Monitoring Java Web Servers using JMX

Boro Jakimovski

Faculty of Compute Science and Engineering

Ss. Cyril and Methodius University in Skopje

"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER
SCIENCE AND ENGINEERING

Background

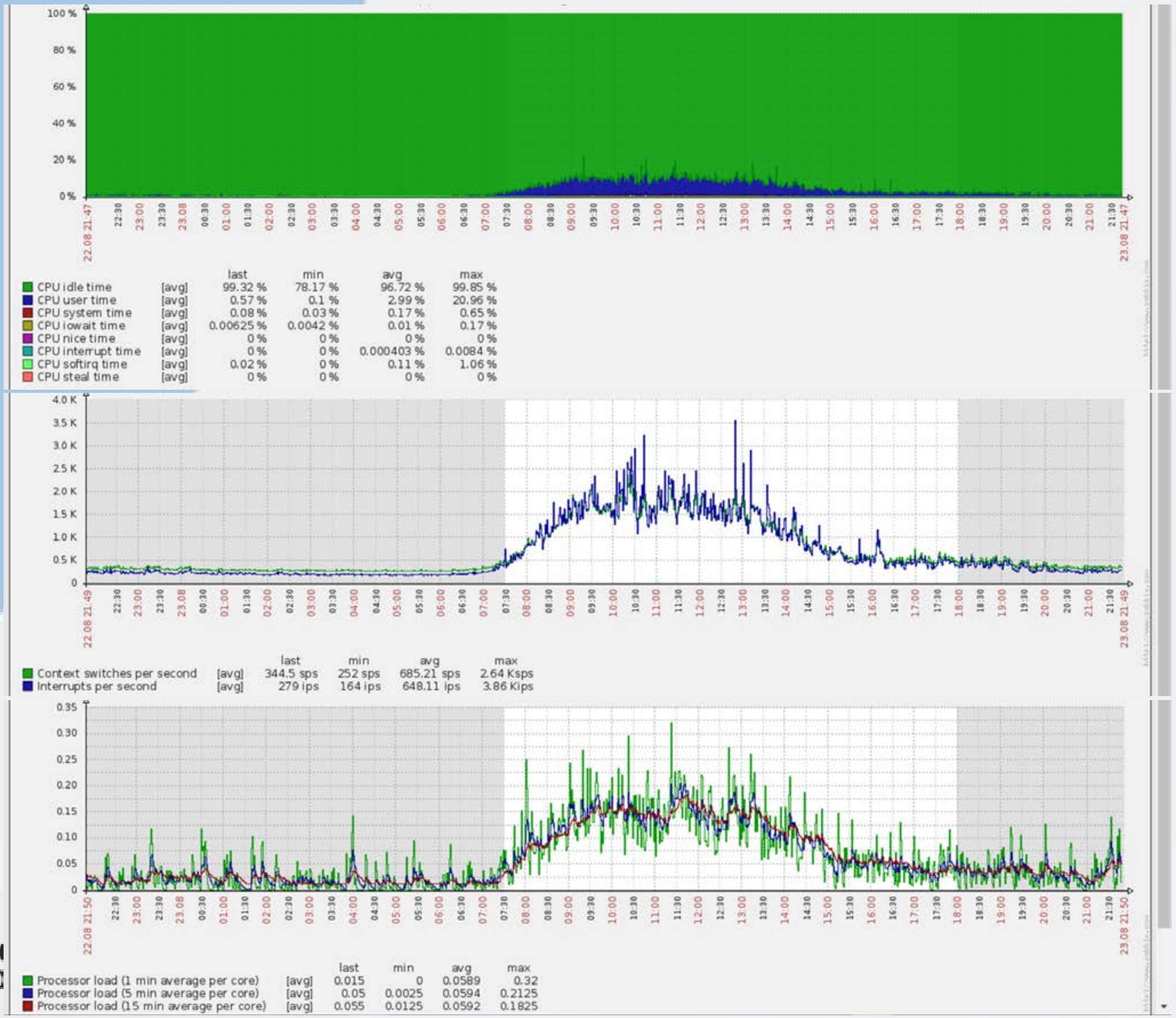
- FCSE Computer Center has been hosting a lot of national services in the past five years
- We support different kinds of software stacks
- Systems have more than 120 requests per second or 1 Mil requests pages per day
- Peaks of up to 500 requests per second, 2000 active sessions and 2 Mil requests per day
- Implemented using a scalable architecture
  - Up to 8 application servers and 3 db servers

Challenge

- The software that is hosted is developed using an agile software process model
- Delivery of new version of the software sometimes on weekly basis
- Due to lack of good stress testing of the developed software, many of the under optimized implementation is not detected during preproduction tests
- This requires of proactive monitoring of the application servers and early detection mechanisms in order to mitigate the problems

How and what to monitor

- Monitoring on OS level
  - CPU
    - Utilization
    - Load
    - Interupts
    - Context switches
  - Memory
  - Network load
- OS Level is not enough
  - Very coarse grain
  - Sometimes the problem is not visible



		last	min	avg	max
🟩 CPU idle time	[avg]	99.32 %	78.17 %	96.72 %	99.85 %
🟦 CPU user time	[avg]	0.57 %	0.1 %	2.99 %	20.96 %
🟥 CPU system time	[avg]	0.08 %	0.03 %	0.17 %	0.65 %
🟨 CPU iowait time	[avg]	0.00625 %	0.0042 %	0.01 %	0.17 %
🟪 CPU nice time	[avg]	0 %	0 %	0 %	0 %
🟦 CPU interrupt time	[avg]	0 %	0 %	0.000403 %	0.0084 %
🟩 CPU softirq time	[avg]	0.02 %	0 %	0.11 %	1.06 %
🟥 CPU steal time	[avg]	0 %	0 %	0 %	0 %

		last	min	avg	max
🟩 Context switches per second	[avg]	344.5 sps	252 sps	685.21 sps	2.64 Ksps
🟦 Interrupts per second	[avg]	279 ips	164 ips	648.11 ips	3.86 Kips

		last	min	avg	max
🟩 Processor load (1 min average per core)	[avg]	0.015	0	0.0589	0.32
🟦 Processor load (5 min average per core)	[avg]	0.05	0.0025	0.0594	0.2125
🟥 Processor load (15 min average per core)	[avg]	0.055	0.0125	0.0592	0.1825

JMX technology

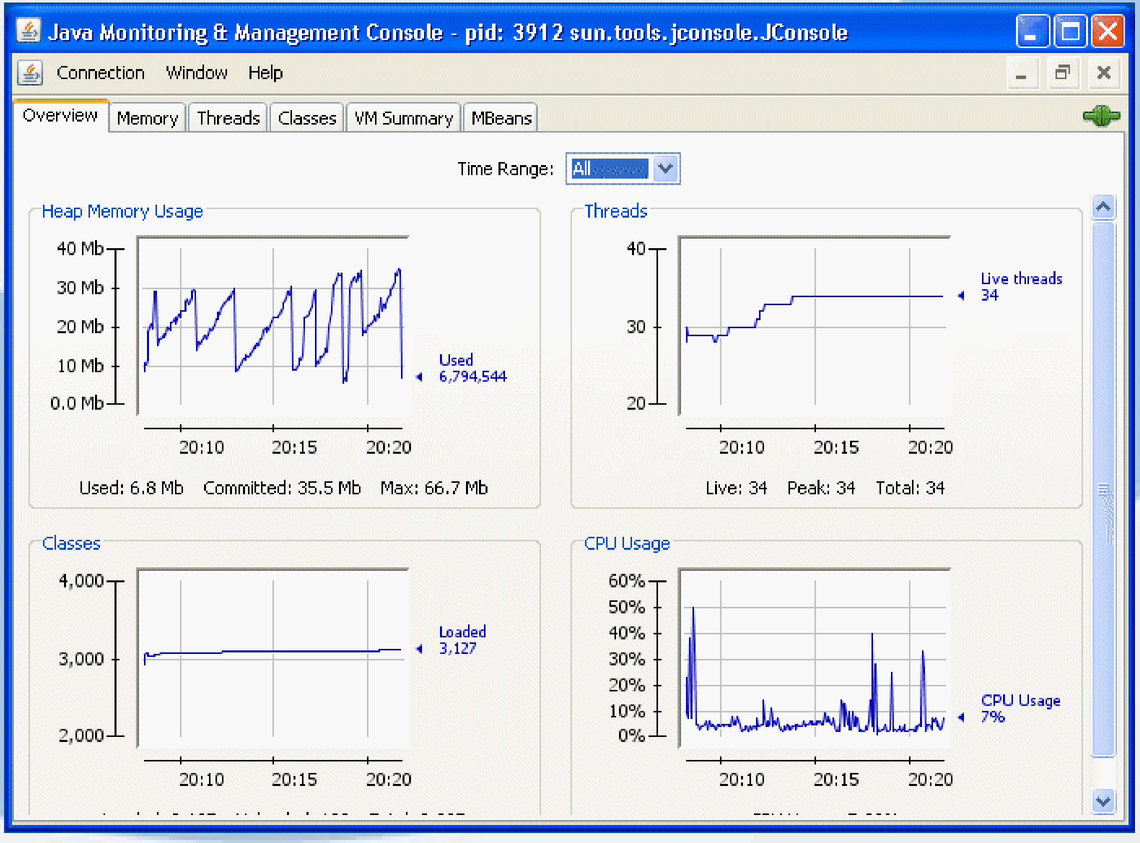
- Monitoring on Java VM level is required
- Java VM enables **Java Management Extensions (JMX)**
- The JMX technology provides a simple, standard way of managing resources such as
  - applications,
  - devices, and
  - services.
- JMX technology is dynamic and can be used to monitor and manage resources as they are created, installed and implemented.
  - instrument Java technology-based applications (Java applications),
  - create smart agents,
  - implement distributed management middleware and managers,
  - and smoothly integrate these solutions into existing management and monitoring systems.
- JMX technology can also monitor and manage the Java Virtual Machine (Java VM).

JMX monitoring of Java VM

- The *platform MXBeans* are a set of MXBeans that is provided with the Java SE platform for monitoring and managing the Java VM and other components of the Java Runtime Environment (JRE).
  - memory
  - threads
  - class-loading system,
  - just-in-time (JIT) compilation system,
  - garbage collector,
- Different monitoring capabilities
  - Jconsole
  - Remote monitoring and management tools

JConsole

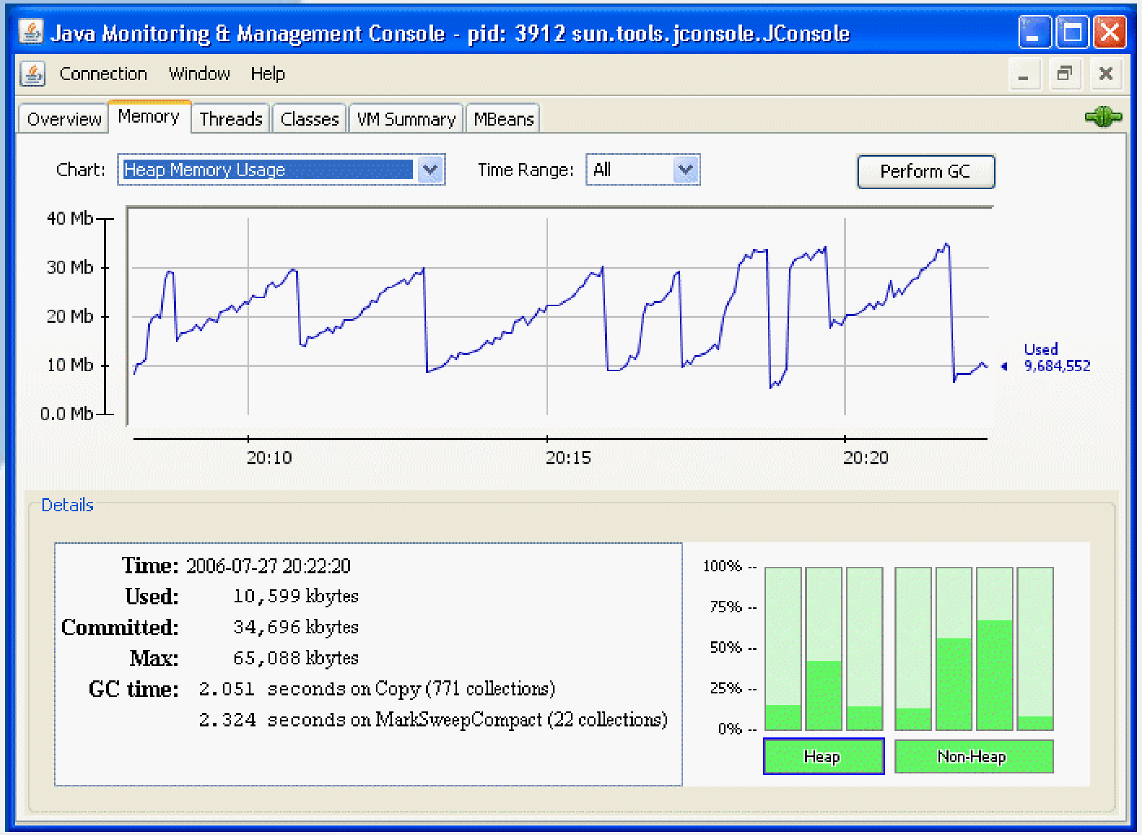
Overview of memory/cpu/threads/classes



Used: 6.8 Mb Committed: 35.5 Mb Max: 66.7 Mb

Live: 34 Peak: 34 Total: 34

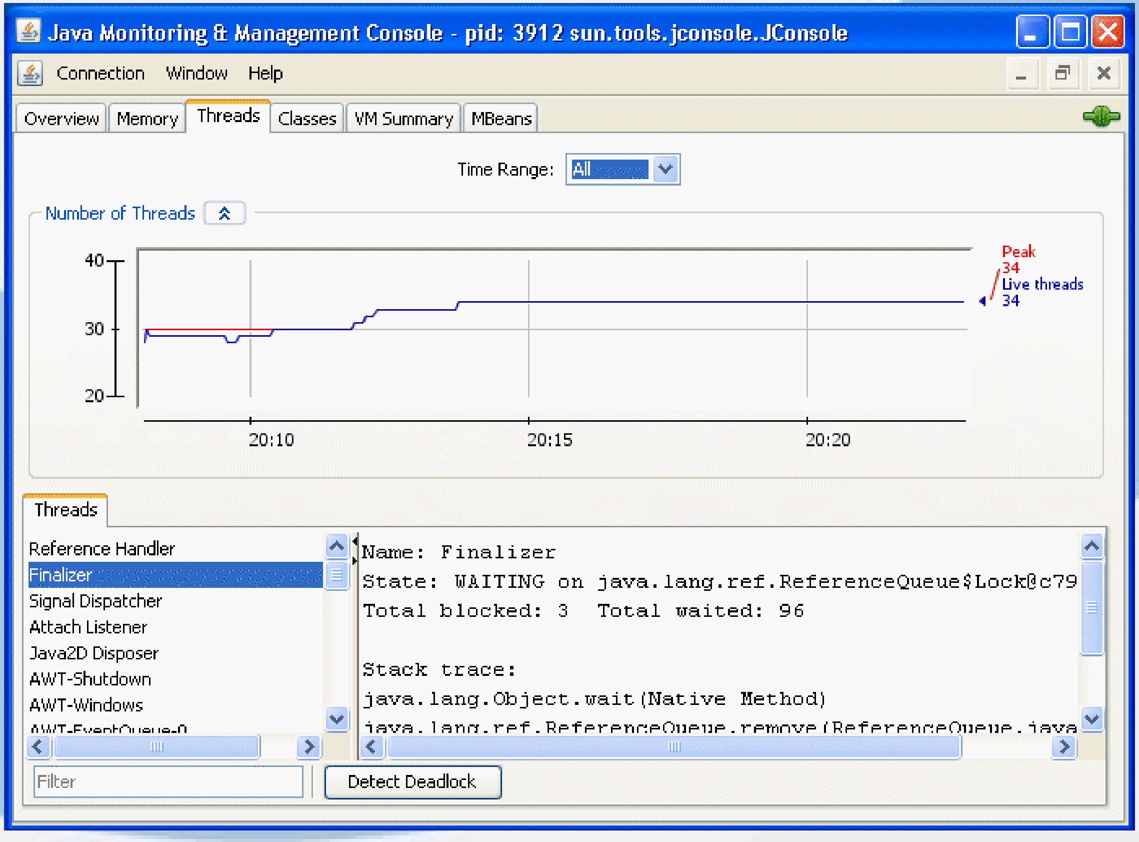
Memory and Garbage collection



Time: 2006-07-27 20:22:20
Used: 10,599 kbytes
Committed: 34,696 kbytes
Max: 65,088 kbytes
GC time: 2.051 seconds on Copy (771 collections)
2.324 seconds on MarkSweepCompact (22 collections)

JConsole

Threads

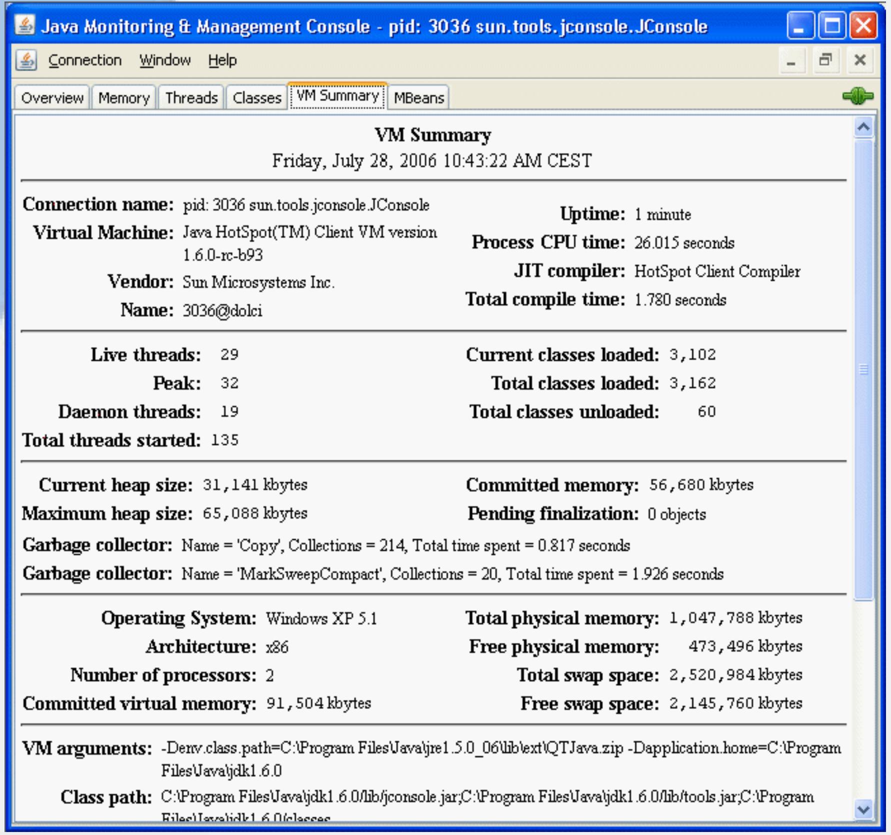


```
Reference Handler
Finalizer
Signal Dispatcher
Attach Listener
Java2D Disposer
AWT-Shutdown
AWT-Windows
AWT-EventQueue-0
```

```
Name: Finalizer
State: WAITING on java.lang.ref.ReferenceQueue$Lock@c79
Total blocked: 3  Total waited: 96

Stack trace:
java.lang.Object.wait(Native Method)
java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java
```

VM Summary


Java Monitoring & Management Console - pid: 3036 sun.tools.jconsole.JConsole
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans

VM Summary
Friday, July 28, 2006 10:43:22 AM CEST

Connection name: pid: 3036 sun.tools.jconsole.JConsole
Virtual Machine: Java HotSpot(TM) Client VM version 1.6.0-rc-b93
Vendor: Sun Microsystems Inc.
Name: 3036@dolci

Uptime: 1 minute
Process CPU time: 26.015 seconds
JIT compiler: HotSpot Client Compiler
Total compile time: 1.780 seconds

Live threads: 29
Peak: 32
Daemon threads: 19
Total threads started: 135

Current classes loaded: 3,102
Total classes loaded: 3,162
Total classes unloaded: 60

Current heap size: 31,141 kbytes
Maximum heap size: 65,088 kbytes
Committed memory: 56,680 kbytes
Pending finalization: 0 objects
Garbage collector: Name = 'Copy', Collections = 214, Total time spent = 0.817 seconds
Garbage collector: Name = 'MarkSweepCompact', Collections = 20, Total time spent = 1.926 seconds

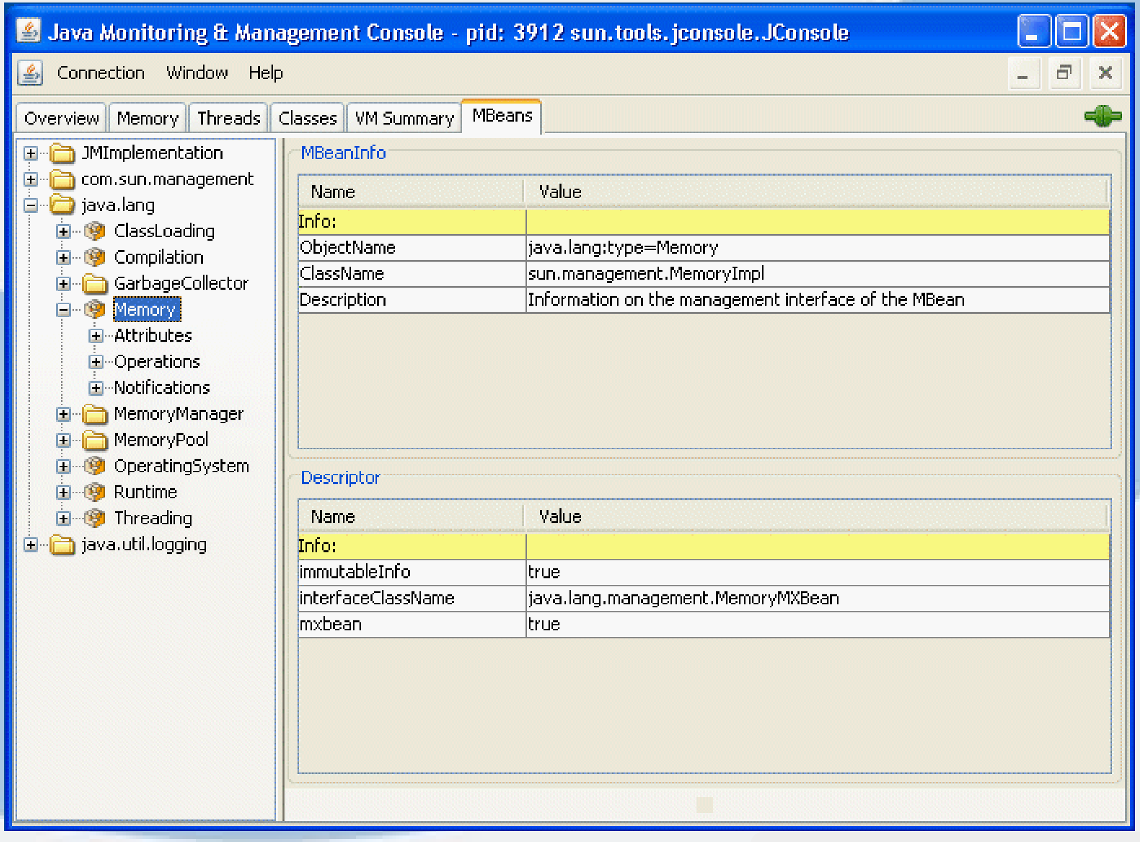
Operating System: Windows XP 5.1
Architecture: x86
Number of processors: 2
Committed virtual memory: 91,504 kbytes

Total physical memory: 1,047,788 kbytes
Free physical memory: 473,496 kbytes
Total swap space: 2,520,984 kbytes
Free swap space: 2,145,760 kbytes

VM arguments: -Denv.class.path=C:\Program Files\Java\jre1.5.0_06\lib\ext\QTJava.zip -Dapplication.home=C:\Program Files\Java\jdk1.6.0
Class path: C:\Program Files\Java\jdk1.6.0/lib/jconsole.jar;C:\Program Files\Java\jdk1.6.0/lib/tools.jar;C:\Program Files\Java\jdk1.6.0/classes

JConsole

Mbeans



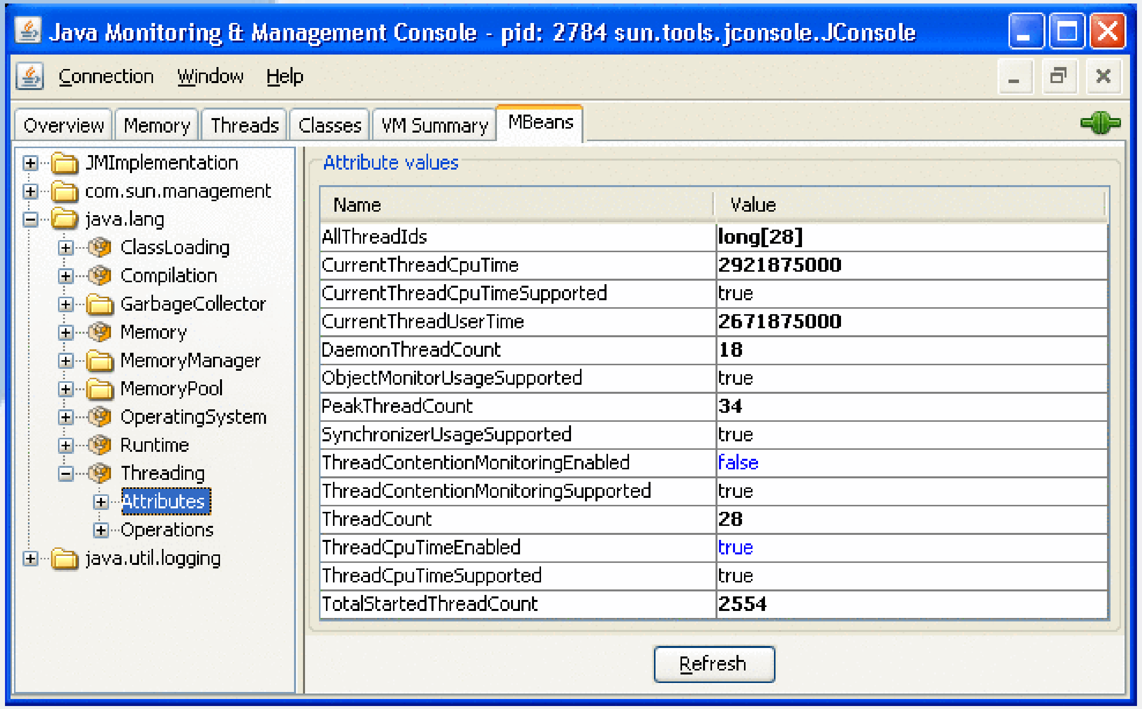
MBeanInfo

Name	Value
Info:	
ObjectName	java.lang:type=Memory
ClassName	sun.management.MemoryImpl
Description	Information on the management interface of the MBean

Descriptor

Name	Value
Info:	
immutableInfo	true
interfaceClassName	java.lang.management.MemoryMXBean
mxbean	true

Read/Write values



Attribute values

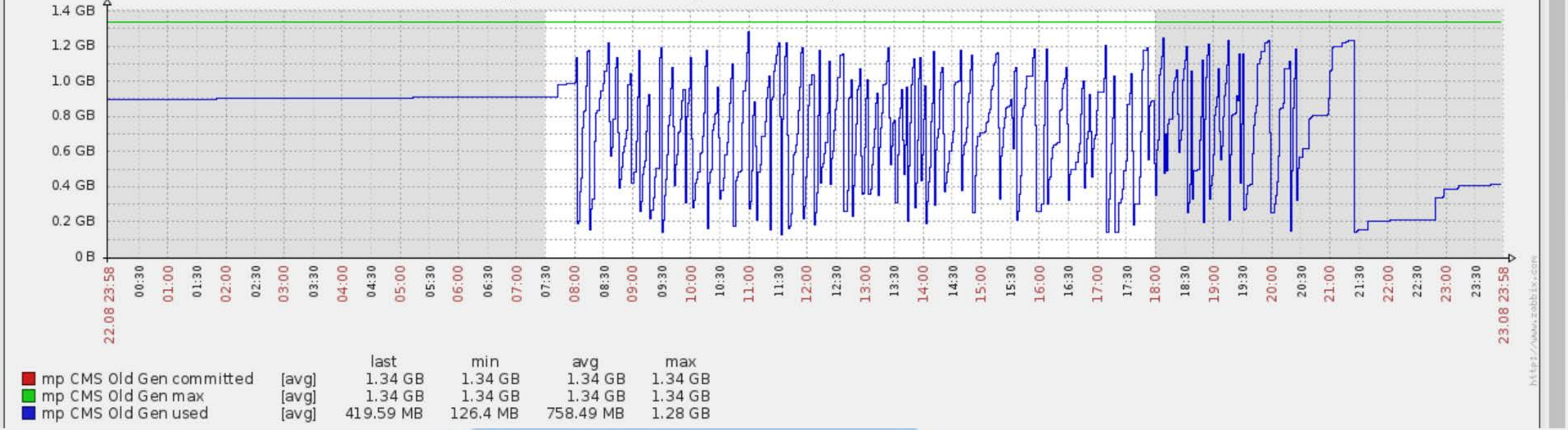
Name	Value
AllThreadIds	long[28]
CurrentThreadCpuTime	2921875000
CurrentThreadCpuTimeSupported	true
CurrentThreadUserTime	2671875000
DaemonThreadCount	18
ObjectMonitorUsageSupported	true
PeakThreadCount	34
SynchronizerUsageSupported	true
ThreadContentionMonitoringEnabled	false
ThreadContentionMonitoringSupported	true
ThreadCount	28
ThreadCpuTimeEnabled	true
ThreadCpuTimeSupported	true
TotalStartedThreadCount	2554

Zabbix java gateway

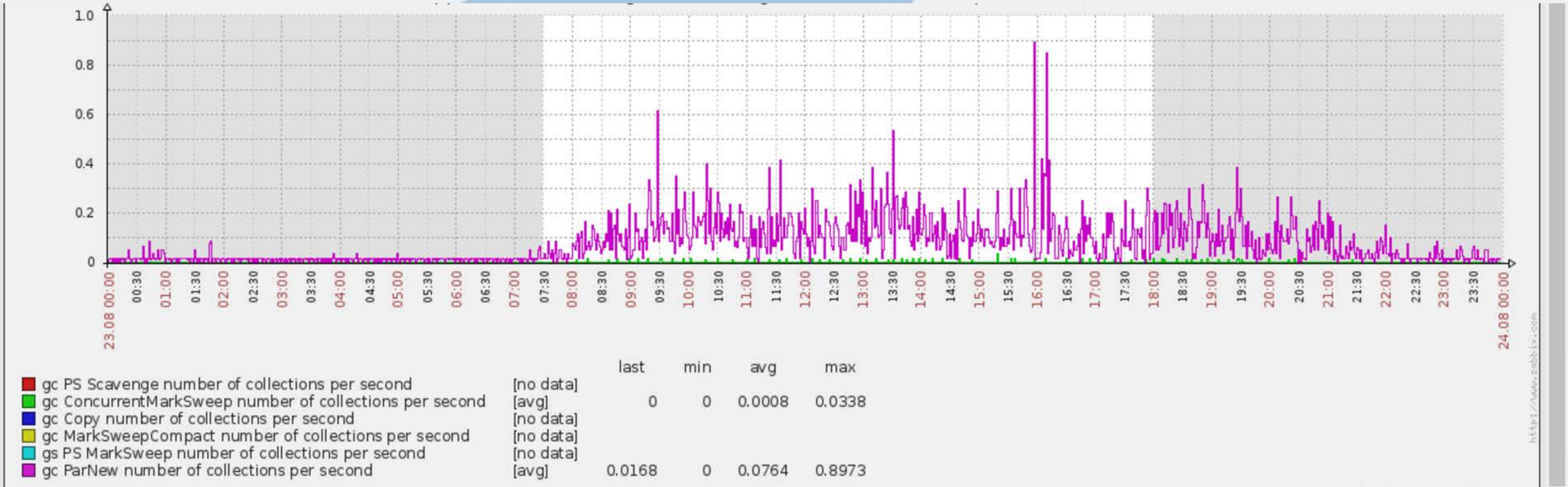
- Monitoring using jConsole is for manual incident handling
- Persistent monitoring needs a more robust monitoring platform
- Zabbix is one of the best open source monitoring projects
  - Enables easy host configuration management
  - Extensible and flexible to address different monitoring data sources
  - Powerful triggering and action engine
- Zabbix supports monitoring using the native client that enables monitoring of OS parameters
- For monitoring of Java services Zabbix has a Java Gateway that uses JMX

Zabbix JMX templates

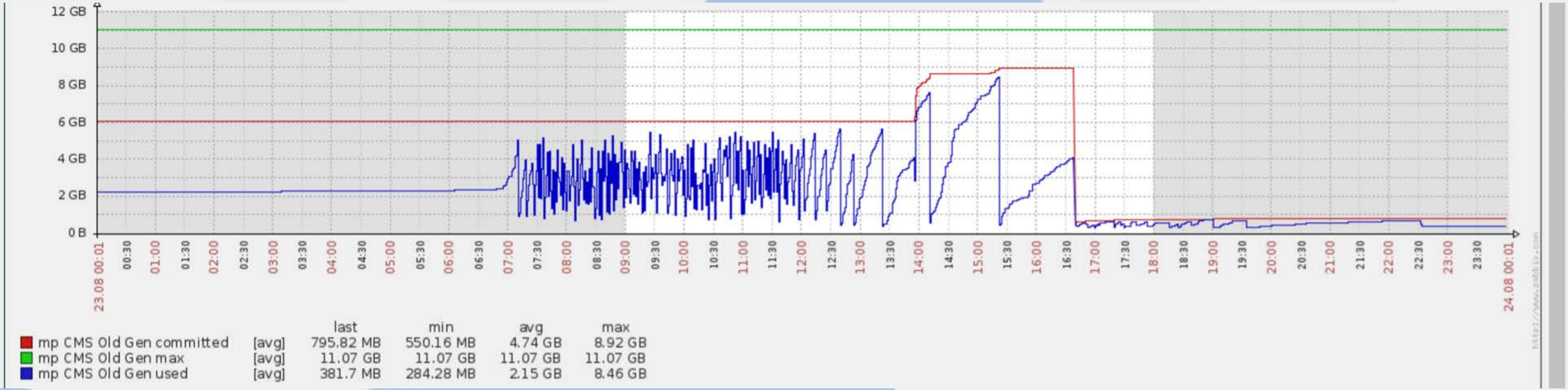
- Zabbix templates generalize monitoring items per server type
  - JMX Generic template
    - Standard Java VM Mbeans
      - Memory – all parts
      - Jvm version
      - Threads
      - Uptime
      - File descriptors
      - Garbage collector
      - Classloader
  - JMX Tomcat template
    - Sessions
    - Connector
      - Threads
      - Network



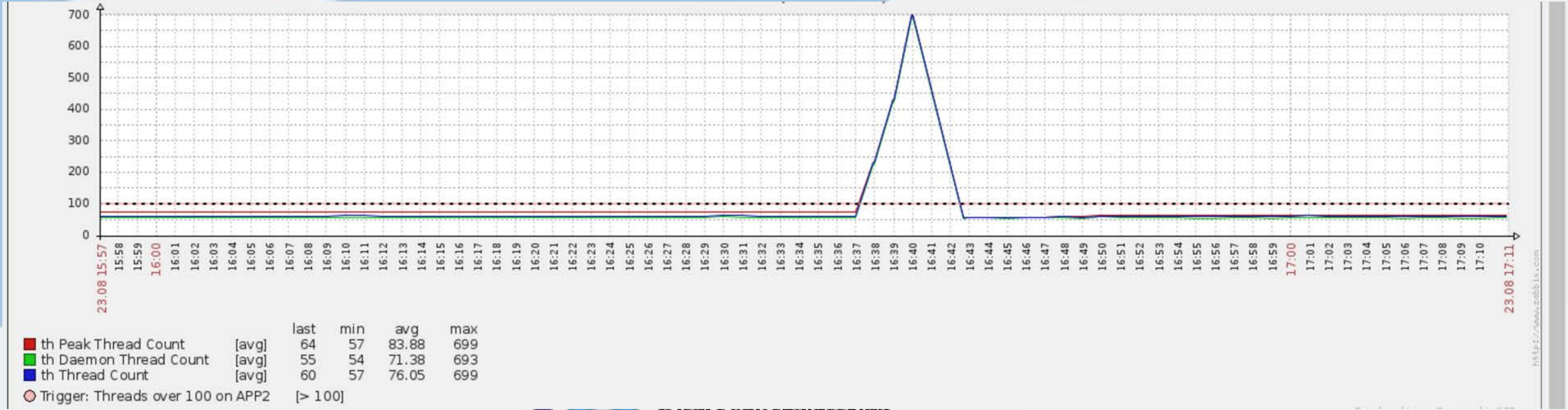
		last	min	avg	max
🟥 mp CMS Old Gen committed	[avg]	1.34 GB	1.34 GB	1.34 GB	1.34 GB
🟩 mp CMS Old Gen max	[avg]	1.34 GB	1.34 GB	1.34 GB	1.34 GB
🟦 mp CMS Old Gen used	[avg]	419.59 MB	126.4 MB	758.49 MB	1.28 GB



		last	min	avg	max
🟥 gc PS Scavenge number of collections per second	[no data]				
🟩 gc ConcurrentMarkSweep number of collections per second	[avg]	0	0	0.0008	0.0338
🟦 gc Copy number of collections per second	[no data]				
🟨 gc MarkSweepCompact number of collections per second	[no data]				
🟦 gs PS MarkSweep number of collections per second	[no data]				
🟪 gc ParNew number of collections per second	[avg]	0.0168	0	0.0764	0.8973



		last	min	avg	max
🟥 mp CMS Old Gen committed	[avg]	795.82 MB	550.16 MB	4.74 GB	8.92 GB
🟩 mp CMS Old Gen max	[avg]	11.07 GB	11.07 GB	11.07 GB	11.07 GB
🟦 mp CMS Old Gen used	[avg]	381.7 MB	284.28 MB	2.15 GB	8.46 GB



		last	min	avg	max
🟥 th Peak Thread Count	[avg]	64	57	83.88	699
🟩 th Daemon Thread Count	[avg]	55	54	71.38	693
🟦 th Thread Count	[avg]	60	57	76.05	699
○ Trigger: Threads over 100 on APP2	[> 100]				

SCIENCE AND ENGINEERING

Conclusion

- JMX presents a powerful Java monitoring and management interface
  - Can be used for instrumentation in run-time verificiation/monitoring
- Information provided for both custom objects as well as Java VM
- Enables better understanding of Java VM
- Critical for performance/uptime of Java Application servers
- Can be used for scale up/down on Cloud instances