110,7  896,0  822,2     63    2,404    4     3,030    5,434
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,3   7552,0  7110,7  896,0  822,2     70    2,642    5     3,030    5,672
0,0   7168,0   0,0   7168,0  40960,0  37888,0   214016,0   173487,9   7552,0  7110,7  896,0  822,2     72    2,689    5     3,638    6,327
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2     82    3,062    6     3,638    6,701
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2     82    3,062    6     3,638    6,701
0,0   2048,0   0,0   2048,0  11264,0  10240,0   248832,0   227159,4   7552,0  7110,7  896,0  822,2     89    3,290    6     4,273    7,563
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2     94    3,481    7     4,273    7,754
0,0   7168,0   0,0   7168,0  40960,0  37888,0   214016,0   173802,4   7552,0  7110,7  896,0  822,2     96    3,529    7     4,912    8,441
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2    106    3,879    8     4,912    8,791
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2    106    3,879    8     4,912    8,791
0,0   2048,0   0,0   2048,0  11264,0  10240,0   248832,0   233815,5   7552,0  7110,7  896,0  822,2    114    4,123    8     5,548    9,671
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,5   7552,0  7110,7  896,0  822,2    118    4,288    9     5,548    9,836
0,0   7168,0   0,0   7168,0  40960,0  37888,0   214016,0   173419,8   7552,0  7110,7  896,0  822,2    120    4,336    9     6,210   10,546
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,3   7552,0  7110,7  896,0  822,2    130    4,693   10     6,210   10,903
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,3   7552,0  7110,7  896,0  822,2    130    4,693   10     6,210   10,903
0,0   2048,0   0,0   2048,0  11264,0  10240,0   248832,0   237911,5   7552,0  7110,7  896,0  822,2    139    4,957   10     6,853   11,809
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,5   7552,0  7110,7  896,0  822,2    143    5,113   11     6,853   11,966
0,0   6144,0   0,0   6144,0  28672,0   8192,0   227328,0   190807,4   7552,0  7110,7  896,0  822,2    145    5,205   11     7,488   12,693
0,0      0,0   0,0      0,0      0,0      0,0   262144,0   261975,4   7552,0  7110,7  896,0  822,2    155    5,517   12     7,488   13,005
```

1_0

```
less ./build/Ex_1_0_gc_$timestamp.log
```

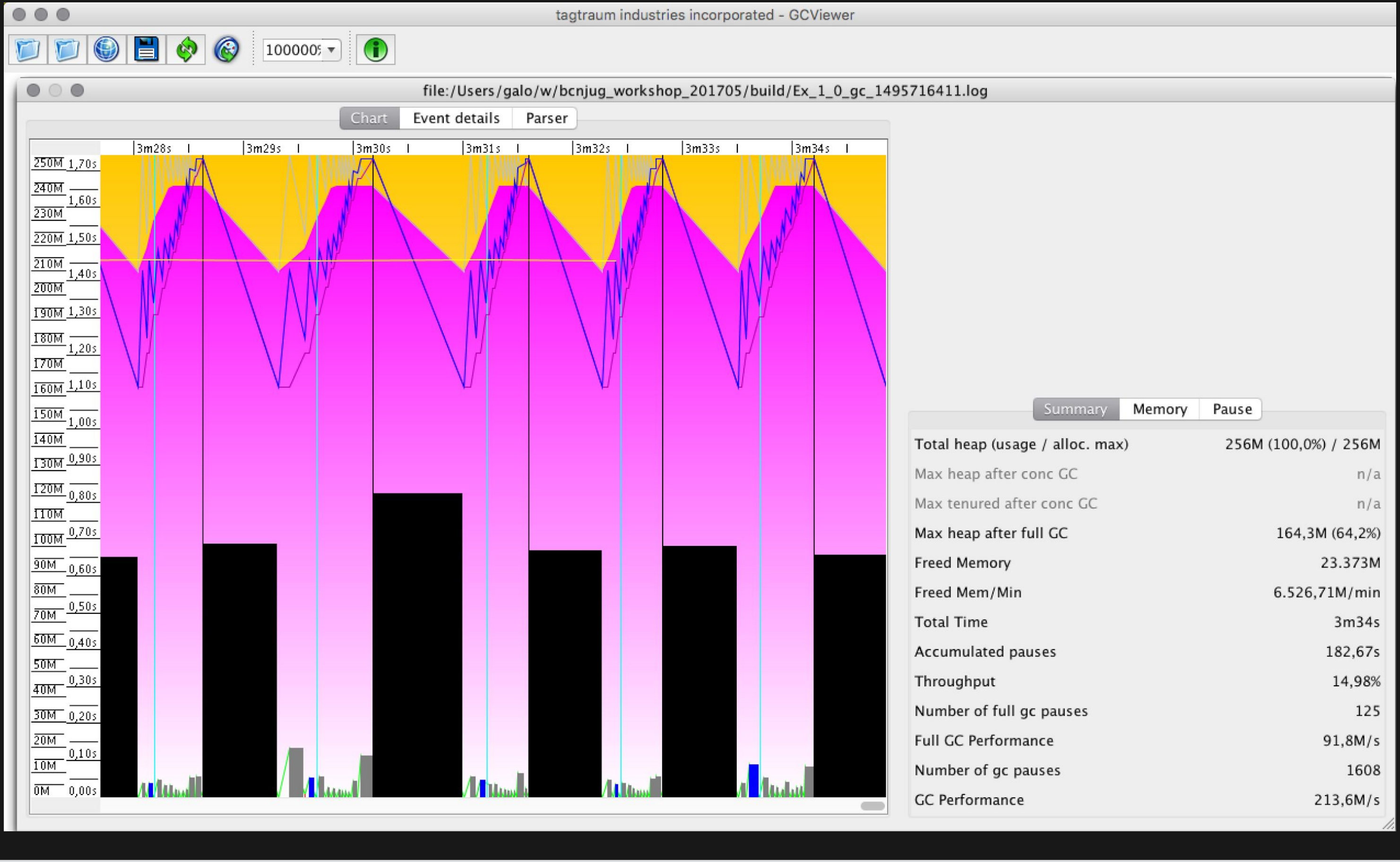
1_0



```
2017-05-25T14:37:27.062-0100: 1,298: [GC pause (G1 Evacuation Pause) (young) (initial-mark), 0,0834642 secs]
   [Parallel Time: 83,0 ms, GC Workers: 4]
      [GC Worker Start (ms): Min: 1297,9, Avg: 1298,0, Max: 1298,1, Diff: 0,2]
      [Ext Root Scanning (ms): Min: 0,1, Avg: 1,2, Max: 4,4, Diff: 4,2, Sum: 5,0]
      [Update RS (ms): Min: 0,0, Avg: 2,2, Max: 3,1, Diff: 3,1, Sum: 8,9]
         [Processed Buffers: Min: 0, Avg: 3,2, Max: 5, Diff: 5, Sum: 13]
      [Scan RS (ms): Min: 0,0, Avg: 0,6, Max: 1,0, Diff: 1,0, Sum: 2,5]
      [Code Root Scanning (ms): Min: 0,0, Avg: 0,0, Max: 0,0, Diff: 0,0, Sum: 0,0]
      [Object Copy (ms): Min: 78,4, Avg: 78,5, Max: 78,6, Diff: 0,2, Sum: 314,1]
      [Termination (ms): Min: 0,1, Avg: 0,2, Max: 0,4, Diff: 0,3, Sum: 0,9]
         [Termination Attempts: Min: 21, Avg: 34,0, Max: 49, Diff: 28, Sum: 136]
      [GC Worker Other (ms): Min: 0,0, Avg: 0,0, Max: 0,0, Diff: 0,0, Sum: 0,1]
      [GC Worker Total (ms): Min: 82,7, Avg: 82,9, Max: 83,0, Diff: 0,2, Sum: 331,4]
      [GC Worker End (ms): Min: 1380,8, Avg: 1380,8, Max: 1380,8, Diff: 0,0]
   [Code Root Fixup: 0,0 ms]
   [Code Root Purge: 0,0 ms]
   [Clear CT: 0,1 ms]
   [Other: 0,4 ms]
      [Choose CSet: 0,0 ms]
      [Ref Proc: 0,1 ms]
      [Ref Enq: 0,0 ms]
      [Redirty Cards: 0,1 ms]
      [Humongous Register: 0,0 ms]
      [Humongous Reclaim: 0,0 ms]
      [Free CSet: 0,0 ms]
   [Eden: 64,0M(64,0M)->0,0B(47,0M) Survivors: 10,0M->10,0M Heap: 199,0M(256,0M)->163,0M(256,0M)]
 [Times: user=0,26 sys=0,01, real=0,08 secs]
2017-05-25T14:37:27.146-0100: 1,381: Total time for which application threads were stopped: 0,0836429 seconds, Stopping threads took: 0,0000183 seconds
```

GCViewer → visualize GC logs

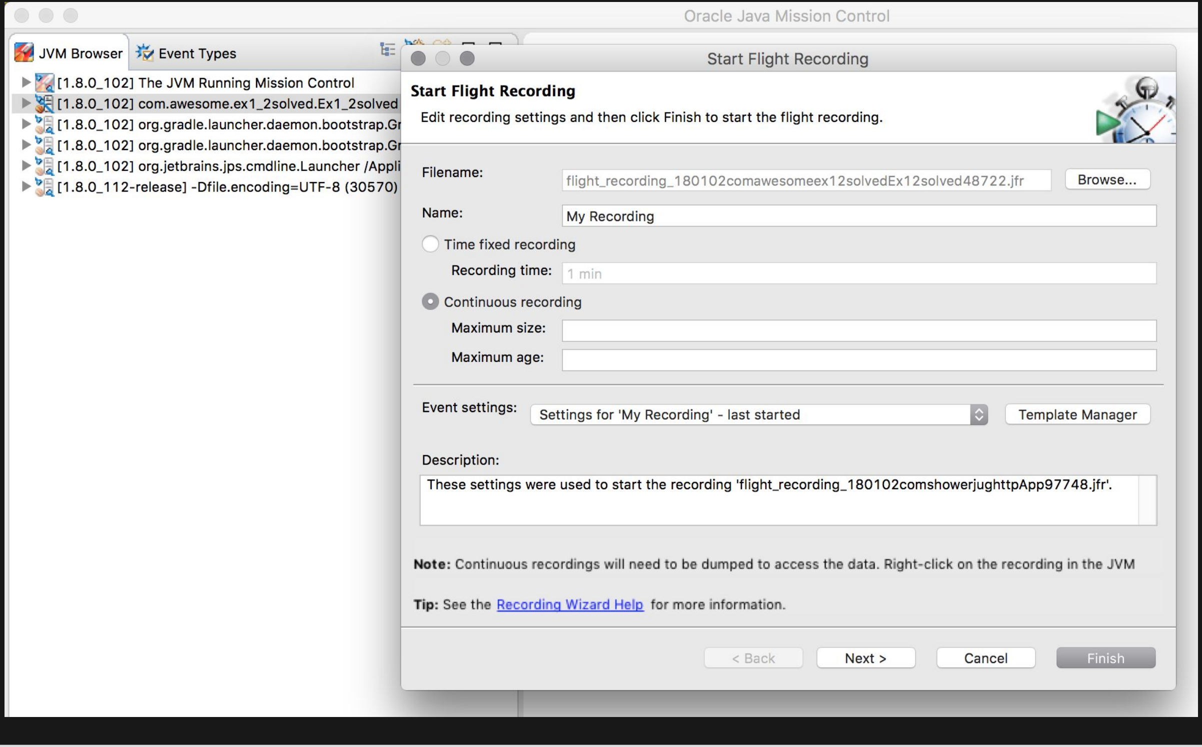
1_0



file:/Users/galo/w/bcnjug_workshop_201705/build/Ex_1_0_gc_1495716411.log

Total heap (usage / alloc. max)	256M (100,0%) / 256M
Max heap after conc GC	n/a
Max tenured after conc GC	n/a
Max heap after full GC	164,3M (64,2%)
Freed Memory	23.373M
Freed Mem/Min	6.526,71M/min
Total Time	3m34s
Accumulated pauses	182,67s
Throughput	14,98%
Number of full gc pauses	125
Full GC Performance	91,8M/s
Number of gc pauses	1608
GC Performance	213,6M/s

jvisualvm / YourKit → Select process → Monitor + Visual GC jmc → Select process → Record → Open recording jmc → Select process → Record → open recording



- [1.8.0_102] The JVM Running Mission Control
- [1.8.0_102] com.awesome.ex1_2solved.Ex1_2solved
- [1.8.0_102] org.gradle.launcher.daemon.bootstrap.Gr
- [1.8.0_102] org.gradle.launcher.daemon.bootstrap.Gr
- [1.8.0_102] org.jetbrains.jps.cmdline.Launcher /Appli
- [1.8.0_112-release] -Dfile.encoding=UTF-8 (30570)

Start Flight Recording

Start Flight Recording

Edit recording settings and then click Finish to start the flight recording.

Filename: flight_recording_180102comawesomeex12solvedEx12solved48722.jfr Browse...

Name: My Recording

○ Time fixed recording

Recording time: 1 min

● Continuous recording

Maximum size:

Maximum age:

Event settings: Settings for 'My Recording' - last started Template Manager

Description:

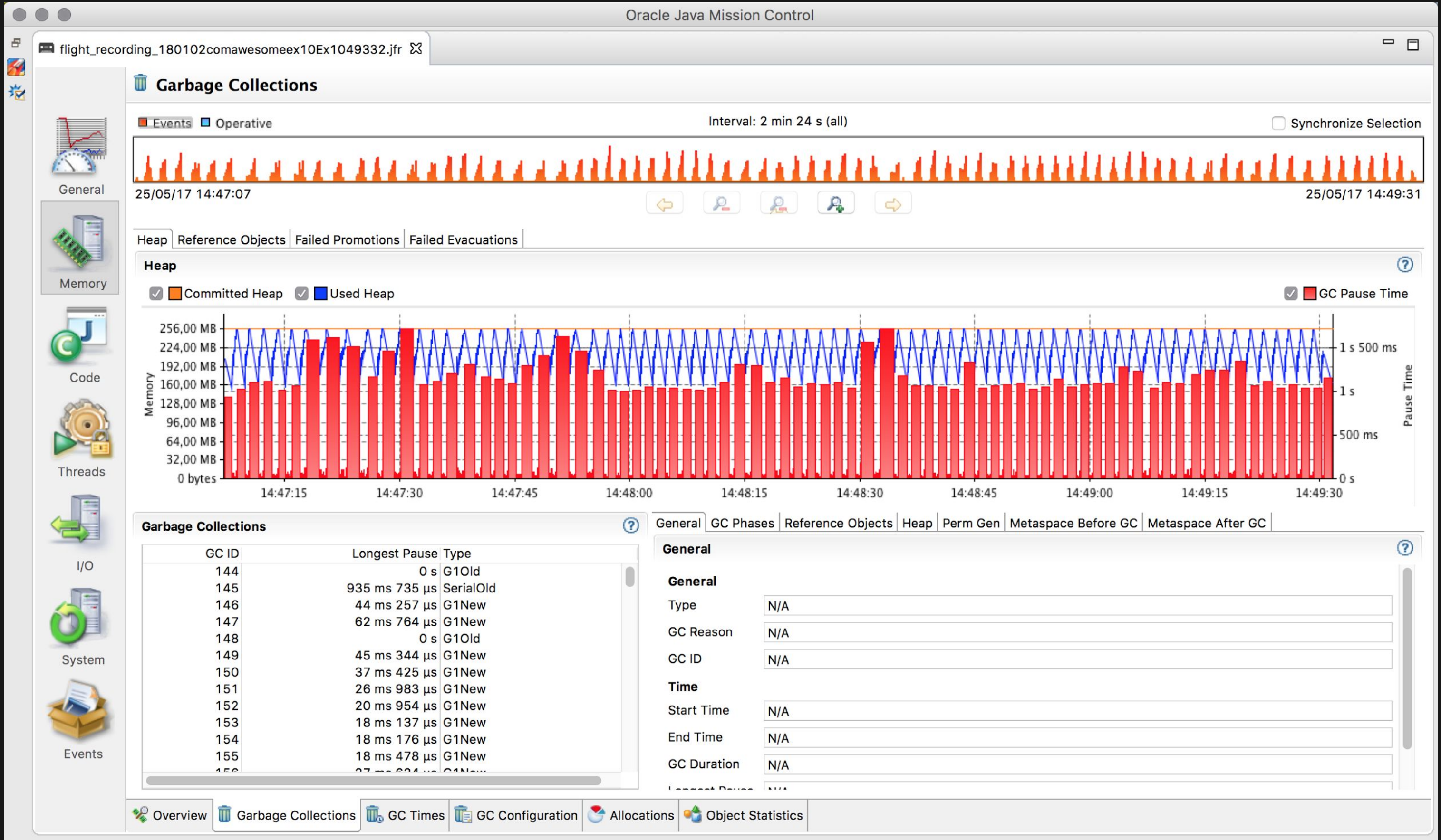
These settings were used to start the recording 'flight_recording_180102comshowerjughttpApp97748.jfr'.

Note: Continuous recordings will need to be dumped to access the data. Right-click on the recording in the JVM

Tip: See the Recording Wizard Help for more information.

< Back | Next > | Cancel | Finish

1_0



Garbage Collections

GC ID	Longest Pause	Type
144	0 s	G1Old
145	935 ms 735 µs	SerialOld
146	44 ms 257 µs	G1New
147	62 ms 764 µs	G1New
148	0 s	G1Old
149	45 ms 344 µs	G1New
150	37 ms 425 µs	G1New
151	26 ms 983 µs	G1New
152	20 ms 954 µs	G1New
153	18 ms 137 µs	G1New
154	18 ms 176 µs	G1New
155	18 ms 478 µs	G1New

General

General

Type	N/A
GC Reason	N/A
GC ID	N/A

Time

Start Time	N/A
End Time	N/A
GC Duration	N/A

1_1

1_1 → some obvious improvements

Rinse and repeat

1_2

1_2 → unimplemented

1_2solved → solutions

1_2

1_2solved → more improvements

Rinse and repeat

Exercise 1: Takeaways

JVM is good at dealing with garbage

But you need to help it

Avoid unnecessary copies (e.g. new String(“a”))

Watch memory overhead of collections & data structures

Exercise 2

A 2-threaded producer / consumer

Look at: profiler & GC

2_1

Let’s replace the linked list with a specialized concurrent queue:

```
    java.util.concurrent.LinkedBlockingQueue
```

2_1 → unimplemented

2_1solved → solution

2_2

A non-blocking array-backed queue:

```
    org.jctools.queues.SpmcArrayQueue
```

2_2 → unimplemented

2_2solved → solution

Bonus points:

If the queue is full, the producer can become a consumer.

Exercise 2: Takeaways

Coordination is expensive

Don’t use synchronized, use specialized data structures

Unbounded queues:

- Produce garbage
- Don’t give back pressure, could fill your heap

Non-blocking queues much faster. But you spin on them.

Exercise 3

An operation looks expensive. How can it be improved?

Exercise 3: Takeaways

The Flyweight pattern, while ugly to write, can give a good performance boost as it avoids the object representation

That same technique can be used for off-heap data structures

Slightly better with Scala’s value types

Exercise 4

A set of unary operations, which can all execute in a single CPU cycle and generate no garbage, produce an uncommon latency profile.

What’s going on?

Hint: you’ll probably need to activate some JVM flags in run.sh

Exercise 4: (De)Optimizations

The JIT can make a bet, but later revoke it through the use of traps

Optimizations apply at a given call site

A call site can be monomorphic (only one observed type), bimorphic or megamorphic (with different frequencies of observed types)

Class Hierarchy Analysis easily optimizes the mono and bimorphic cases

Exercise 4: Tiered Compilation

```
enum CompLevel {
  CompLevel_none              = 0, // Interpreter
  CompLevel_simple            = 1, // C1
  CompLevel_limited_profile   = 2, // C1, invocation & backedge counters
  CompLevel_full_profile      = 3, // C1, invocation & backedge counters + mdo
  CompLevel_full_optimization = 4, // C2
}
```

Exercise 4: Analysis

```
Running Ex5
Running loop 1
com.awesome.ex5.Ex5$DoubleOperation
   8862   90      4       com.awesome.ex5.Ex5::operation (138 bytes)   made
not entrant
   ...
   8863  101      3       com.awesome.ex5.Ex5::operation (138 bytes)
```

Exercise 4: Analysis

```
com.awesome.ex5.Ex5$HalfOperation
  12841    94       4       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
  ...
  12842   103       3       com.awesome.ex5.Ex5::measureOp (20 bytes)
  ...
  12875   110       4       com.awesome.ex5.Ex5::measureOp (20 bytes)
  12881   103       3       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
```

Exercise 4: Analysis

```
com.awesome.ex5.Ex5$HalfOperation
  12841   94       4       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
    <uncommon_trap thread='7171'
            reason='class_check'
            action='maybe_recompile'
            compile_id='94'
            compiler='C2' level='4'
            count='3'
            state='class_check' stamp='12.841'>
        <jvms bci='7' ... decompiles='1' class_check_traps='3'/>
    </uncommon_trap>
```

Exercise 4: Analysis

```
com.awesome.ex5.Ex5$HalfOperation
  12841   94        4       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
   ...
  12842  103        3       com.awesome.ex5.Ex5::measureOp (20 bytes)
   ...
  12875  107        4       com.awesome.ex5.Ex5::measureOp (20 bytes)
  12881  103        3       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
```

Exercise 4: Analysis

```
com.awesome.ex5.Ex5$IncOperation
  12224  108       4       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
  12224  110       3       com.awesome.ex5.Ex5::measureOp (20 bytes)
  ...
  12227  116       4       com.awesome.ex5.Ex5::measureOp (20 bytes)
  ...
  12232  110       3       com.awesome.ex5.Ex5::measureOp (20 bytes)   made
not entrant
```

Exercise 4: Analysis

```
com.awesome.ex5.Ex5$DecOperation
   ...
```

Exercise 4: Takeaways

Careful with highly polymorphic code in your fast path

JVM performance testing

Many pitfalls:

- Classloading
- Compilation
- Fake warmups
- Garbage Collection
- Eliminated code

Q & A