

# Performance Optimisations

Angelo D' Agnano

John Kostaras

# Performance activities

- \* Performance Monitoring
  - \* an activity of nonintrusively collecting or observing performance data from a running application
- \* Performance Profiling
  - \* an activity of collecting performance data from a running application that may be intrusive on application performance responsiveness or throughput
- \* Performance Tuning
  - \* an activity of changing tuneables, source code, or configuration attribute(s) for the purposes of improving application responsiveness or throughput

# OS Performance Monitoring

- \* CPU

- \* Graphical

- \* gnome-system-monitor, xosview (Unix/Linux)

- \* cpubar (Solaris)

- \* TaskManager, Performance Monitor (Windows)

- \* Text

- \* vmstat, mpstat, pidstat, top

- \* prstat (Solaris)

- \* typeperf (Windows)

# OS Performance Monitoring (cont.)

- \* Memory

- \* vmstat

- \* cpubar (Solaris)

- \* Performance Monitor, typeperf (Windows)

- \* Network

- \* nicstat, typeperf (Windows)

- \* I/O

- \* iostat, iosnoop.d (Solaris)

- \* Other

- \* sar (sysstat), kstat (Solaris), cpustat, cputrack

# Performance Profiling - Types of Profilers

- ❖ Method profiler
  - Collects information about method execution times
  - Look for: internal/external method times, frequently called methods/classes etc.
- ❖ Memory profiler
  - Collects information about object creation and/or garbage collection
- ❖ Thread profiler
  - Looks for thread conflict situations

# Performance Profiling - Profilers

## ❖ Method profilers

- ❑ Oracle Solaris Studio Performance Analyzer, HPJMeter, JVisualVM
- ❑ Free: `java -agentlib:hprof=[help][[<option>=<value>, ...], jvmstat, JRockit Runtime Analyzer`
- ❑ Commercial: JProbe, Optimizelt!, JProfiler, YourKit

## ❖ Memory profilers

- ❑ Oracle Solaris Studio Performance Analyzer
- ❑ HPJMeter
- ❑ JVisualVM

## ❖ Thread profilers

- ❑ JVisualVM
- ❑ Google's JChord, Java Race Finder, Mutability Detector, YourKit

# A few things about the application to optimise

- \* A soft real-time Java application based on NetBeans RCP
- \* Displays plots and tracks on a digital map
- \* Requirement to run on SunBlade 1500 (1 CPU 1.062 GHz UltraSPARC, 1 GB RAM)

# Steps executed

- 1) Research of root causes
  - \* Analysis of GC impact
  - \* Verification of lock contention
  - \* Hotspots in application code
- 2) Profile main branch application
- 3) Identification of hotspots
- 4) Optimisation of hotspots
- 5) Performance measurement with & without changes
- 6) Integration of changes
- 7) Restart from point 1

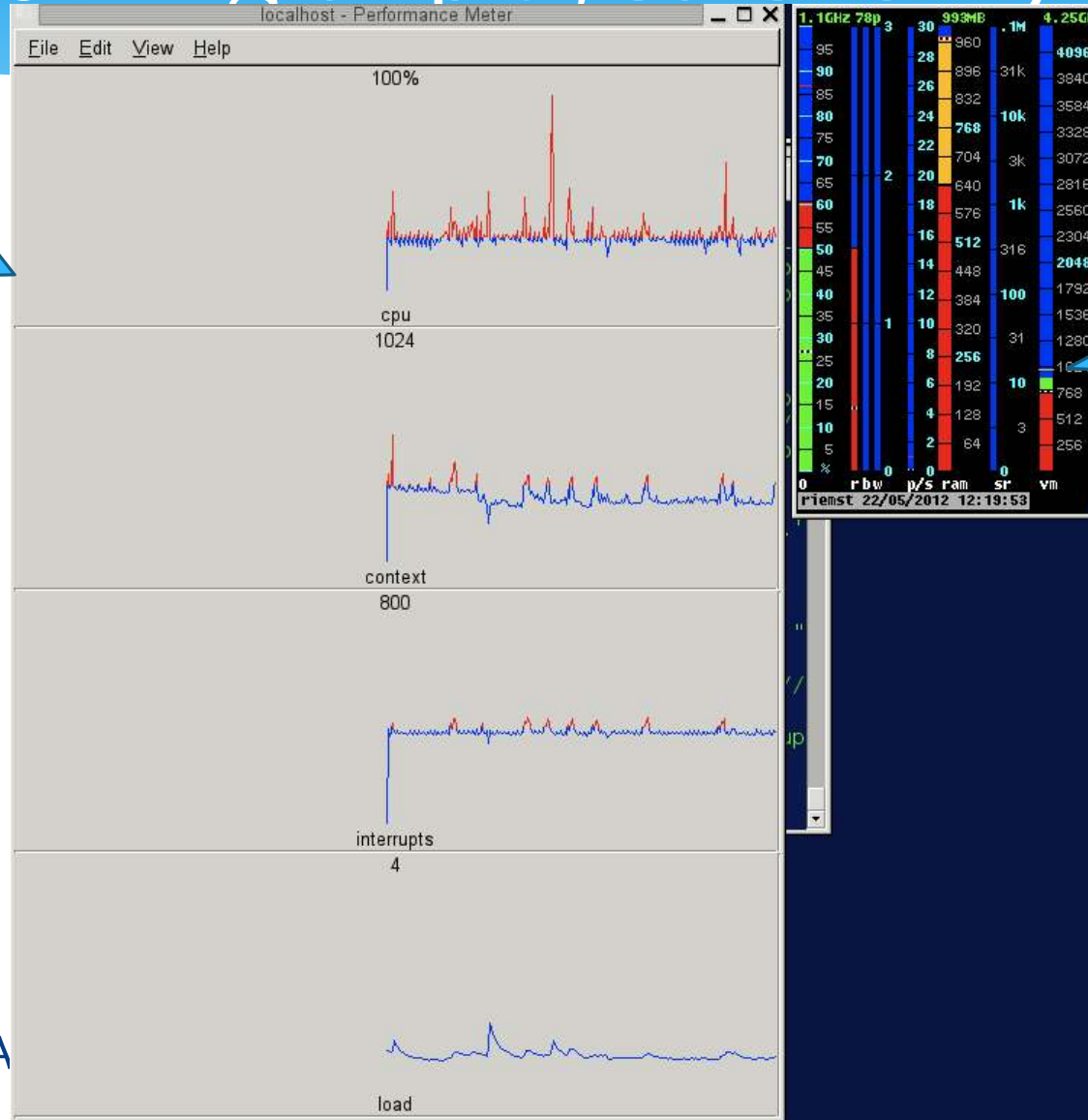


# CPU Performance Monitoring

# Monitoring Results of Main branch

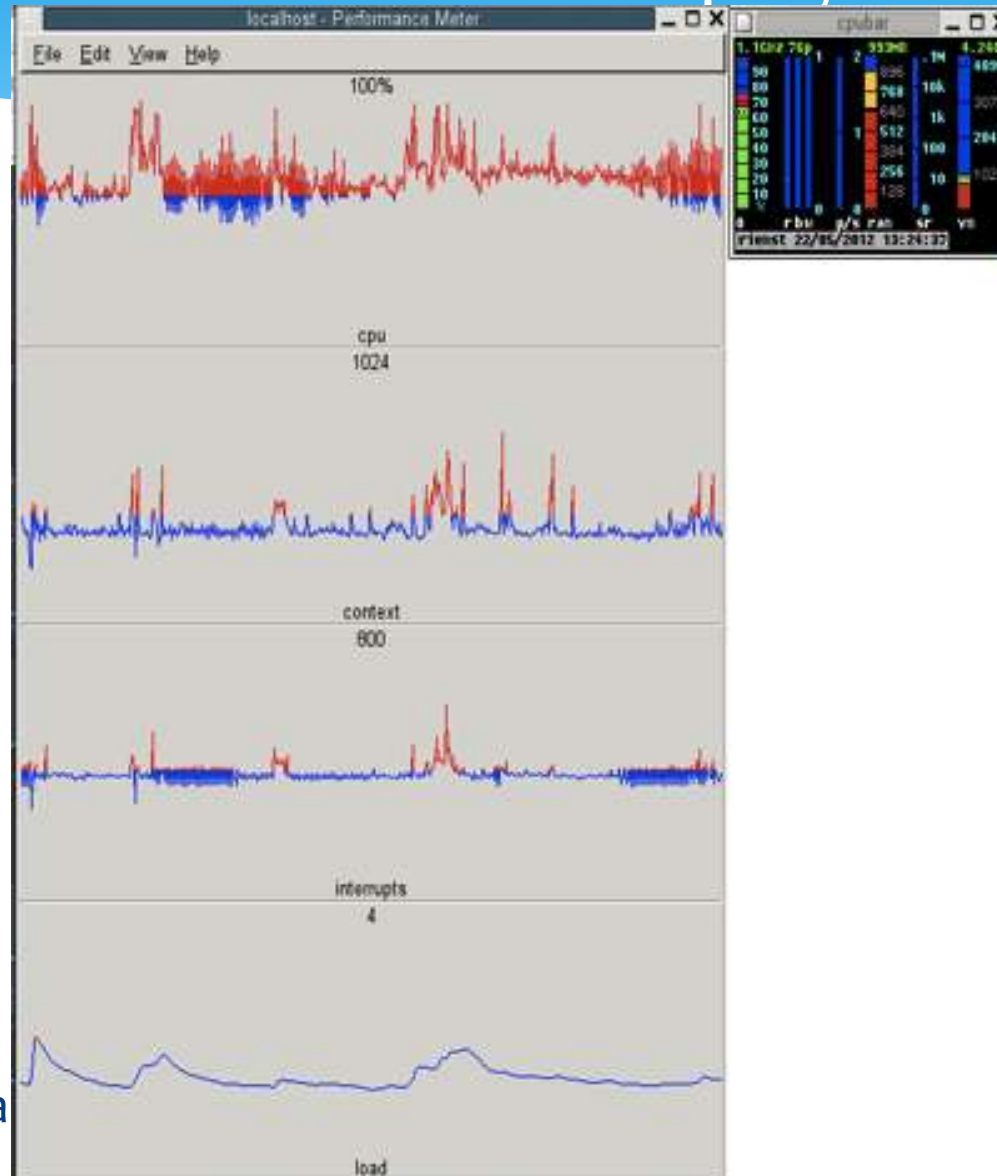
## Nothing displayed CPU: 27%

gnome-system-monitor



cpubar

# Monitoring Results of Main branch Plots & tracks with labels displayed CPU: 64%



# CPU Performance Profiling

# Identification of Hotspot 1

Exclusive & inclusive user  
CPU utilisation metrics

The screenshot shows the VisualVM interface with the 'Functions' tab selected. A red box highlights the top few rows of the function list, which are sorted by 'User CPU (%)'. The top row shows 100.00% user CPU for the root function. The second row shows 8.92% user CPU for the system. The third row, highlighted in blue, shows 8.91% user CPU for the function `java.util.concurrent.ConcurrentLinkedQueue.size()`. The fourth row shows 5.44% user CPU for `sun.awt.image.CopyOnDemand`.

On the right side, the 'Selected Object' panel is visible, showing details for the selected object. Below it, the 'Metrics for Selected Object' table is displayed, comparing 'Exclusive' and 'Inclusive' metrics.

	Exclusive	Inclusive
User CPU:	8.326 (31.23%)	9.096 (34.12%)
Wait:	0. (0.%)	0. (0.%)
Total Thread:	13.630 (0.97%)	14.600 (1.04%)
System CPU:	0. (0.%)	0. (0.%)
Wait CPU:	4.953 (18.27%)	5.154 (19.01%)
User Lock:	0.350 (0.07%)	0.350 (0.07%)
Text Page Fault:	0. (0.%)	0. (0.%)
Data Page Fault:	0. (0.%)	0. (0.%)
Other Wait:	0. (0.%)	0. (0.%)

# Identification of Hotspot 2

The screenshot displays the Oracle Solaris Studio Performance Analyzer interface. The main window shows a call tree with columns for User CPU (%), CPU (%), and Name. A red box highlights a specific entry in the call tree, which is selected in the right-hand pane. The right-hand pane shows the 'Selected Object' details, including Name, PC Address, Size, Source File, Object File, Load Object, Mangled Name, and Aliases. Below this, the 'Metrics for Selected Object' section provides a table of performance metrics.

	Exclusive	Inclusive
User CPU:	0. ( 0. %)	10.998 (13.75%)
Wait:	0. ( 0. %)	0. ( 0. %)
Total Thread:	0. ( 0. %)	24.527 ( 0.58%)
System CPU:	0. ( 0. %)	0.040 ( 1.72%)
Wait CPU:	0. ( 0. %)	12.339 (15.17%)
User Lock:	0. ( 0. %)	1.151 ( 0.08%)
Text Page Fault:	0. ( 0. %)	0. ( 0. %)
Data Page Fault:	0. ( 0. %)	0. ( 0. %)
Other Wait:	0. ( 0. %)	0. ( 0. %)



# Details of Hotspot 2

The screenshot displays the HotSpot VM GUI. The main window is titled 'User CPU' and shows a call tree with the following entries:

- 93.49 [redacted] track.AbstractTrack.updateProperties (java.lang.Object)
- 6.31 [redacted] track.AbstractTrack.a (java.lang.Object, [redacted] ppi.dataobject.e)
- 0.51 [redacted] track.AbstractTrack.a (java.lang.Object, java.lang.Class, [redacted] ..services.PropertiesC...
- 43.74 [redacted] ..dex.track.properties.BasicPropertiesProvider.getPropertyValue (java.lang.Object, [redacted] ppi.dat...
- 29.25 [redacted] track.properties.TrackPropertiesProvider.getPropertyValue (java.lang.Object, [redacted] ppi...
- 6.71 [redacted] track.AbstractTrack.b (java.lang.Object, [redacted] ppi.dataobject.e)
- 4.35 [redacted] track.properties.BasicPropertiesProvider.getPropertyValue (java.lang.Object, [redacted] ppi.da...
- 2.66 [redacted] ppi.services.PropertiesCollector.getPropertyProvider (java.lang.Class)
- 2.04 java.lang.Class.cast (java.lang.Object)
- 0.73 java.util.Collections\$UnmodifiableCollection1.next ()
- 0.76 java.util.Collections\$UnmodifiableCollection1.hasNext ()
- 0.09 java.beans.PropertyChangeSupport.firePropertyChange (java.beans.PropertyChangeEvent)
- 0.09 java.util.AbstractList\$Itr.hasNext ()
- 0.09 java.util.Collections\$UnmodifiableCollection1.iterator ()
- 0.09 [redacted] track.properties.TrackPropertiesProvider.getProperties ()
- 0.09 [redacted] track.properties.TrackAssociationPropertiesProvider.getPropertyValue (java.lang.Object, [redacted] ..services...
- 0. java.util.AbstractList\$Itr.next ()

The right-hand pane shows the 'Selected Object' details:

- Name: [redacted] track.A
- PC Address: 11:0x00000000
- Size: 4294967295
- Source File: [redacted] track/AbstractTrack.java
- Object File: [redacted] track.AbstractTrack
- Load Object: <jclasses>
- Mangled Name: [redacted] track.A
- Aliases:

Below the object details is a table for 'Metrics for Selected Object':

	Exclusive	Inclusive
User CPU:	0.040 ( 0.08%)	11.268 (14.09%)
Wait:	0. ( 0. %)	0. ( 0. %)
Total Thread:	0.060 ( 0.00%)	24.737 ( 0.59%)
System CPU:	0. ( 0. %)	0.040 ( 1.72%)
Wait CPU:	0. ( 0. %)	12.219 (15.02%)
User Lock:	0. ( 0. %)	1.211 ( 0.08%)
Text Page Fault:	0. ( 0. %)	0. ( 0. %)
Data Page Fault:	0. ( 0. %)	0. ( 0. %)
Other Wait:	0. ( 0. %)	0. ( 0. %)

# Details of Hotspot 2 (cont.)

The screenshot shows the Oracle Java HotSpot VM Performance Analyzer (JProfiler) interface. The main window displays a call stack with the top frame highlighted in blue. The right-hand pane displays the 'Selected Object' details and a 'Metrics for Selected Object' table.

**Selected Object:**

- Name: java.lang.String.format(java.lang.String, java.la
- PC Address: 11:0x000039ff
- Size: 16
- Source File: java/lang/String.java
- Object File: /tmp/analyser.String.class.1283
- Load Object: <classes>
- Mangled Name: java.lang.String.format
- Aliases:

**Metrics for Selected Object:**

	Exclusive	Inclusive
User CPU:	0.010 ( 0.014)	8.426 (10.534)
Wait:	0. ( 0. %)	0. ( 0. %)
Total Thread:	0.010 ( 0.004)	17.792 ( 0.424)
System CPU:	0. ( 0. %)	0.030 ( 1.294)
Wait CPU:	0. ( 0. %)	8.466 (10.414)
User Lock:	0. ( 0. %)	0.871 ( 0.064)
Text Page Fault:	0. ( 0. %)	0. ( 0. %)
Data Page Fault:	0. ( 0. %)	0. ( 0. %)
Other Wait:	0. ( 0. %)	0. ( 0. %)



# Details of Hotspot 2 (cont.)

The screenshot displays the Oracle Solaris Studio Performance Analyzer interface. The main window shows a call tree with the following entries:

User CPU (%)	Name
50.00	[redacted] track.properties.MessagePropertiesProvider.getPropertyValue(soclib.dataelements.b, [redacted].common.p...
15.91	[redacted] comon.util.m.e(double, java.lang.String)
12.59	[redacted] comon.util.m.b(double, java.lang.String)
12.35	[redacted] track.properties.TrackPropertiesProvider.getPropertyValue([redacted] track.link16...
7.36	[redacted] plots.PlotSer.a([redacted] plots.messages.gfa, int, java.lang.String)
0.83	edlib.link16.datafields.Mode_III_Code.b(int)
0.83	edlib.link16.datafields.Mode_II_Code.b(int)
0.12	[redacted] comon.ppi.statusser.b.run()

The right-hand pane shows the 'Selected Object' details for the selected method:

**Selected Object:**

- Name: java.lang.String.format(java.lang.String, java.la...
- PC Address: 1110x00039ff
- Size: 16
- Source File: java/lang/String.java
- Object File: /tmp/analyser.String.class.1203
- Load Object: <classes>
- Mangled Name: java.lang.String.format
- Aliases:

**Metrics for Selected Object:**

	Exclusive	Inclusive
User CPU:	0.010 (0.01%)	8.426 (10.53%)
Wait:	0. (0. %)	0. (0. %)
Total Thread:	0.010 (0.00%)	17.792 (0.42%)
System CPU:	0. (0. %)	0.030 (1.29%)
Wait CPU:	0. (0. %)	8.466 (10.41%)
User Lock:	0. (0. %)	0.871 (0.06%)
Text Page Fault:	0. (0. %)	0. (0. %)
Data Page Fault:	0. (0. %)	0. (0. %)
Other Wait:	0. (0. %)	0. (0. %)

# Identification of Hotspot 2 (cont.)

The screenshot displays the Oracle Solaris Studio Performance Analyzer interface. The main window shows a call tree with columns for User CPU (%), CPU V (%), and Name. A red box highlights a specific entry in the call tree. The right-hand pane shows the 'Selected Object' details, including its name, PC address, size, source file, object file, load object, mangled name, and aliases. Below this, a table provides metrics for the selected object, comparing exclusive and inclusive values for User CPU, Wait, Total Thread, System CPU, Wait CPU, User Lock, Text Page Fault, Data Page Fault, and Other Wait.

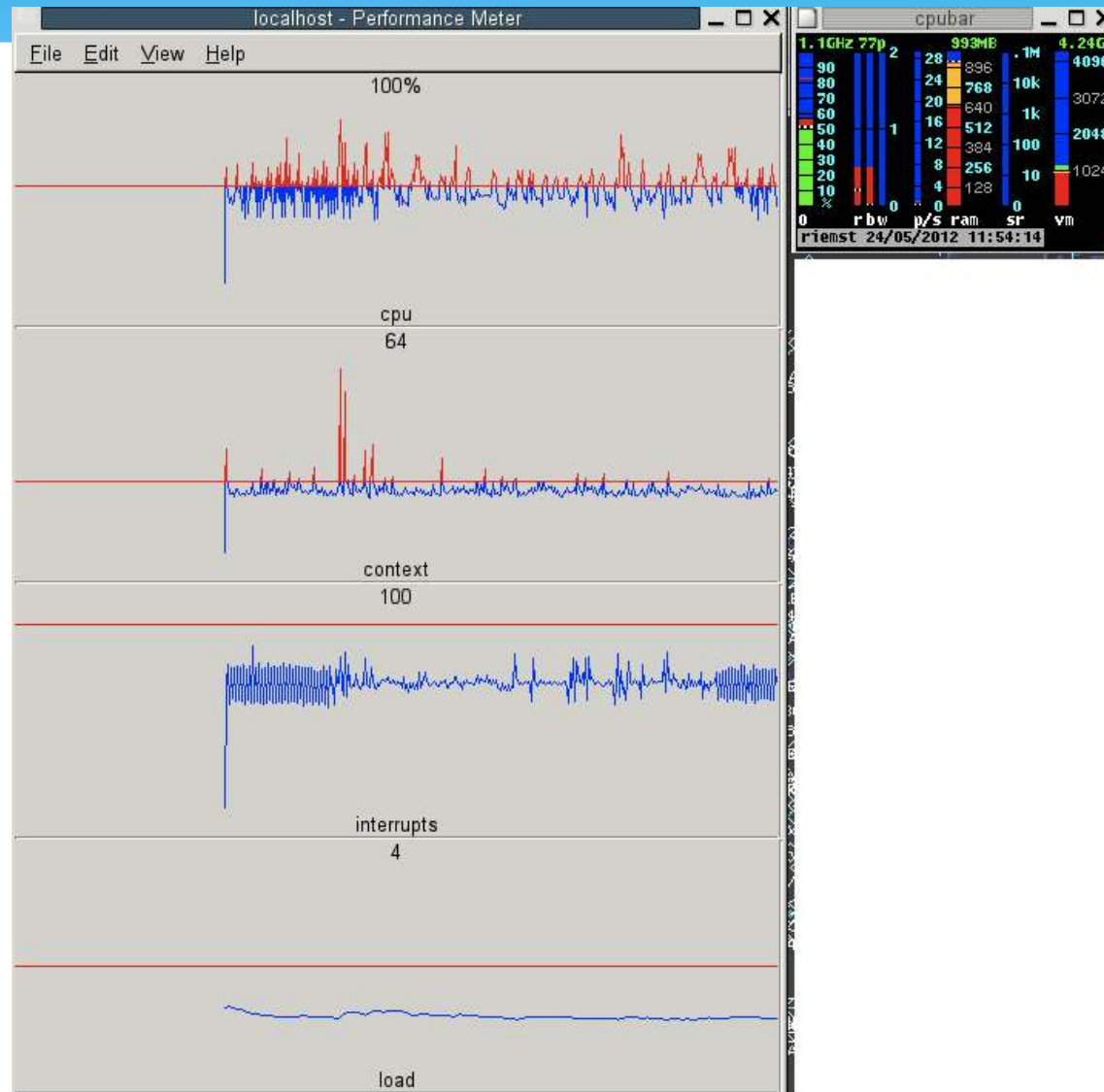
Metrics for Selected Object:		
	Exclusive	Inclusive
User CPU:	0.010 ( 0.014)	8.426 (10.534)
Wait:	0. ( 0. %)	0. ( 0. %)
Total Thread:	0.010 ( 0.004)	17.792 ( 0.424)
System CPU:	0. ( 0. %)	0.030 ( 1.294)
Wait CPU:	0. ( 0. %)	8.466 (10.414)
User Lock:	0. ( 0. %)	0.871 ( 0.064)
Text Page Fault:	0. ( 0. %)	0. ( 0. %)
Data Page Fault:	0. ( 0. %)	0. ( 0. %)
Other Wait:	0. ( 0. %)	0. ( 0. %)

# CPU Performance Tuning



# Performance after the changes

## CPU: 50%



# Profiling comparison 1

Oracle Solaris Studio Performance Analyzer | MainBranch-Tracks.er [java] ...

File View Help

View Mode User Find Text:

Functions Callers-Callees Call Tree Source Disassembly Timeline Experiments

MainBranch-Tracks.er User CPU	NewQueue NewMap-Tracks.er User CPU	MainBranch-Tracks.er User CPU	NewQueue NewMap-Tracks.er User CPU	Name
▼ (%)	(%)	(%)	(%)	
100.00	100.00	100.00	100.00	<Total>
8.92	8.86	8.92	8.86	<JVM-System>
8.91	0.	9.76	0.	java.util.concurrent.ConcurrentLinkedQueue.size()
5.64	8.20	5.64	8.20	mLib_ImageCopy_na
3.92	2.94	8.04	6.81	java.lang.StrictMath.pow(double, double)
3.80	3.74	3.92	3.79	__ieee754_pow
2.01	2.54	2.01	2.54	mLib_v_ImageClear_s32_1
1.94	2.30	15.06	14.24	com.luciad.internal.transformation.i.a(com.luciad.shape.y, com.luciad
1.63	1.78	11.61	10.48	com.luciad.projection.TLcdLambertConformal.a(com.luciad.shape.y, com.
1.30	1.27	1.30	1.27	plots.AbstractPlot.getBounds()
1.28	1.14	4.63	5.44	sun.java2d.loops.DrawLine.DrawLine(sun.java2d.SunGraphics2D, sun.java
1.20	1.01	26.57	26.50	common.ppi.model.h.a(int, com.luciad.util.
0.98	0.93	0.98	0.93	java.lang.Object.clone()
0.95	0.71	0.95	0.71	com.luciad.util.Y.d(double)
0.91	0.87	0.98	0.94	common.ppi.layer.Efc.accept(java.lang.Cbje
0.88	1.06	0.88	1.06	java.lang.String.charAt(int)
0.84	0.74	0.86	0.75	com.luciad.view.gxy.arfa.a(double, double, java.awt.Point)
0.83	0.04	0.86	0.06	java.util.concurrent.ConcurrentLinkedQueue.succ(java.util.concurrent.
0.76	0.93	15.70	15.01	com.luciad.transformation.TLcdGeodetic2Grid.a(com.luciad.shape.y, com
0.75	1.07	0.75	1.07	<no Java callstack recorded>
0.75	0.98	0.75	0.98	com.luciad.shape.shape3D.n.getSinY()
0.73	0.68	0.73	0.72	java.lang.Integer.equals(java.lang.Object)
0.68	0.52	0.75	0.56	java.util.concurrent.ConcurrentLinkedQueue\$Itr.advance()
0.66	0.77	3.35	4.29	Java_sun_java2d_loops_DrawLine_DrawLine
0.63	0.65	1.39	1.53	com.luciad.view.s.a(java.util.List, java.lang.Object, int, int)



# Profiling comparison 2

The screenshot displays the Oracle Solaris Studio Performance Analyzer interface. The main window shows a function call tree with columns for MainBrancher User CPU, String/Isr User CPU, and Name. The selected object is 'Multiple Selection' with a size of 858993460E. The metrics for the selected object are as follows:

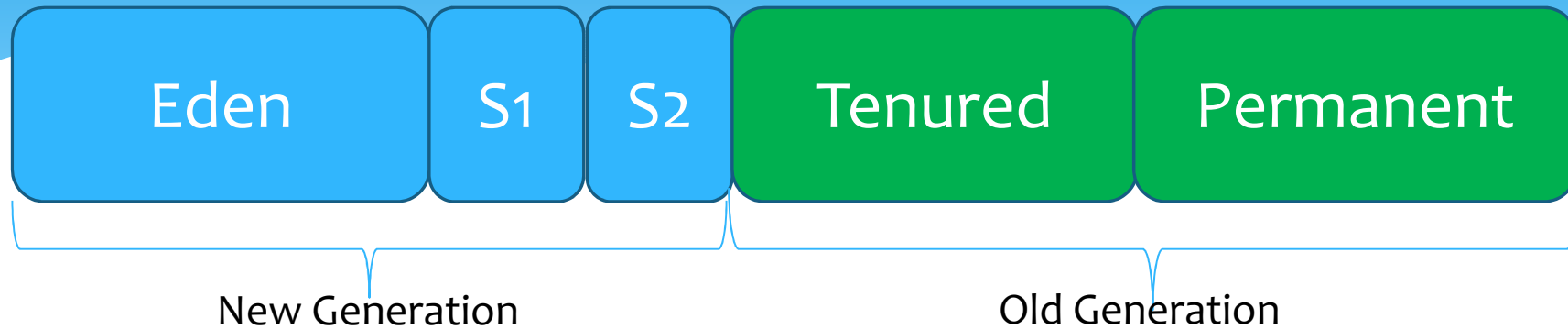
Metrics for Selected Object:		
	Exclusive	Inclusive
User CPU:	0.030 ( 0.10%)	4.563 (15.07%)
Wait:	0. ( 0. %)	0. ( 0. %)
Total Thread:	0.080 ( 0.00%)	12.829 ( 0.56%)
System CPU:	0. ( 0. %)	0.020 ( 1.81%)
Wait CPU:	0.050 ( 0.15%)	6.595 (20.27%)
User Lock:	0. ( 0. %)	1.651 ( 0.14%)
Text Page Fault:	0. ( 0. %)	0. ( 0. %)
Data Page Fault:	0. ( 0. %)	0. ( 0. %)
Other Wait:	0. ( 0. %)	0. ( 0. %)

# Memory Performance Profiling/Tuning

Goal is not to improve the CPU usage but  
to mitigate the impact of the GC on the  
real-time behaviour of the application



# Generational Spaces & Garbage Collectors



New Generation GCs	Old Generation GCs
Copying collector (< Java 5) -XX:+UseSerialGC	Mark-Sweep collector (< Java 5) -XX:+UseSerialGC
Parallel copying collector (≥ Java 5) -XX:+UseParNewGC	Parallel Scavenge MarkSweep collector -XX:+UseParallelOldGC
Parallel scavenge collector (≥ Java 5, >10GB heap space) -XX:+UseParallelGC	Concurrent Mark Sweep collector (≥ Java 6) -XX:+UseConcMarkSweepGC
G1 young generation (≥ Java 7) -XX:+UseG1GC	G1 mixed generation (≥ Java 7) -XX:+UseG1GC

# Recommended Settings

<http://randomlyrr.blogspot.be/2012/03/java-tuning-in-nutshell-part-1.html>

- `-Xmx = -Xms`
- `-XX:PermSize = -XX:MaxPermSize`
- **Use `-Xmn` instead of `-XX:NewSize` and `-XX:MaxNewSize`**
- **Disable adaptive sizing of generations**  
`-XX:-UseAdaptiveSizePolicy`  
**if `-XX:SurvivorRatio` is specified**
- **Use `-XX:+UseConcMarkSweepGC`**
- `40<-XX:CMSInitiatingOccupancyFraction<70`  
**and always use**  
`-XX:+UseCMSInitiatingOccupancyOnly` **with it**

# Application's Runtime arguments

```
run.args.extra =
```

```
-Xms256m -Xmx768m -Xincgc
```

```
-XX:+UseConcMarkSweepGC
```

```
-XX:SurvivorRatio=3
```

```
-XX:-UseAdaptiveSizePolicy
```

```
-XX:CMSInitiatingOccupancyFraction=60
```

```
-XX:+UseCMSInitiatingOccupancyOnly
```

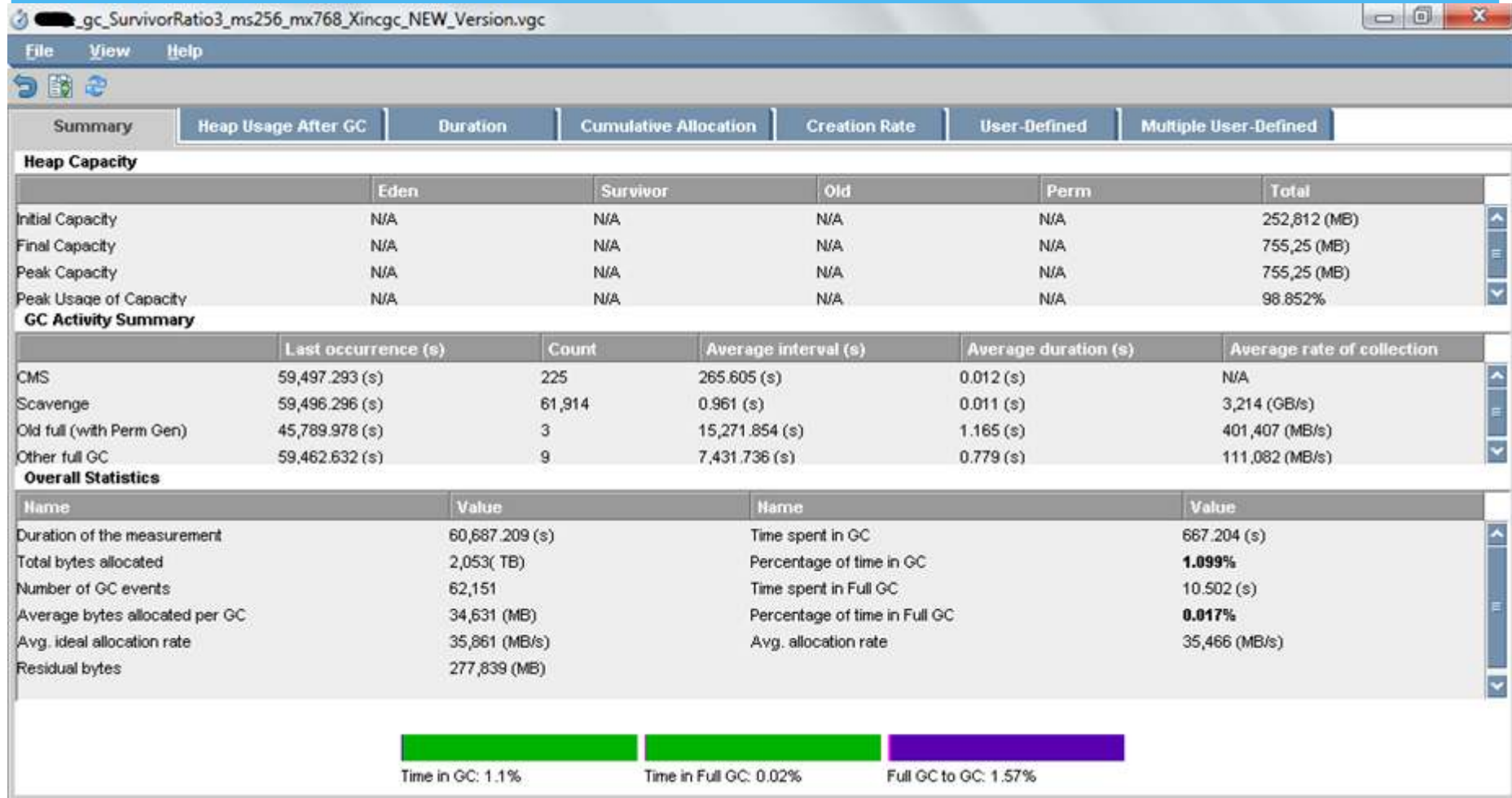
Init heap size

Max heap size

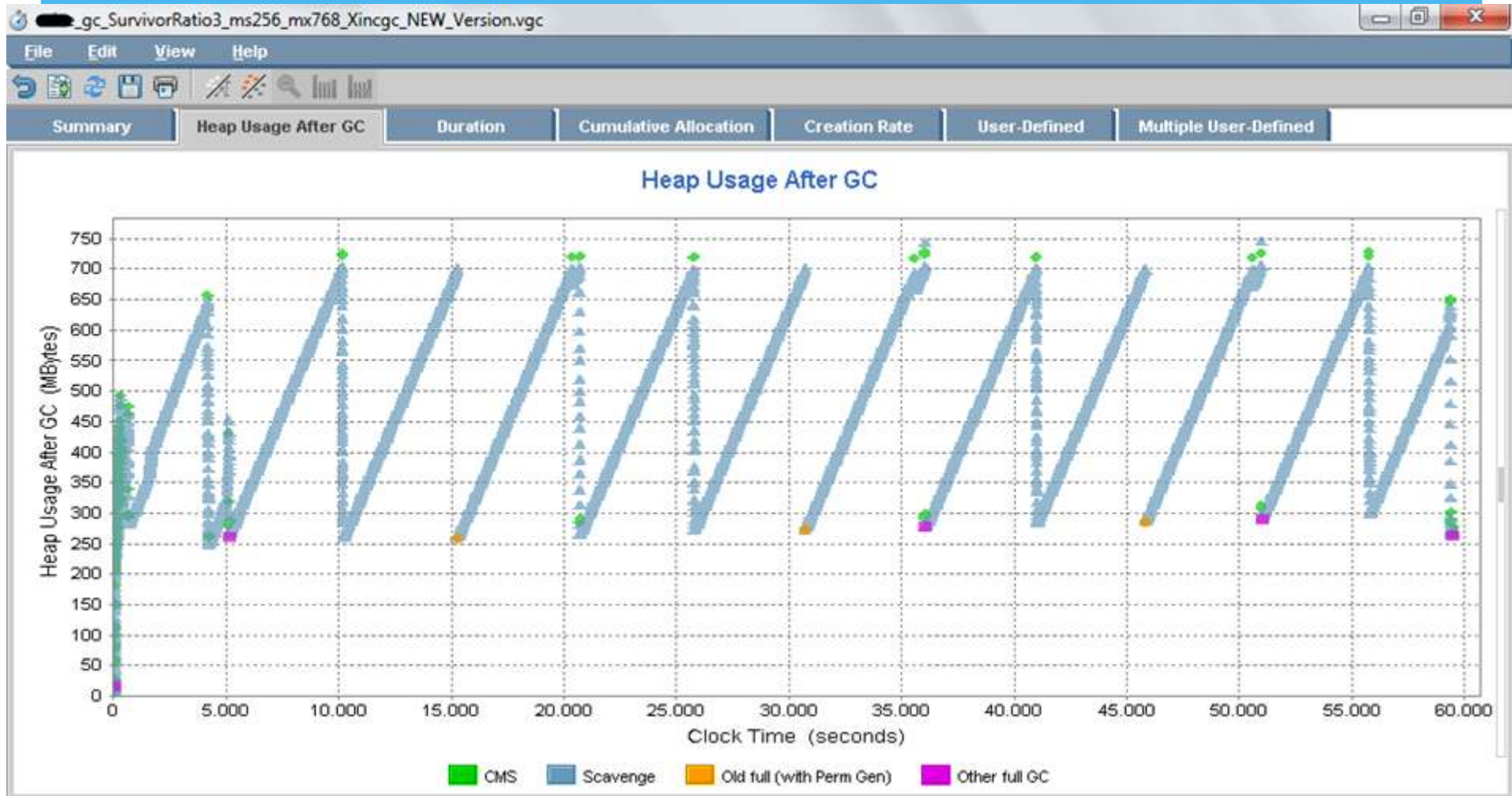
Eden=76,8 Mb,  
S1,S2= 25,6 Mb

CMS: Concurrent MarkSweep

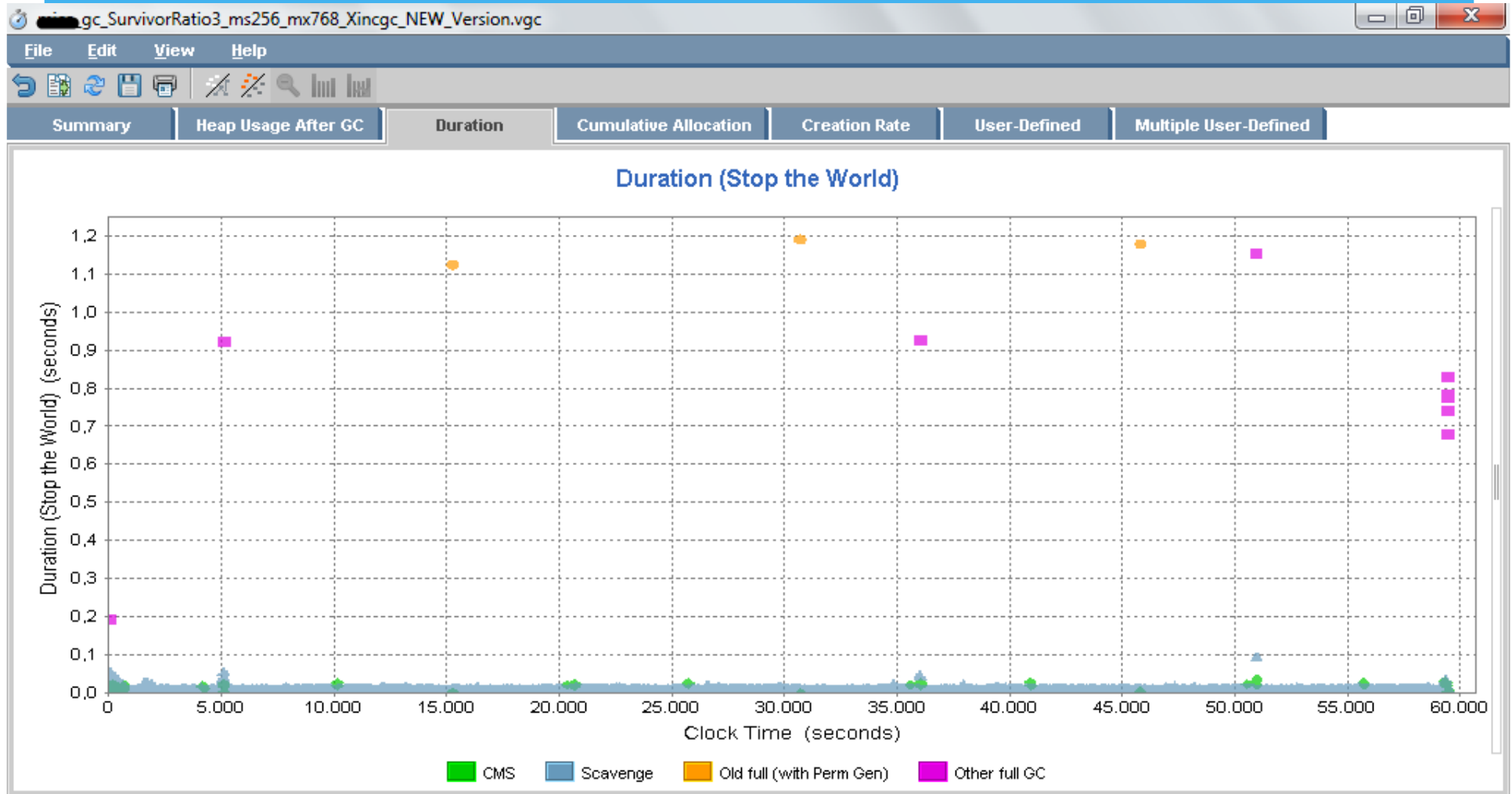
# HPJMeter



# HPJMeter

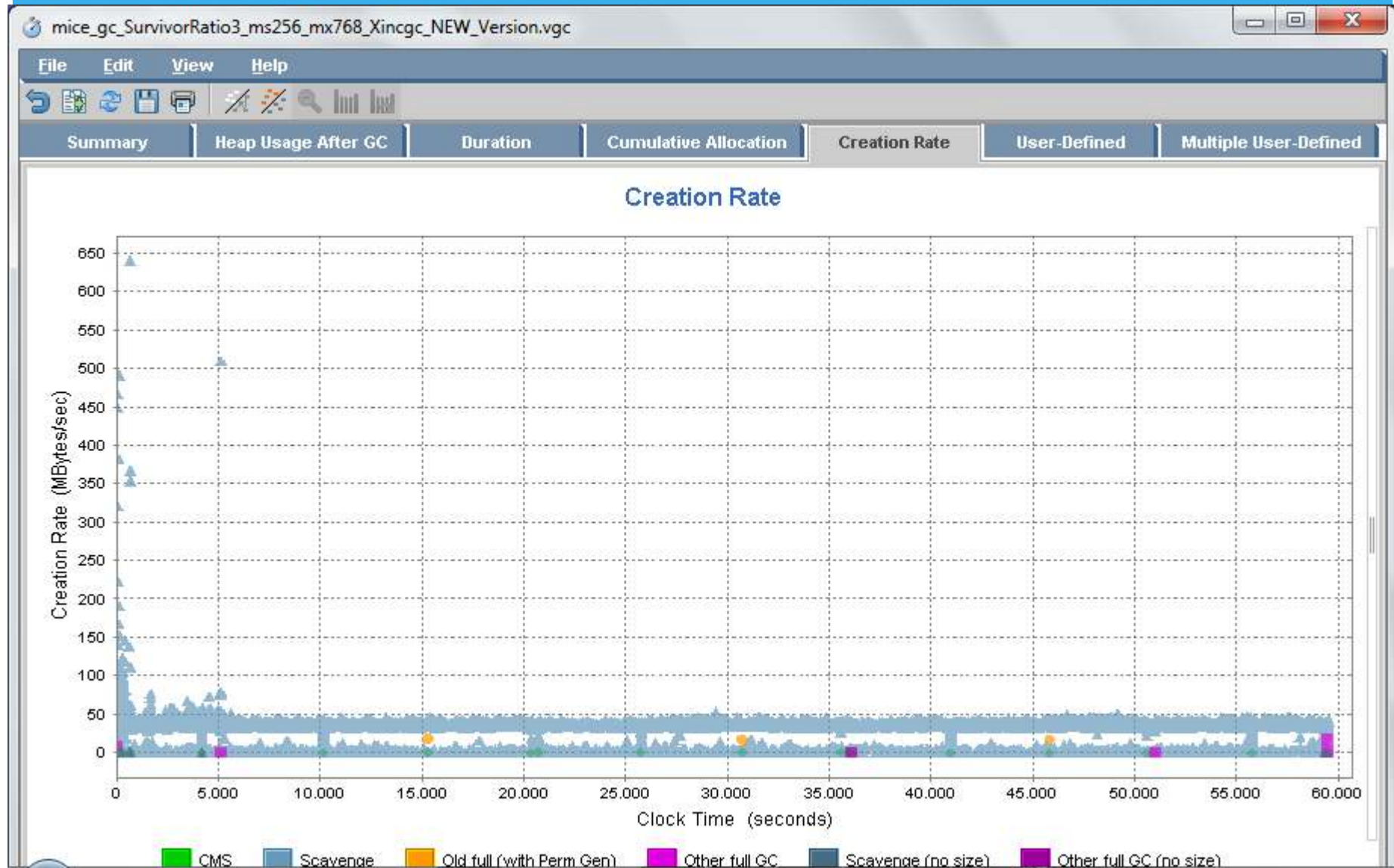


# HPJMeter

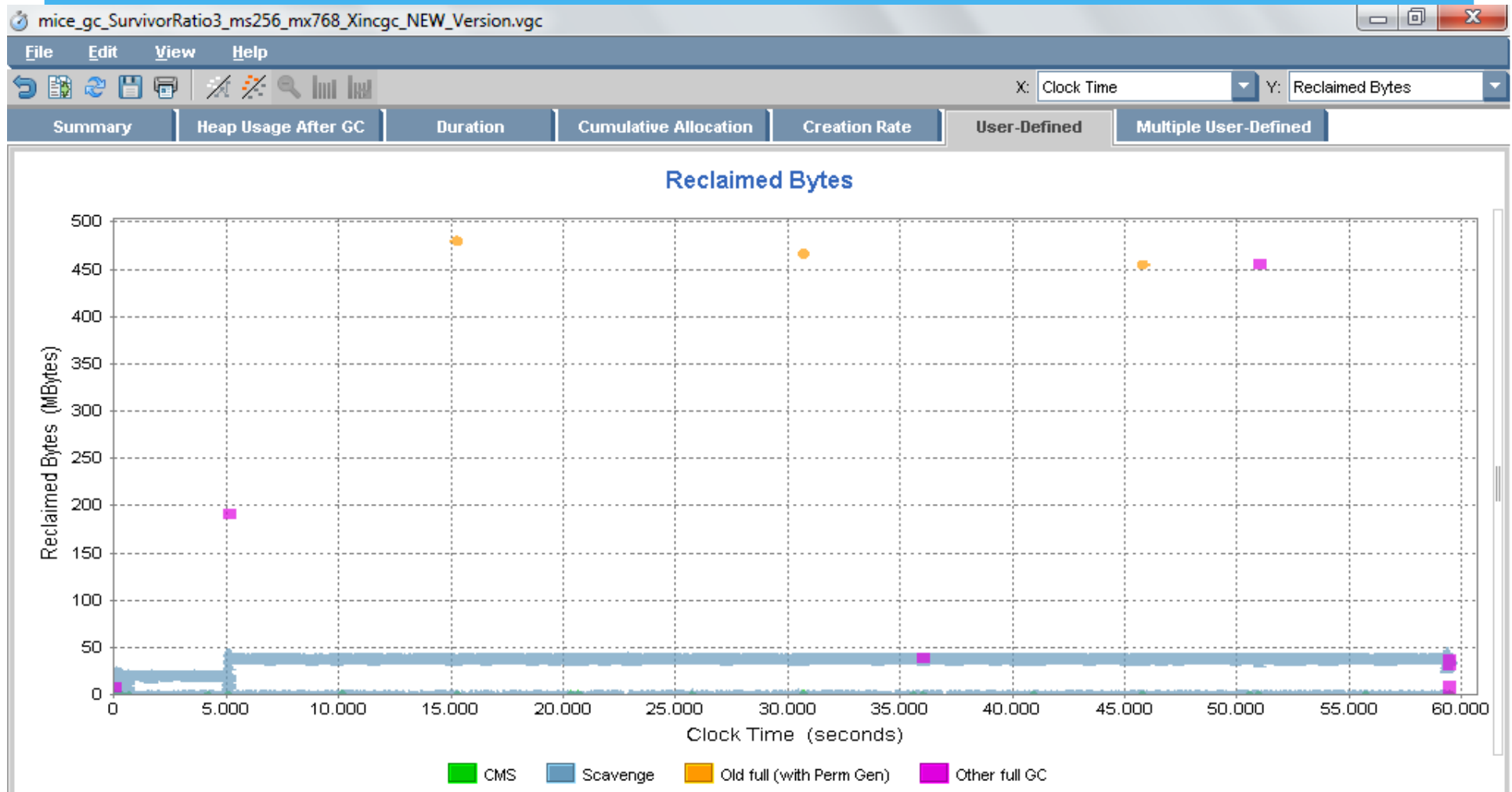




# HPJMeter



# HPJMeter





# Thread Profiling/Tuning ?

# References

- \* Hunt C. & John B. (2011), *Java Performance*, Prentice Hall
- \* Kabutz H. (2011), *Java Master's course slides*
- \* <http://www.youtube.com/watch?v=VGQAL9aUKfs>
- \* <http://randomlyrr.blogspot.be/2012/03/java-tuning-in-nutshell-part-1.html>
- \* <http://www.fasterj.com/articles/oraclecollectors1.shtml>
- \* <http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>

# Questions

