Neil Masson – IBM Java Service Technical Lead

25th September 2012

Practical Performance
Understanding the Performance of Your Application

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Goals of the Talk

- So...
  - You have a performance problem..
  - You are not sure what the application is doing under the covers…

- What next ?

- After this talk you will:
  - Understand when and why to use performance tools
  - Have a toolkit of performance tools and techniques
  - Get to know your Java application better

Agenda

- Performance – why should you care?
- Approaches to performance
- Layers of the application
- Identifying bottlenecks

Approaches to performance

- Outside in approach
  - Start from where performance can be measured
  - Work along the activity path
  - Ideal for identified performance problems

- Layered approach
  - Analyze and eliminate layers of the application
  - Simplify the problem as you go
  - Ideal for application health check

- A hybrid of both approaches can often be useful

Performance baseline

- Important to have a repeatable performance test
- Measure baseline performance
  - Internal measurements affect the performance of what your measuring
  - External measurements have less impact on system performance

Layers of a Java application

- Three layers of a deployment:
  - Infrastructure: Hardware and Operating System
  - Java Runtime: Garbage Collection
  - Java Application: Java application code

- Each can suffer from resource constraints, typically:
  - Memory
  - CPU
  - Synchronization
  - I/O

Infrastructure

- Typical resource constraints:
  - Memory: insufficient physical memory results in paging/swapping
  - CPU: insufficient CPU time limits throughput of the application
  - I/O: insufficient I/O limits throughput of the application
  - Synchronization driven by Java runtime/Java application

- Easy to diagnose

- Easy to resolve (relatively)

- Note that each can also be caused by deficiencies higher up the stack!

Infrastructure memory usage

- Infrastructure uses memory for:
  – Backing the process data: OS runtime, Java runtime, Java application
  – Caching of IO: filesystem and network buffers

- Lack of physical memory causes:
  – Reduction and removal of IO caching
  – Paging/swapping of process memory to disk

- Paging/swapping is costly for a Java process
  – Particularly affects Garbage Collection performance
    - Paging usually occurs on Least Recently Used basis
    - All of Java heap is traversed during mark and sweep phases
    - Least Recently Used does not work well for the Java heap

Infrastructure CPU usage

- Insufficient CPU time availability will reduce performance

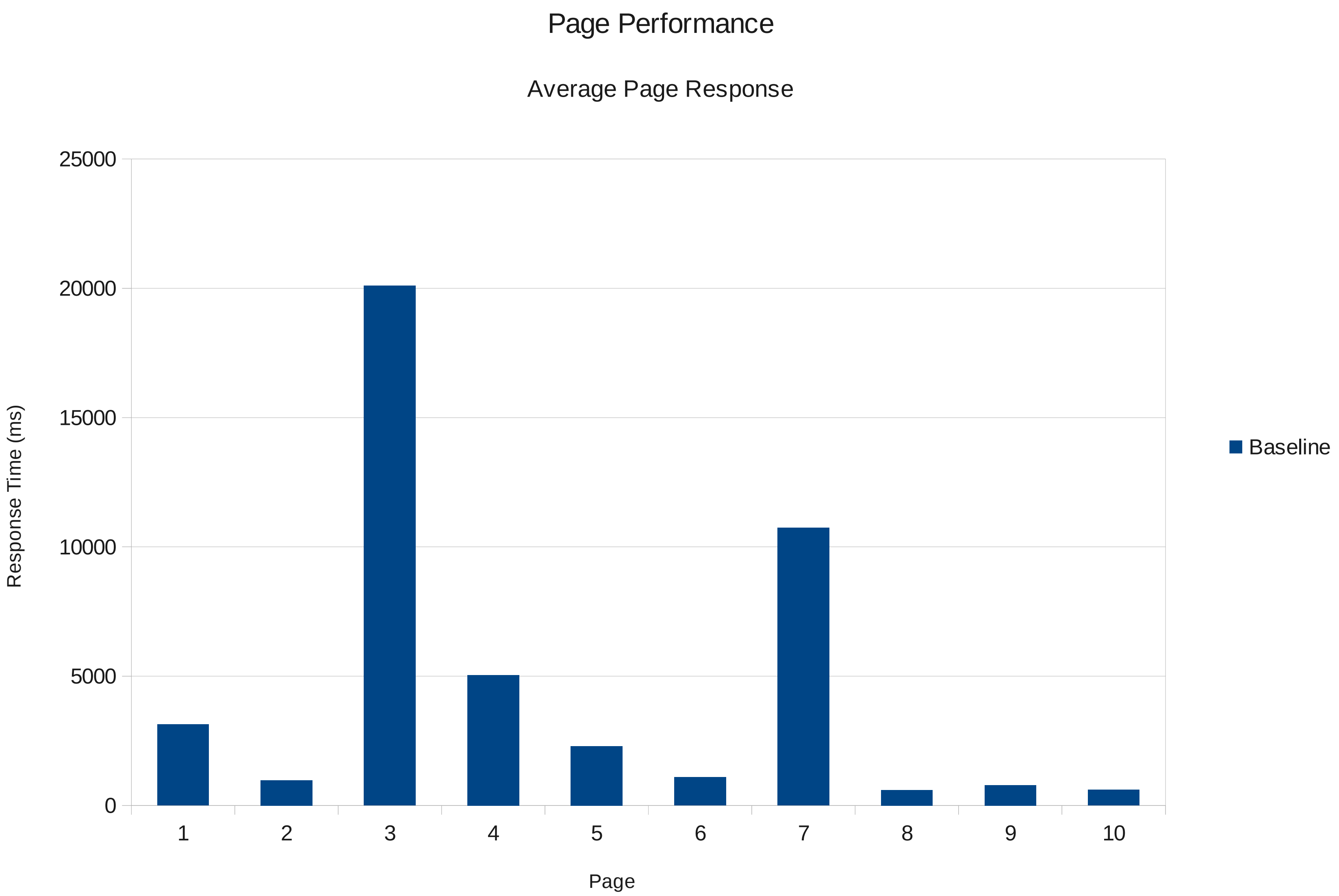
- Can occur periodically:
  – Cron Jobs running batch applications
  – Database backups

- Or during periods of high load:
  – System becomes CPU bound, limiting performance

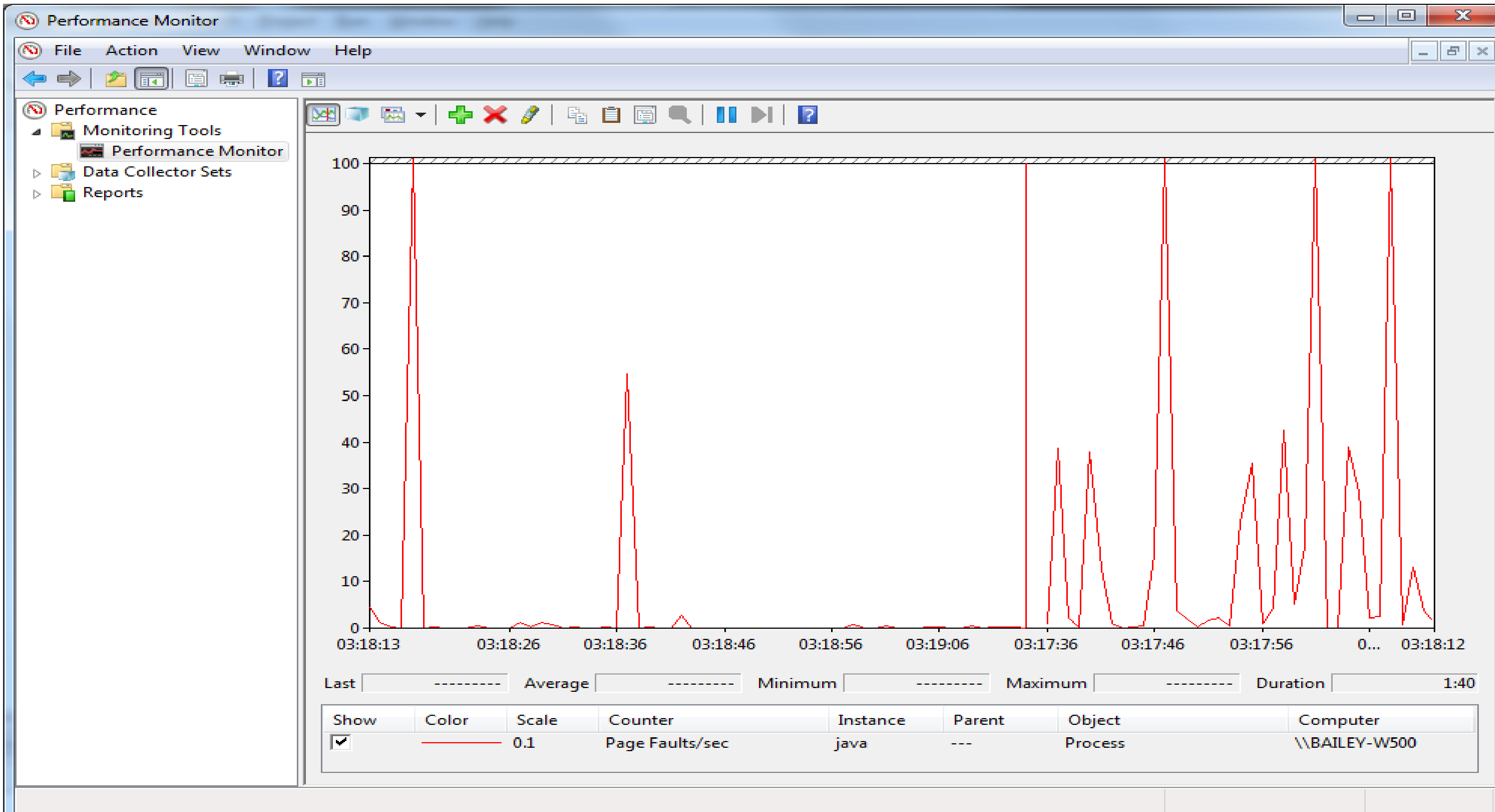
Detecting infrastructure issues

- Detect using Operating System level tools
- Memory on Windows:
  - Paging: using "perfmon" with "Process" counter for "Page Faults/sec"
  - File Cache: using "perfmon" with "Memory" counter for "System Cache Resident Bytes"
- CPU on Windows:
  - Per process: using "perfmon" with "Process" counter for "% Processor Time"
  - Per machine: using "perfmon" with "Processor" counter for "% Processor Time"
- IO on Windows:
  - Network: using "perfmon" with "Network Interface" counter for "Output Queue Length"
  - Disk: using "perform" with "Physical Disk" counter for "Current Disk Queue Length"

Page response performance benchmark: baseline



Paging in perfmon

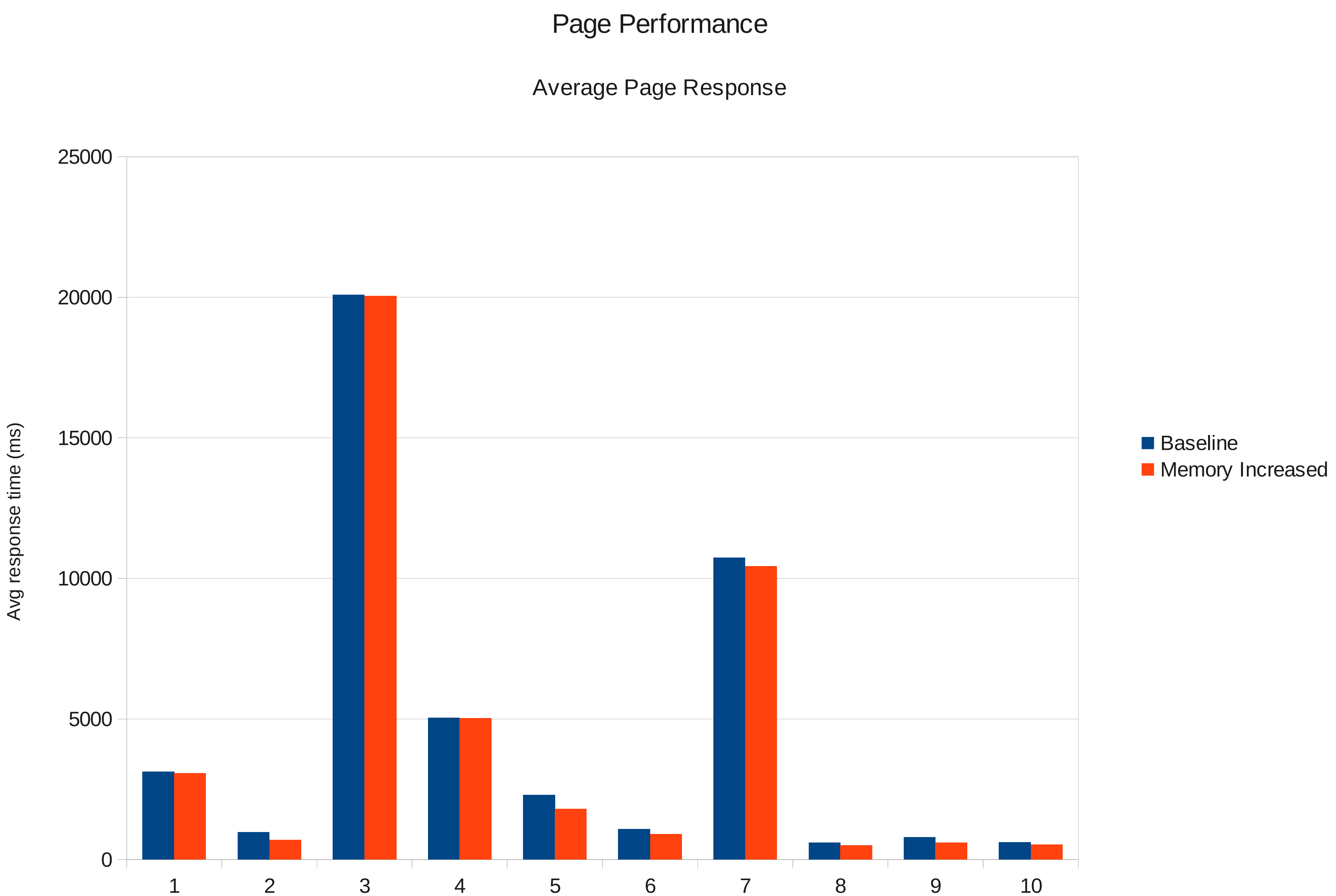


Resolving infrastructure issues

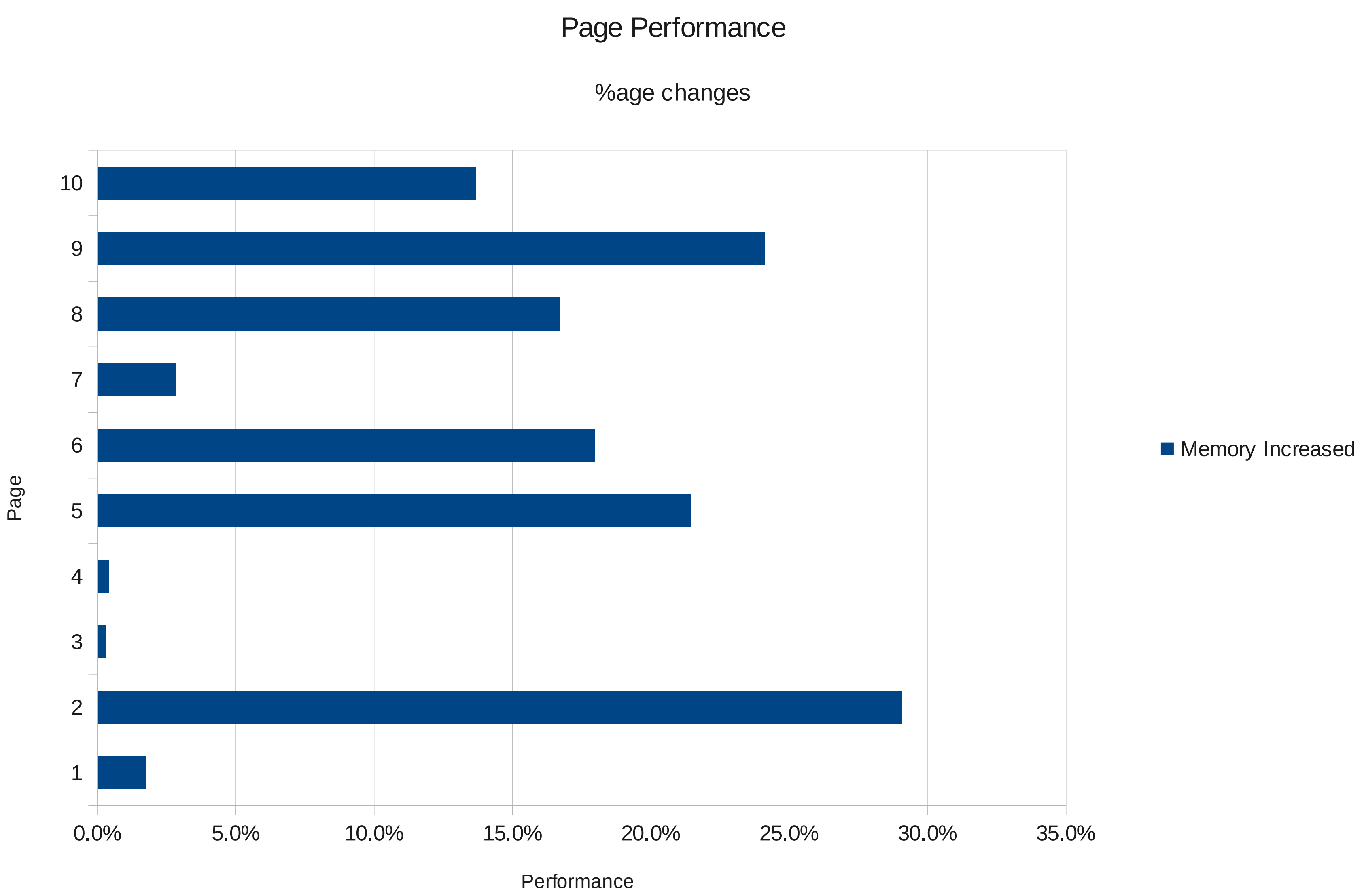
- Add more physical resources to the process
  – Assign more to the: Machine, Guest OS, LPAR, Zone, etc

- Reduce the physical resource requirements
  – Reduce the application footprint
  – Reduce the application CPU usage
  – Reduce the IO

Page response performance benchmark: memory increased



Page response performance benchmark: memory increased



Java runtime

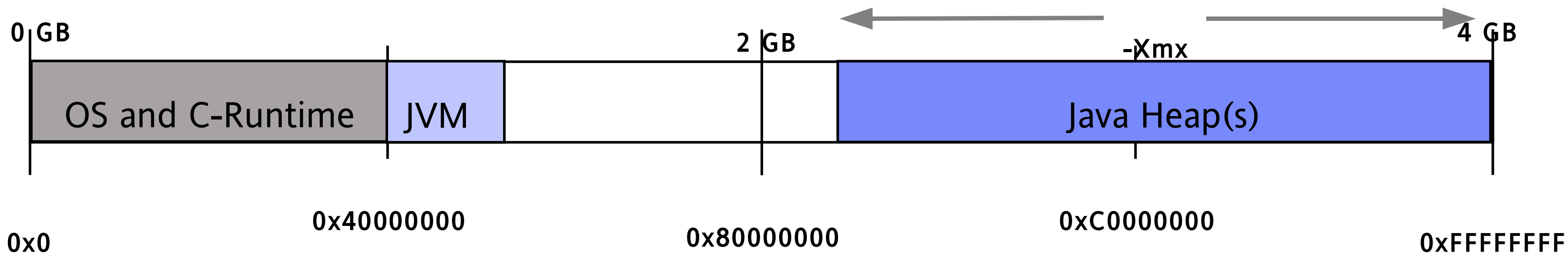
- Typical resource constraints:
  - Memory: insufficient Java heap results in OutOfMemory or high GC overhead
  - CPU Garbage Collection overhead, or driven by Java application
  - Synchronization driven by Java application
  - IO driven by Java application

- Easy to diagnose

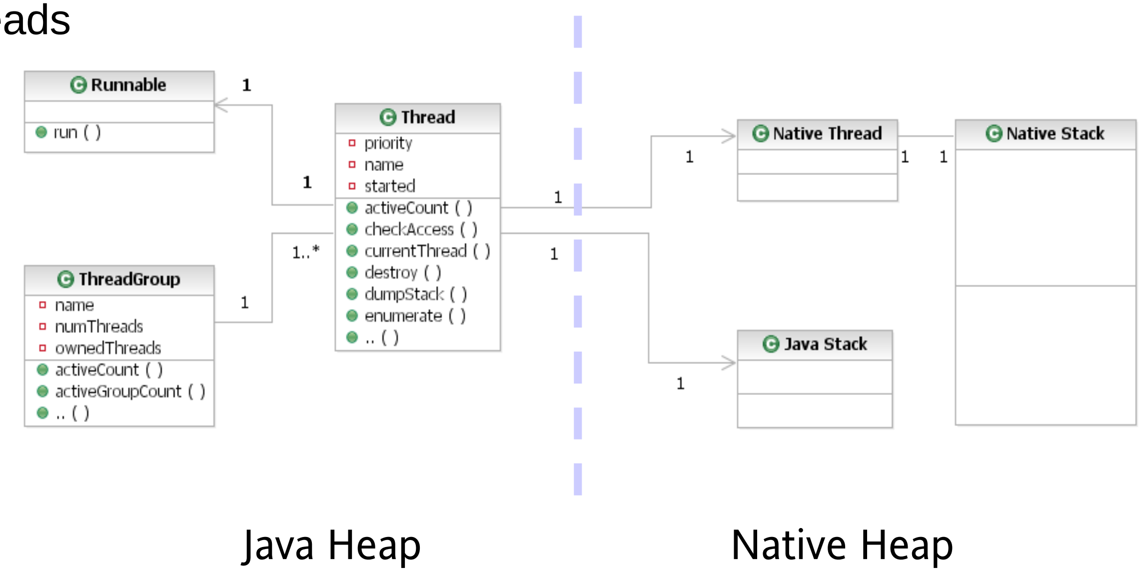
- Easy to resolve (relatively)

Java runtime memory

- Java runtime uses memory for:
  - Java Heap(s), Java Virtual Machine (JVM), “Native” heap, OS and C-language runtime



- Java heap(s) are managed using Garbage Collection
- Other memory usage can be indirectly driven by application usage and garbage collection
  - eg. Java Threads



Java runtime problems

- Insufficient Java heap memory leads to:
  – OutOfMemoryError due to Java heap exhaustion
  – Garbage Collection running excessively, increasing CPU and affecting performance

- Insufficient non-Java (“native”) heap leads to:
  – OutOfMemoryError due to process address space exhaustion
  – Driver for Java heap garbage collection (DirectByteBuffer cleaners)

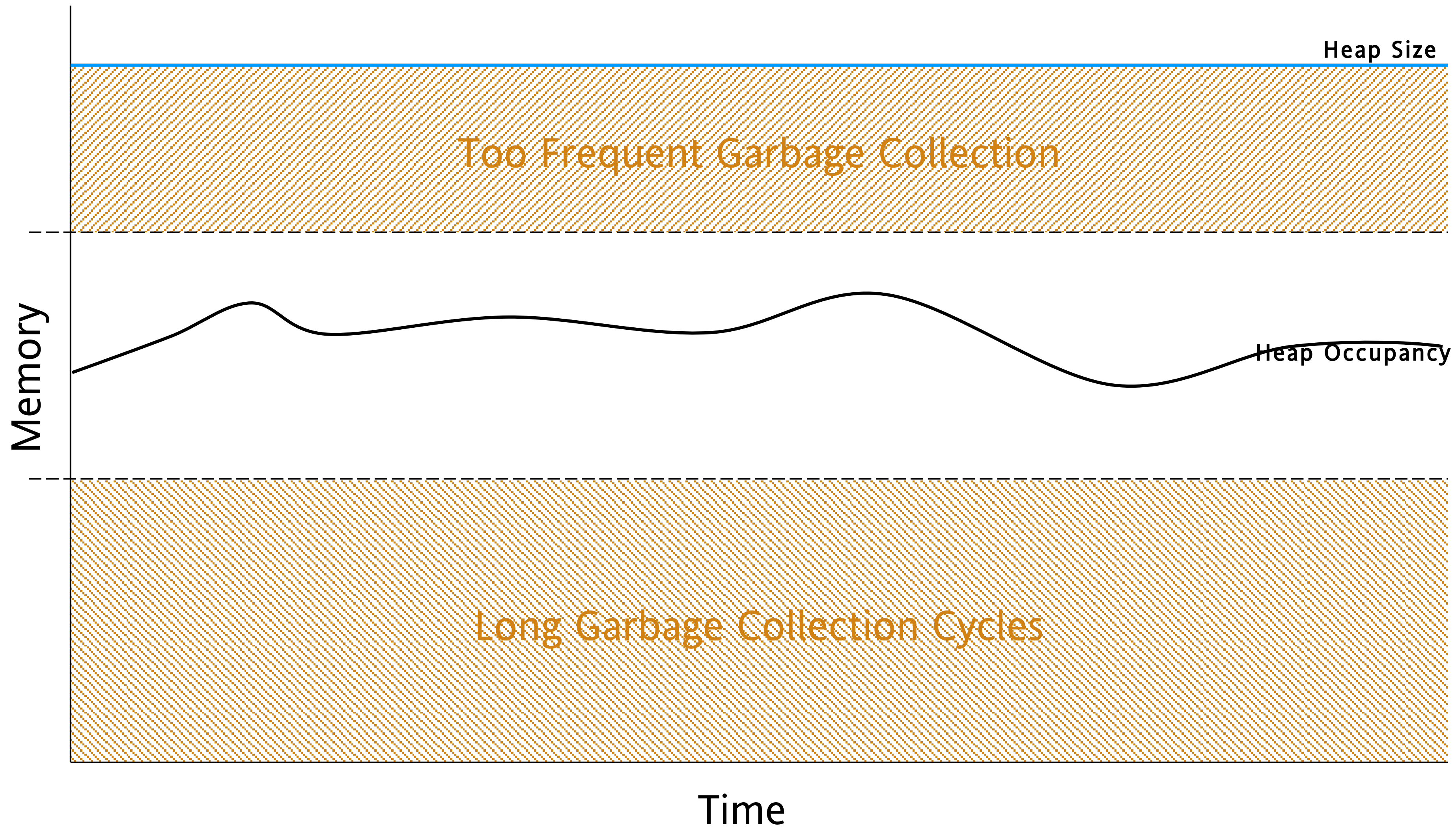
Detecting Java runtime problems

- Log and trace analysis:
  – "Native" heap: OS level logs (ps, svmon, perfmon)
  – Java heap: verbose:gc output

  – Post processed using:
  IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer (GCMV)

- Live monitoring:
  – "Native" heap IBM Monitoring and Diagnostic Tools for Java - Health Center
  – Java heap: IBM Monitoring and Diagnostic Tools for Java - Health Center
  Visual VM, Mission Control

Garbage collection performance

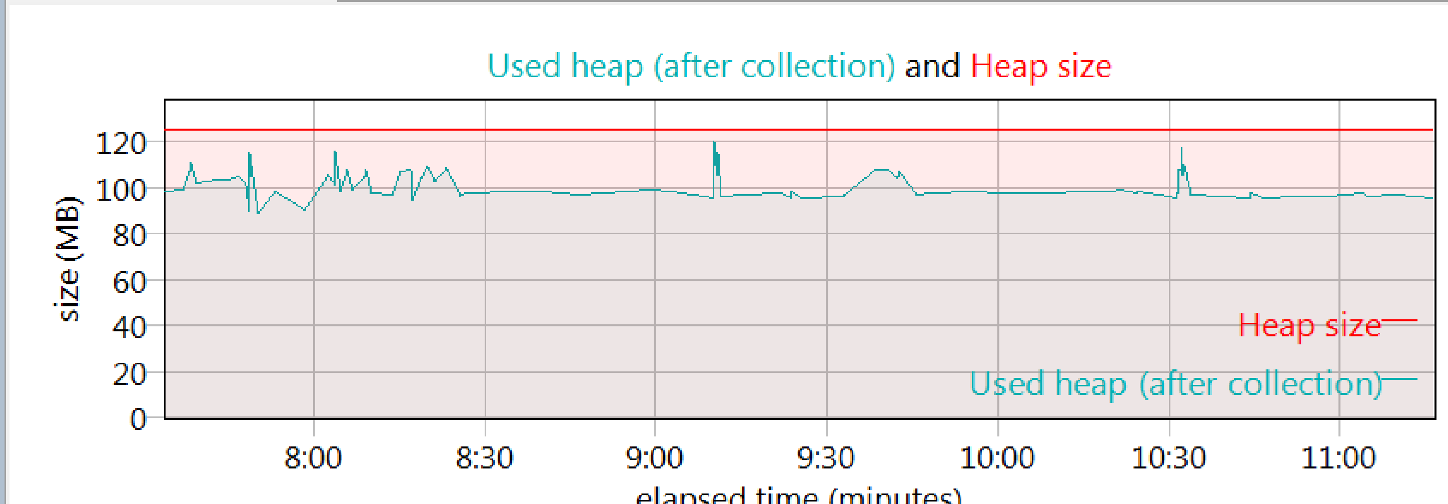


Too Frequent Garbage Collection

Status

- Classes
- Environment
- Garbage Collection
- I/O
- Locking
- Native Memory
- Profiling

Used heap (after collection)



Analysis and Recommendations

The mean occupancy is 81%. This is high, so you may improve application performance by increasing your heap size.

The application seems to be using some quite large objects. The largest request which triggered an allocation failure was for 10000 KB.

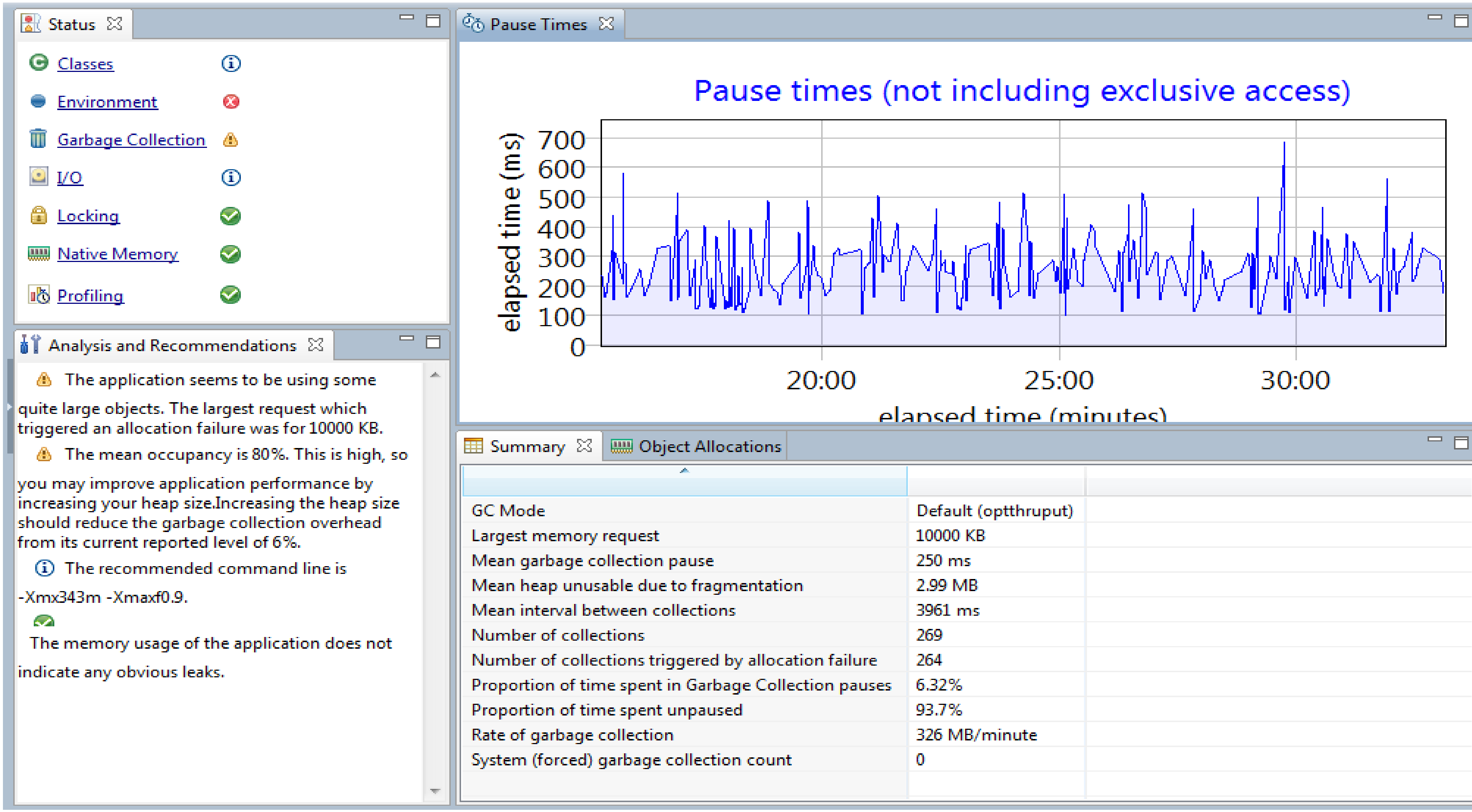
The recommended command line is -Xmx336m.

The memory usage of the application does not indicate any obvious leaks.

Summary | **Object Allocations**

GC Mode	Default (optthruput)
Largest memory request	10000 KB
Mean garbage collection pause	239 ms
Mean heap unusable due to fragmentation	1.6 MB
Mean interval between collections	2924 ms
Number of collections	76
Number of collections triggered by allocation failure	75
Proportion of time spent in Garbage Collection pauses	8.16%
Proportion of time spent unpaused	91.8%
Rate of garbage collection	437 MB/minute
System (forced) garbage collection count	0

Garbage Collection Pause Times



Resolving Java runtime problems

- Add more resources to the Java runtime
  - Java heap: Increase Java heap size
  - Native heap: Move to 64bit or reduce Java heap size

- Reduce the memory requirements
  - Reduce the Java application footprint

Increased Java heap size

Status

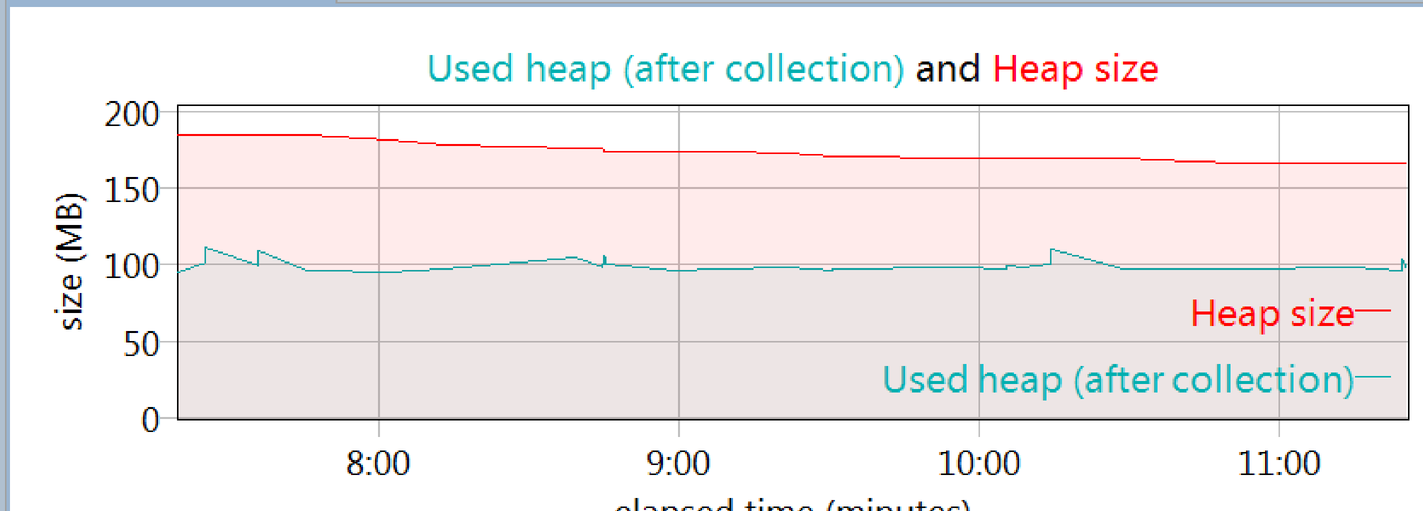
- Classes
- Environment
- Garbage Collection
- I/O
- Locking
- Native Memory
- Profiling

Analysis and Recommendations

The application seems to be using some quite large objects. The largest request which triggered an allocation failure was for 10000 KB.

The mean occupancy is 57%. This is close to optimal, so you do not need to tune your heap size.

Used heap (after collection)



Summary | Object Allocations

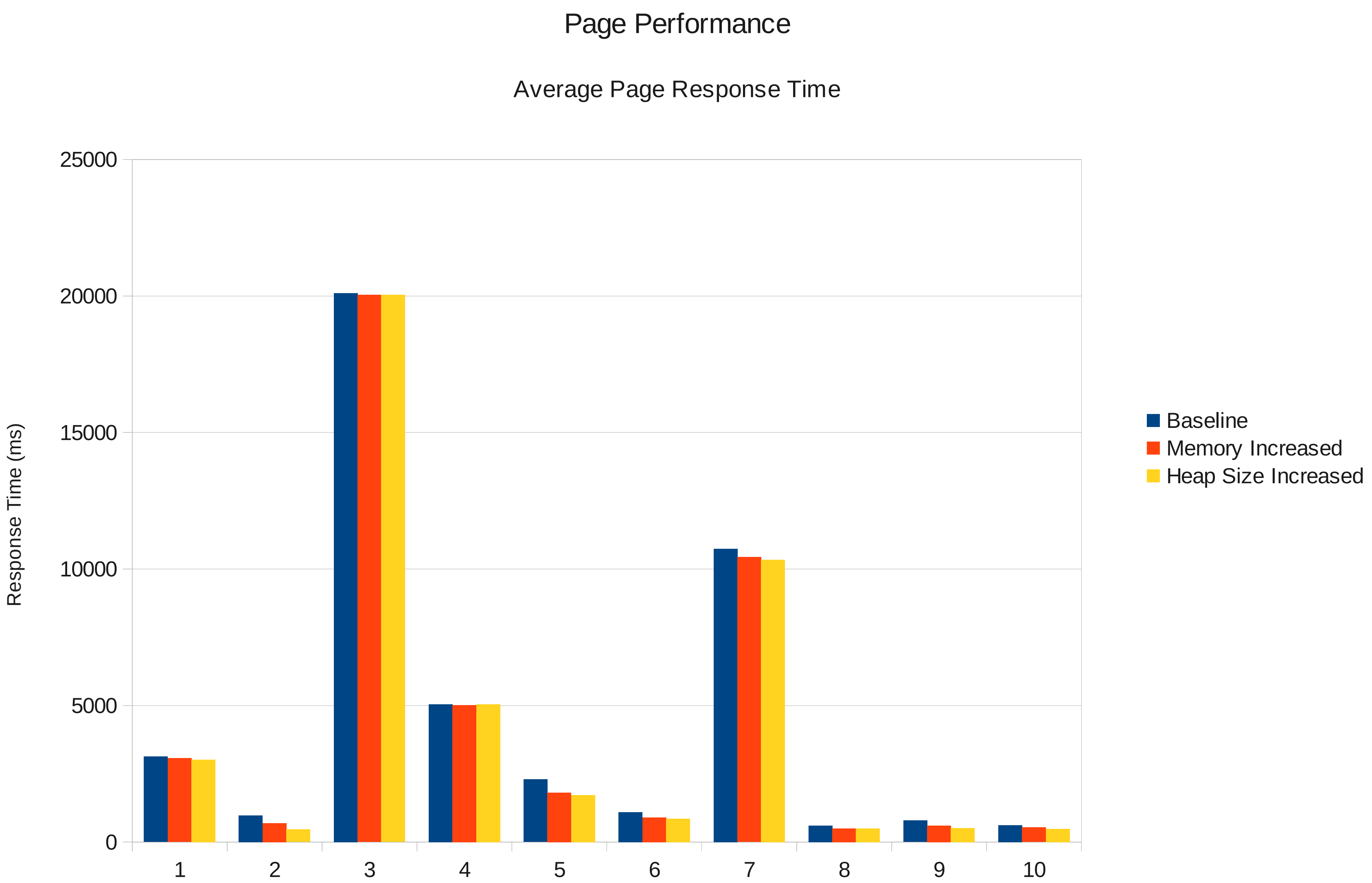
GC Mode	Default (optthruput)
Largest memory request	10000 KB
Mean garbage collection pause	326 ms
Mean heap unusable due to fragmentation	11.2 MB
Mean interval between collections	9824 ms
Number of collections	25
Number of collections triggered by allocation failure	25
Proportion of time spent in Garbage Collection pauses	3.32%
Proportion of time spent unpaused	96.7%
Rate of garbage collection	379 MB/minute
System (forced) garbage collection count	0

Effect on Garbage Collection Pause Times

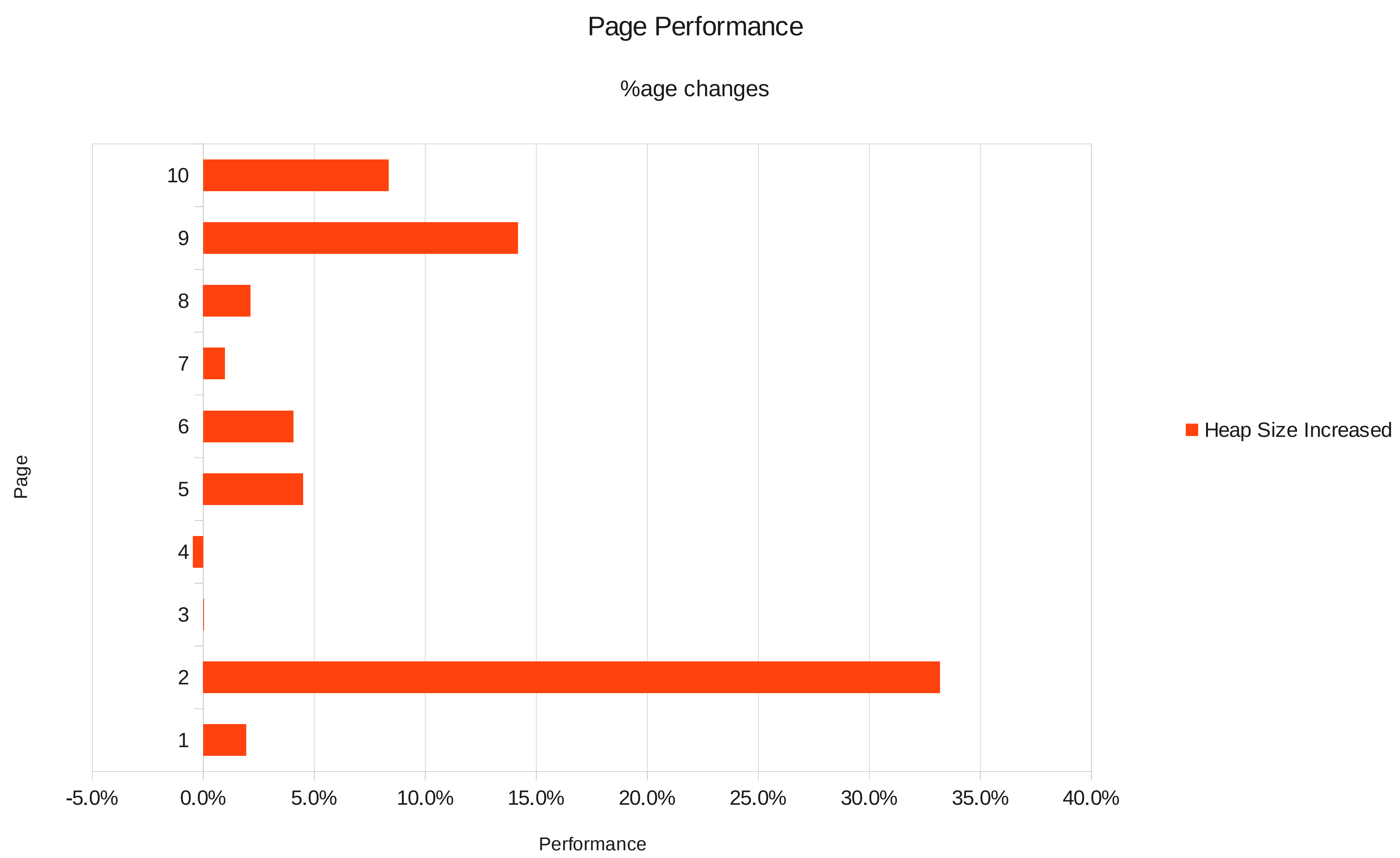
- Reduction in:
  – Time spent in GC 59%

- However this is only 4.84% of total time

Page response performance benchmark: Java Heap Size Increased



Page response performance benchmark: Java Heap Size increased



Java application

- Typical resource constraints:
  - Memory: insufficient caching affects application throughput and responsiveness
  - CPU: insufficient threading causes limits on scalability
  - Synchronsation: synchronized resources limits scalability and throughput of the application
  - I/O: blocking on I/O limits throughput and responsiveness

- Hard to diagnose

- Can be expensive (or impossible!) to resolve

Java application CPU usage

- High CPU usage by Java methods highlight areas of potential optimization
  - Code is being invoked more than it needs to be
    - Easily done with event driven models
  - An algorithm is not the most efficient
    - Easily done if performance is not the focus at development time

- Fixing CPU bound applications requires knowledge of what code is being run
  - Identify methods which are suitable for optimisation
    - Optimising methods which the application doesn't spend time in is a waste of your time
  - Identify methods where more time is being spent that you expect
    - "Why is so much of time being spent in this trivial method?"

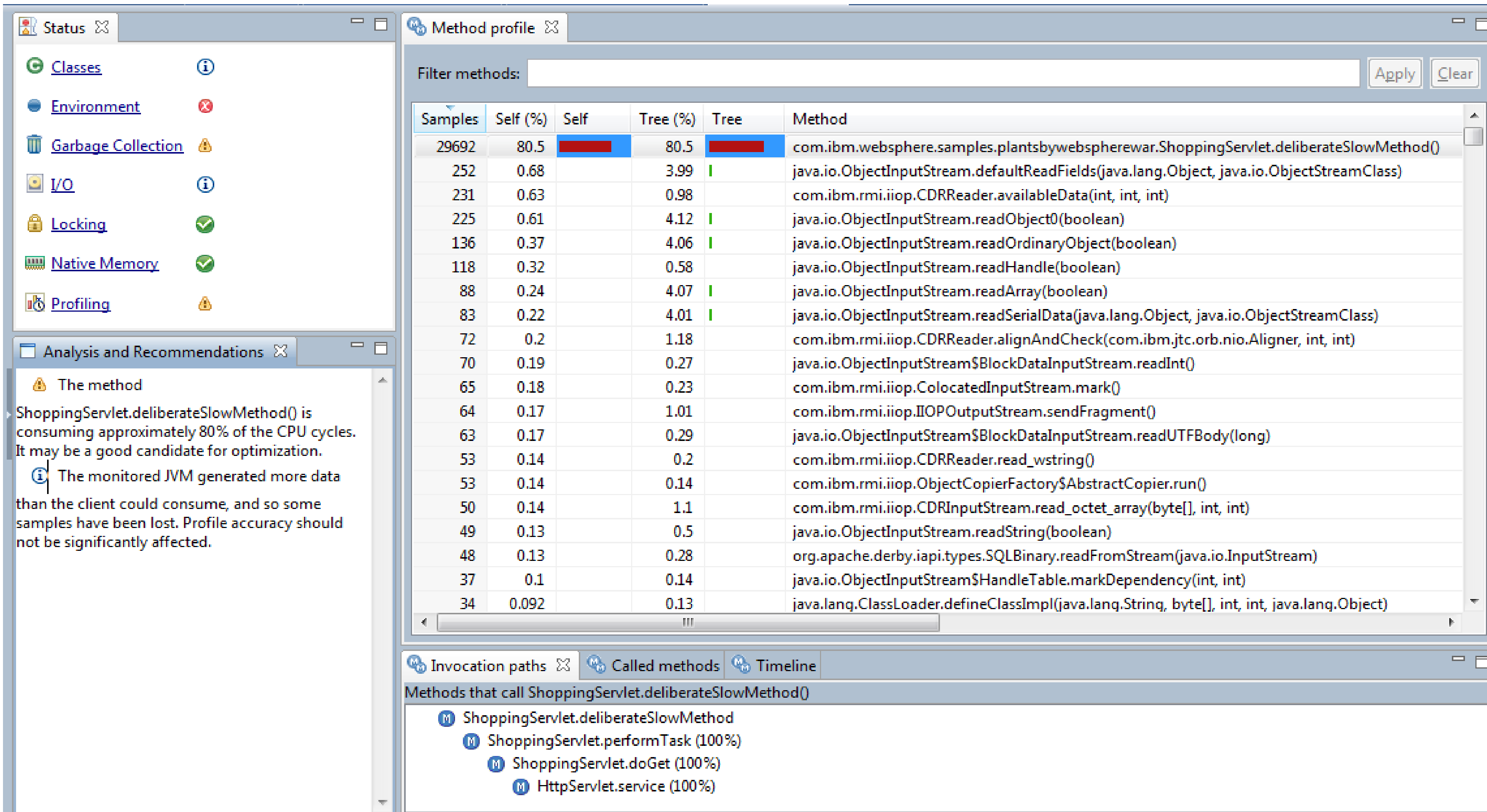
Java application synchronization

- Throughput does not increase linearly with load

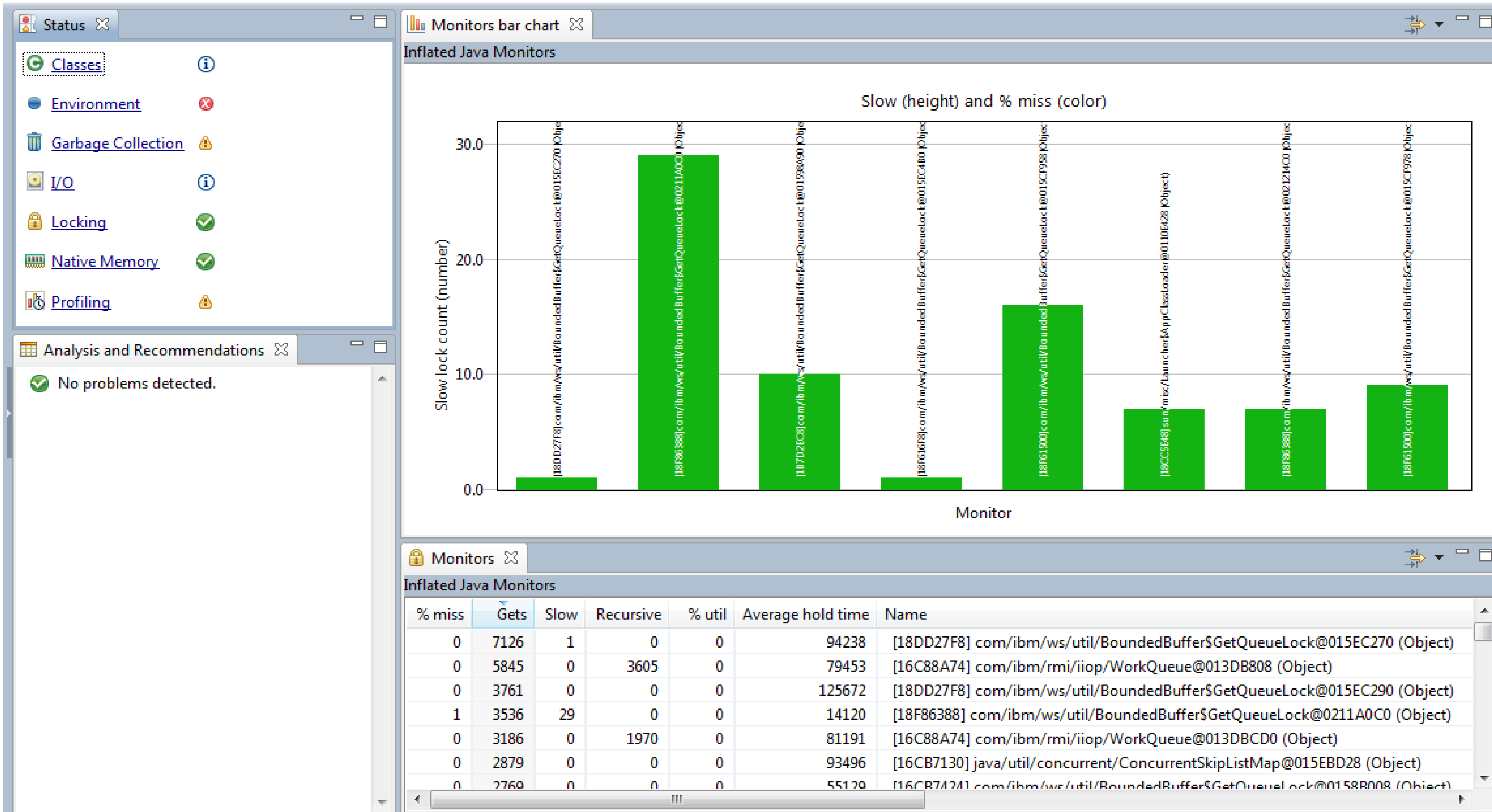
- At limit of throughput the CPU is still low
  – Inability to scale
  – Not all CPU can be utilized
  – Limit on throughput and responsiveness

- Bottleneck where threads need to synchronize with each other for application correctness
  – Caused by large numbers of threads requiring synchronized resource at the same time
  – Caused by long hold time by thread that owns resource
  – Or a mixture of both

Health Center: application method CPU usage



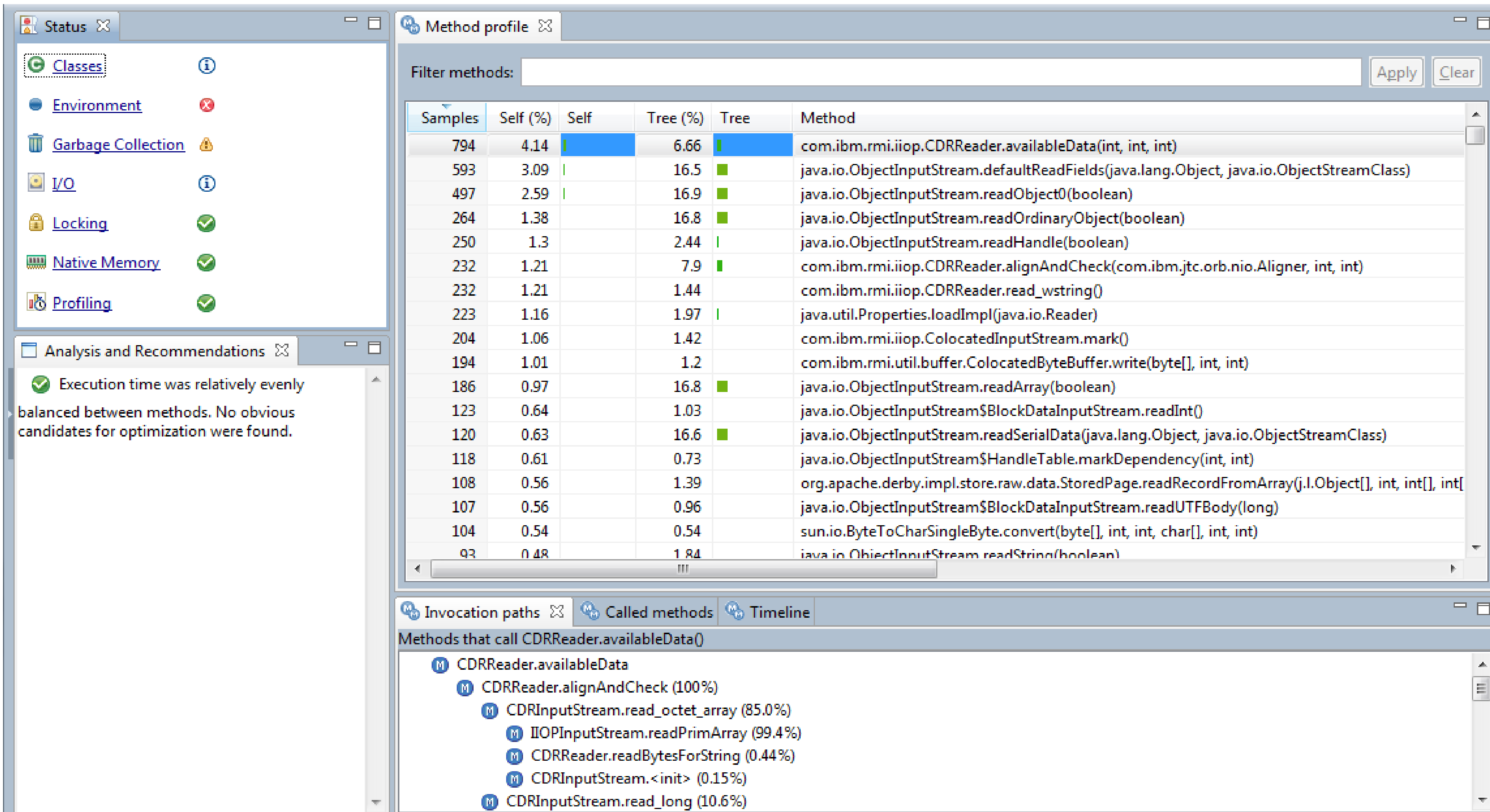
Health Center: application synchronization



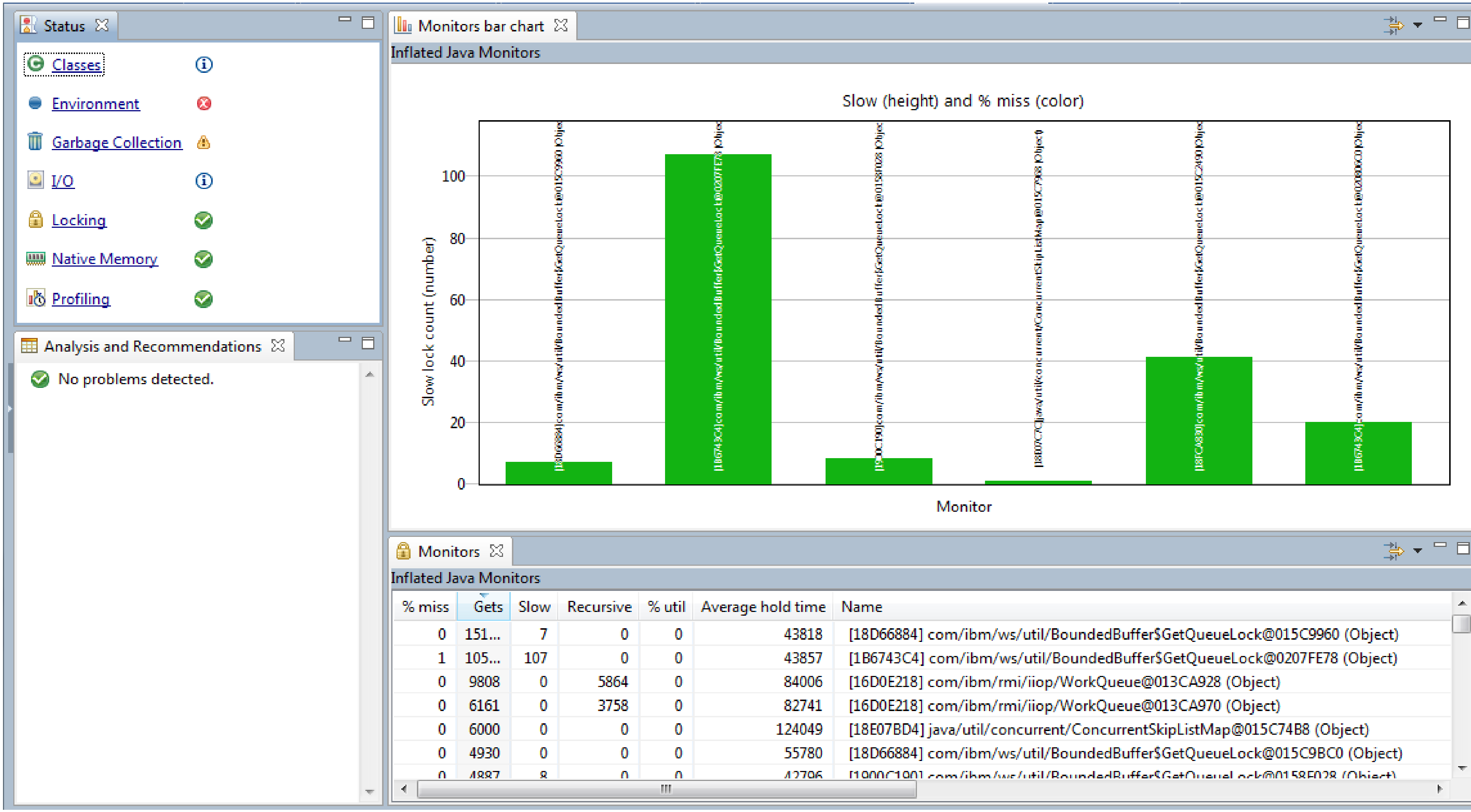
ShoppingServlet.deliberateSlowMethod()

```java
private void deliberateSlowMethod() {

    // ------------------------------------------------------------------
    // User clicked on the Tulips, let's tip toe through a
    // slow method
    // ------------------------------------------------------------------

    System.out.println("==> STARTING SLOW METHOD");

    long timestamp = System.currentTimeMillis();
    long target = timestamp + SLOWTIME;

    System.out.println("timestamp="+timestamp);
    System.out.println("resume at="+target);
    while(timestamp < target) {

        timestamp = System.currentTimeMillis();

    }

    System.out.println("==> ENDING SLOW METHOD");
}
```

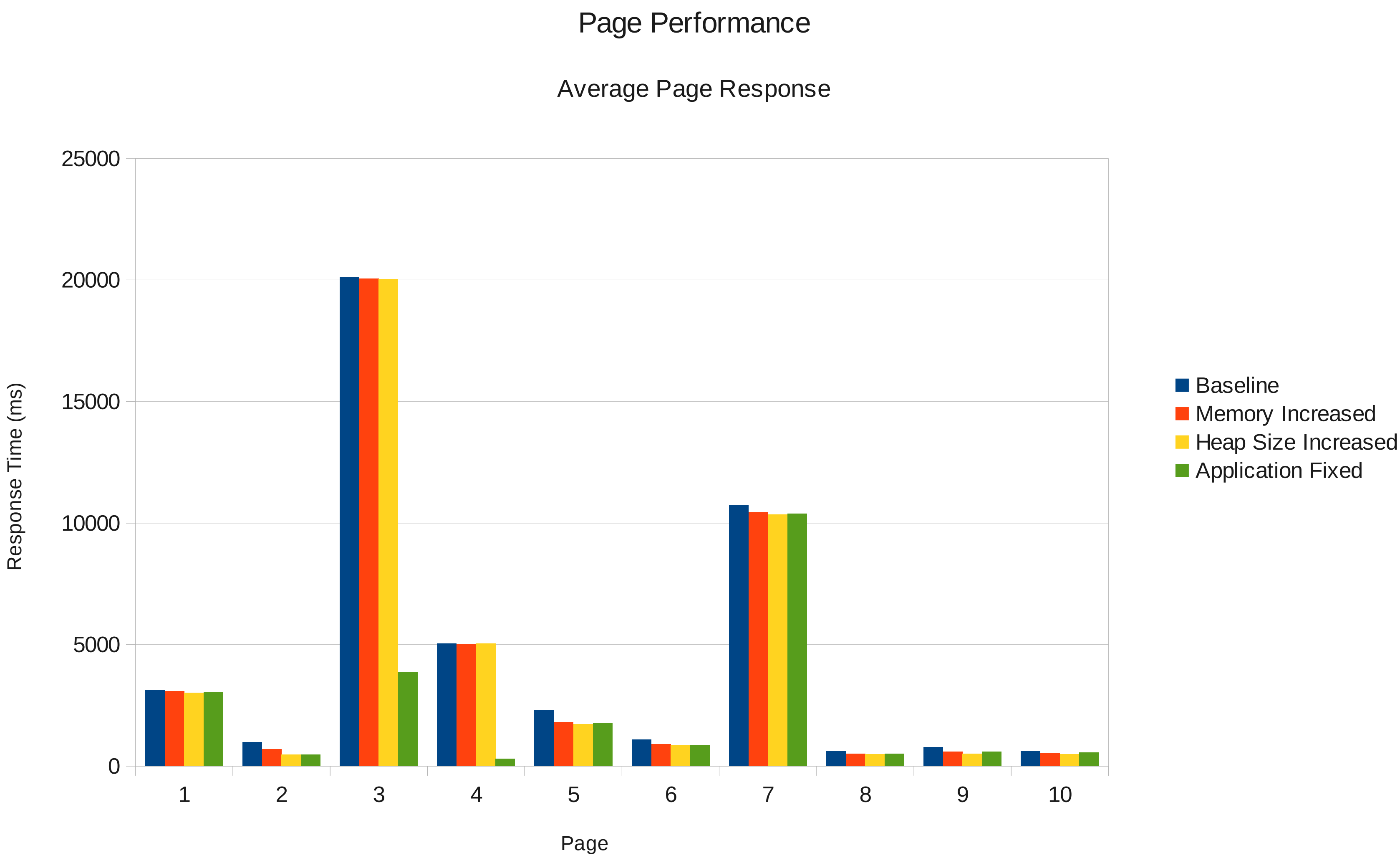
Health Center: application method CPU usage



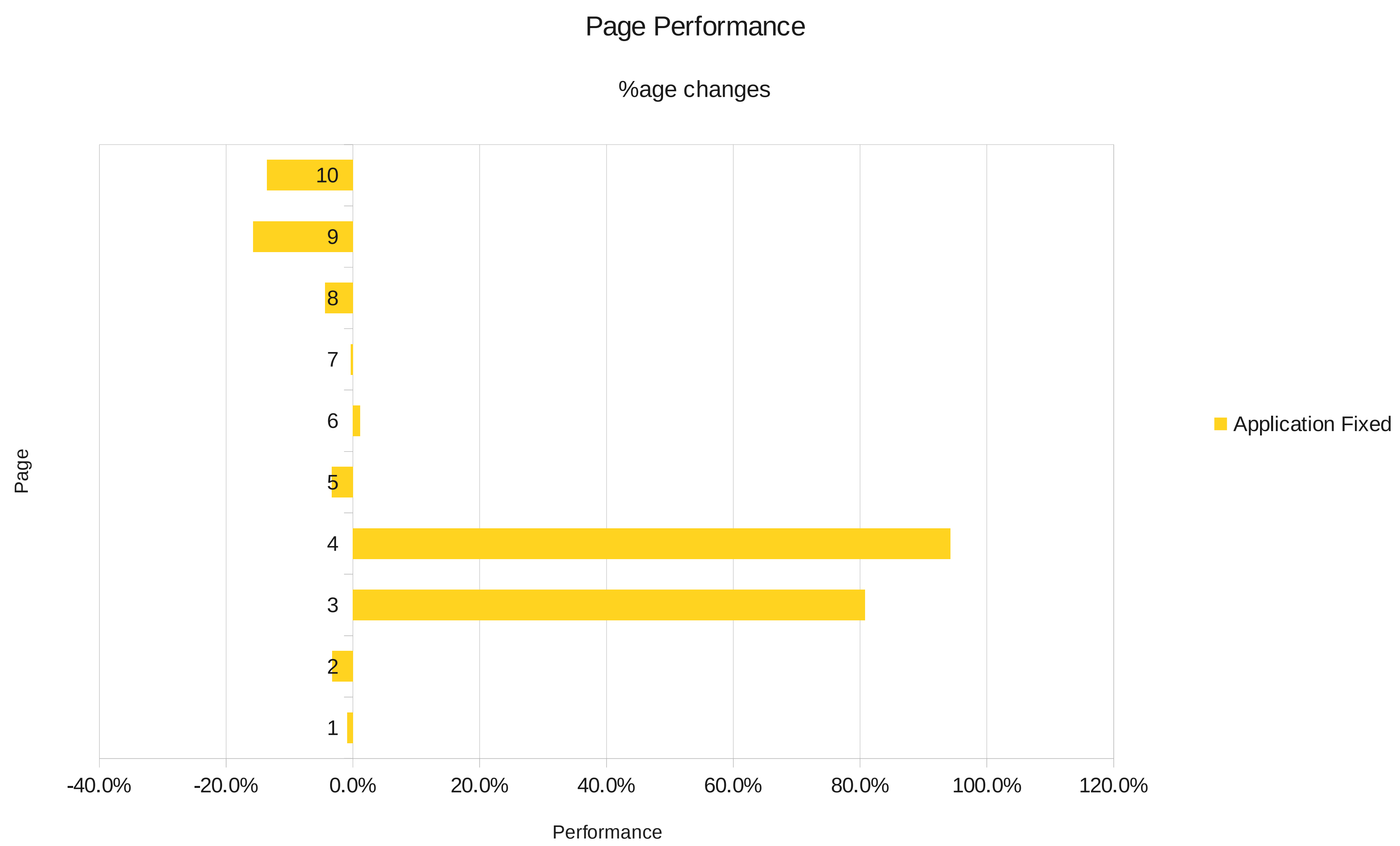
Health Center: application synchronization



Page response performance benchmark: deliberateSlowMethod() changed



Page response performance benchmark: deliberateSlowMethod() changed



Java application memory usage

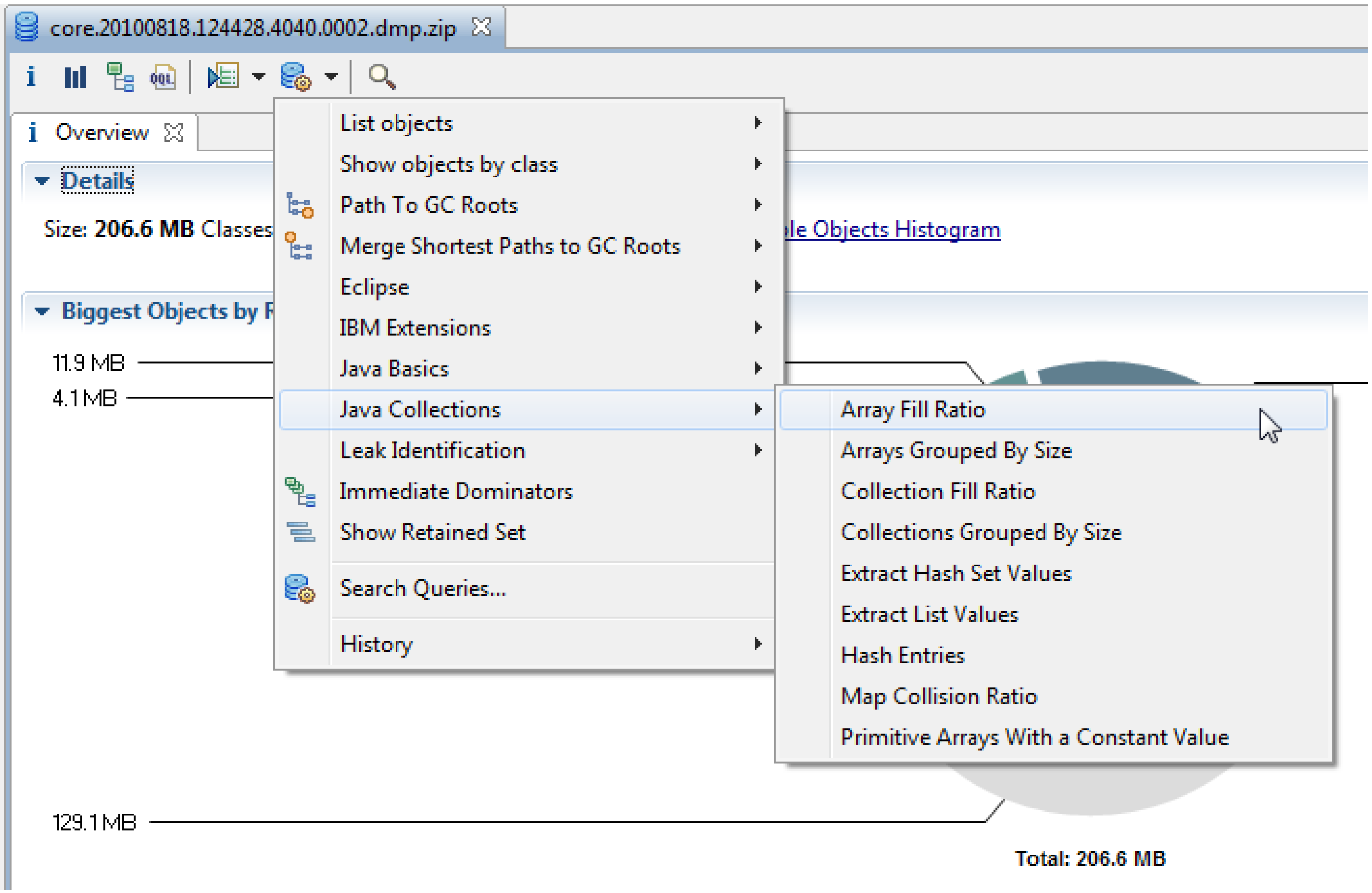
- Used for “in-flight” work
  - eg. Currently active transactions in a messaging system
- Used for caching data
  - Reduce volume of IO and improve responsiveness

Java application memory problems

- Memory Leaks
  - Unbounded growth of collections
  - OutOfMemoryErrors
- Memory Footprint
  - Incorrectly sized caches
  - Inefficient collection selection
  - Leads to lower performance
- Garbage generation
  - Creation/destruction of large amounts of data
  - Leads to lower performance

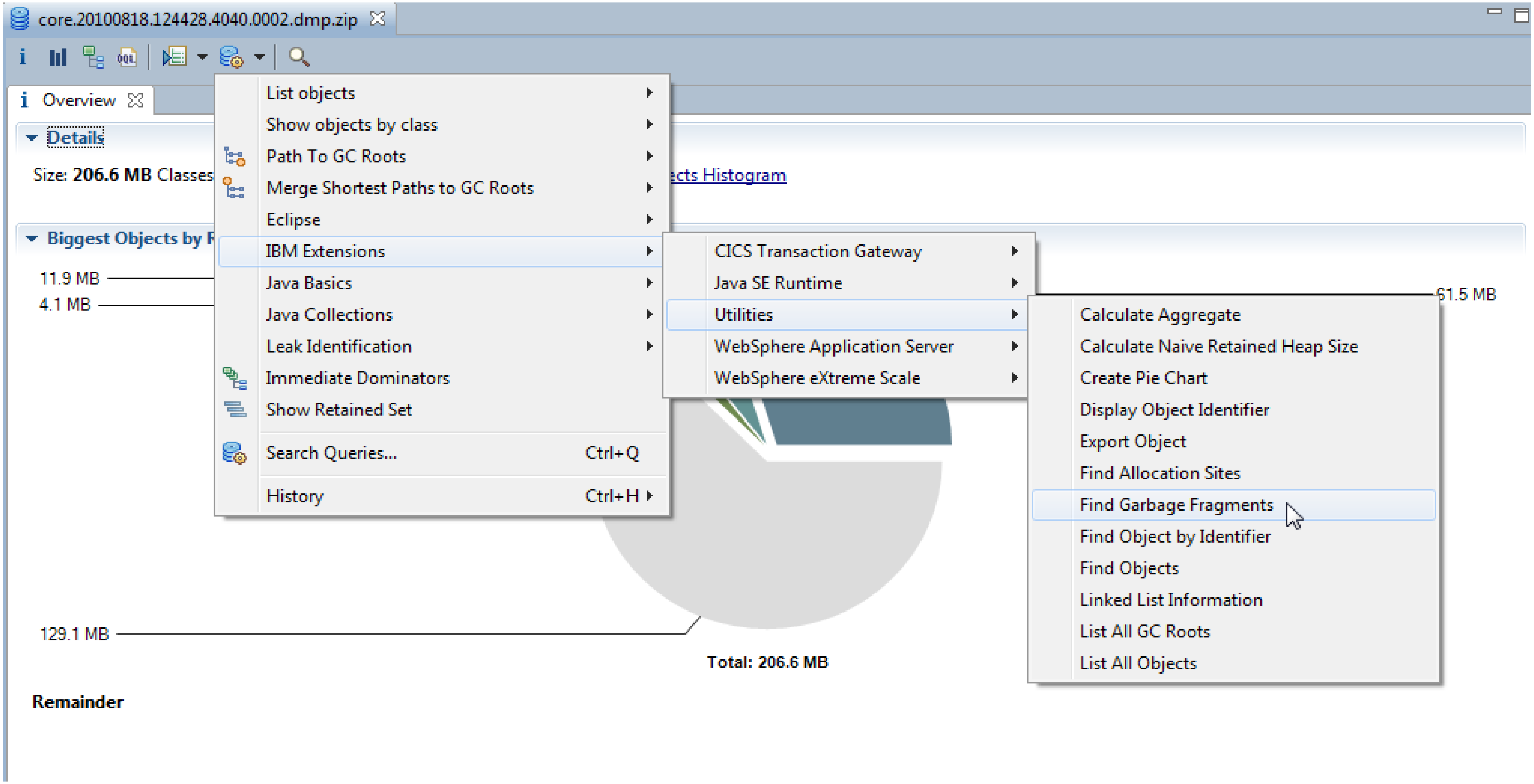
Analyzing your Collections

- Eclipse Memory Analyzer Tool (MAT) provides Collection analysis:



Analyzing your garbage

- Eclipse Memory Analyzer Tool (MAT) with the IBM Extensions for Memory Analyzer provides garbage analysis:



Memory Footprint Summary

- Collections exist in large numbers in many Java applications

- Example: IBM WebSphere Application Server running PlantsByWebSphere:
  - HashSet 1,551 instances
  - HashMap 12,151 instances
    10,600 instances (excluding HashSets)
  - LinkedList 1,148 instances
  - ArrayList 9,530 instances
    22,829 total collection instances

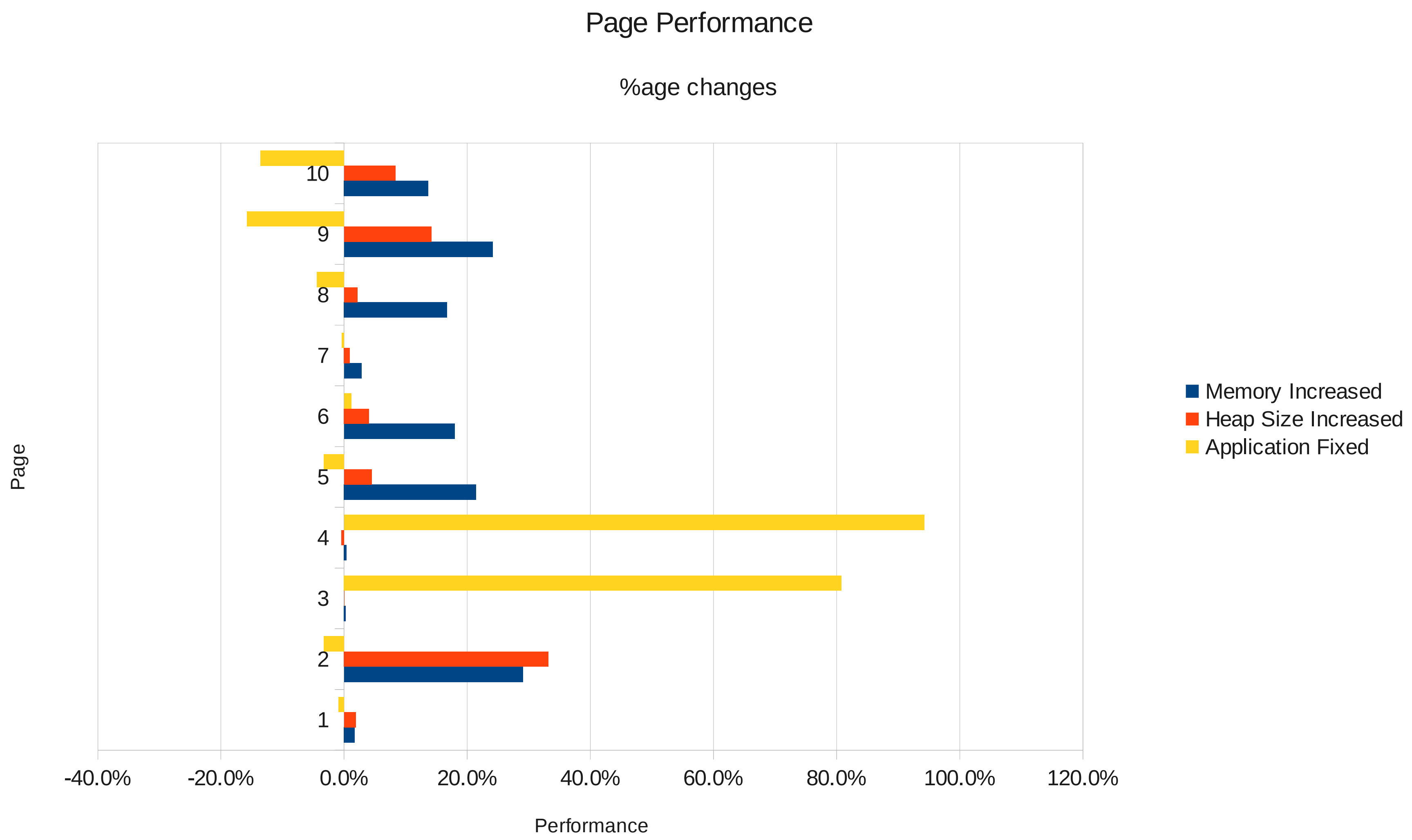
  - When running a 5 user test load, and using 206MB of Java heap

Summary

- Importance of:
  - Repeatable benchmark
  - Incremental measurements as changes are made

- Tools are available to help you see what's going on:
  - Garbage Collection and Memory Visualizer (all vendors)
  - HealthCenter (IBM only)
  - Other profilers (eg. YourKit) (all vendors)
  - Memory Analyzer (all vendors)

Page response performance benchmark: Summary of Changes



Summary

- Infrastructure resources affect performance
  – Paging and Garbage Collection much less than you might expect
  – However, beware of CPU “starvation” from other processes!

- Vast majority of performance gains are in the application!

References

- **Get Products and Technologies:**
  - IBM Monitoring and Diagnostic Tools for Java:
    - https://www.ibm.com/developerworks/java/jdk/tools/

- **Learn:**
  - Health Center InfoCenter:
    - http://publib.boulder.ibm.com/infocenter/hctool/v1r0/index.jsp

- **Discuss:**
  - IBM on Troubleshooting Java Applications Blog:
    - https://www.ibm.com/developerworks/mydeveloperworks/blogs/troubleshootingjava/
  - Health Center Forum:
    - http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1461
  - IBM Java Runtimes and SDKs Forum:
    - http://www.ibm.com/developerworks/forums/forum.jspa?forumID=367&start=0

Copyright and Trademarks