Java Performance Tooling

Dr. Holly Cummins
Java Performance Tooling Lead
IBM

Kirk Pepperdine
Consultant
Kodewerk Ltd.
Dr. Holly Cummins

- Software engineer at IBM’s Java Technology Center
- Author of GC and Memory Visualizer (formerly EVTK)
- Technical lead for IBM Monitoring and Diagnostic Tools for Java - Health Center
- Speaks frequently on Java performance and garbage collection
Kirk Pepperdine

- Independent consultant
- performance tuning service
- benchmarking
- performance tuning course

- Nominated Java Champion
- Co-author www.javaperformancetuning.com
- Past editor of www.theserverside.com
- Speak frequently about tuning
- Other stuff.. blahh blahh blahh....
Disclaimer

Any performance tuning advice provided in this presentation.....

will be wrong!
Knowing what to measure and how to measure it makes a complex world much less so

Steven D. Levitt
Stephen J. Dubner
Authors of Freakonomics
Puzzle
Performance Tuning

- Understanding of Technology
- Methodology
- Tooling
Performance Tuning

Understanding of Technology

Methodology

Tooling
Performance Tuning Aspects

Understanding of Technology

Methodology

Tooling
Performance Tuning Aspects

Understanding of Technology

Tooling

Methodology
Poor Performance Defined

Activities that creates demands for system resources causes response times to exceed stated tolerances.

Scarcnon-sharable hardware software components.

Soft requirements

Dynamic driver

Load

User experience
Causes

- Insufficient capacity
  - hardware

- Inefficient use of capacity
  - architecture
  - algorithmic strength
  - unnecessary work

- Combinations of the above
Performance Tuning Defined

Activities that create demands for system resources causes response times to exceed stated tolerances.

Identify
Measure utilization

Benchmark
Goal

Monitor

14
A Complete System “The Box”

- Actors
- Application
- JVM
  (Operating System)
- Hardware
Dynamic Aspects

- **Actors**
  - end users, batch processes
  - external systems

- **Usage patterns**
  - use case + load + timing attributes
  - quantities and frequency
  - time of day, day of week, month etc...

- Creates pressure on pinch points
Static Aspects

- Components that don’t change
- Manage or provide resources
Application

- Static layer that translates actor’s intents into sequential sets of instructions
- Controls access to non-sharable resources with locks.
- Includes external systems
  - data bases
  - internal integrated applications
  - external integrated applications
  - payment systems, forward caching
JVM/OS

- Static Translation layer
  - Converts application’s set of sequential instructions into machine instructions

- Manages memory

- Includes operating system
  - Manages access to hardware resources
Hardware

- Bundle of non-sharable scarce resources with finite capacities
  - Defines capacities and rates
    - compute rate (CPU/clock speed)
    - data volumes
    - permanent and volatile storage areas
    - communication channel throughputs

Hardware

CPU, Memory, Disk IO, Network
System Resources

- **Actors**
  - Usage patterns

- **Application**
  - Locks, external systems

- **JVM/OS**
  - Memory, Hardware management

- **Hardware**
  - CPU, Memory, Disk IO, Network
Flow of Actions

Actors
- Usage patterns

Application
- Locks, external systems

JVM/OS
- Memory, Hardware management

Hardware
- CPU, Memory, Disk IO, Network

Actors drive the Application
- reflection of the user interface

Application feeds the JVM
- implementation dependent

JVM/OS feed the hardware
- implementation dependent

Hardware execute instructions
- limited by clock speed and capacities
Latency

- Amount of time spent waiting
- Green, work makes forward progress
- Yellow, work is queued is time in queues (latency)
Injection of Latency

- Overflow is queued at every level
  - Every queue injects latency into the response time
- Hardware lacks capacity or is slow
  - Users experience poor response times
- JVM is poorly coded or configured
  - Users experience poor response times
- Application is poorly coded or suffers from contention
  - Users experience poor response times
All we know!

Users are experiencing poor response times
Hardware Induced Latency

- Finite non-sharable scarce resource
  - components are points of serialization
  - heavy utilization of queues
- Defines capacities and rates
  - compute rate (CPU)
  - data volumes
  - permanent and volatile storage
  - communication channel throughputs

Hardware
- CPU, Memory, Disk IO, Network
OS Induced Latency

Hardware management

- Memory, context switching, IO sub systems

Symptoms include

- Relatively high CPU utilization
- High rates of context switching interrupt handling, and IO channel interactions
VM Induced Latency

- Java Heap memory management
  - object creation
  - garbage collection
- Symptoms include
  - high rates of object creation
  - Smalltalk: loci of object creation
  - low GC throughput
  - likely to see high CPU utilization
Application Induced Latency

- Locks blocks thread from making forward progress (queuing)
- Synchronous calls to external systems park threads
- Reduces load on underlying layers
- Symptoms include
  - Long response times
  - Inability to fully utilize CPU
How is Latency Expressed?

- Inflated user response times
- Ability or inability of the application to consume the CPU
Questions to Ask

- Condition: Slow response with under utilized CPU
  - what is keeping threads out of the CPU?
  - suggests up-stream bottleneck throttling threads

- Condition: Slow response with fully utilized CPU
  - who is the dominating consumer of the CPU
Consumers of the CPU

- Three candidates
  - Application
  - JVM
  - Operating System

- Looking for dominating consumer
- Each candidate suggests a different problem
- Each candidate suggest a different type of tuning activity
Dominating Consumer

- Application
  - CPU @ 100%
  - execution profiling
- JVM
  - CPU @ 100%
  - memory profiling
Application and JVM run in the same user process

JVM primary activities include
- JIT compilation
- Memory management
- Object creation
- Memory reclamation (garbage collection)

Differentiate by monitoring GC
Yet Another Wrinkle

- OS needs to be reported separately
- OS will prevent CPUs from being fully utilized
- CPU @ < 100%
- Kernel utilization is a significant portion of overall utilization
  - > 20% of total
- High rates of interrupt handling or context switching
Implications

- Application is asking too much of the OS
  - thread scheduling
    - spinning on lock
    - spinning on an IO channel
  - System calls
How do we start?

Setup a test environment, start benchmarking and investigate
Light Monitoring
### Command line tools that expose kernel counters

<table>
<thead>
<tr>
<th>proc</th>
<th>memory</th>
<th>page</th>
<th>faults</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>b</td>
<td>w</td>
<td>re</td>
<td>at</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>0</td>
<td>7876216</td>
<td>8497759</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>0</td>
<td>7876216</td>
<td>8497759</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>0</td>
<td>7879723</td>
<td>8497356</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>0</td>
<td>7879723</td>
<td>8493614</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td>7888432</td>
<td>8497759</td>
</tr>
</tbody>
</table>
Monitoring GC

- JVM switch -verbose:gc
- alternate -Xloggc:<filename>
- GCHisto analysis tool (new)
- HPJMeter
  - Heap utilizations
  - Pause times
  - Object creation rates
- More details later
Routes to the Problem
Synopsis

- GC throughput is very high
  - Object creation profiling
- Application execution time is very high
  - Execution profiling
Fixing performance problems

- Performance problems are caused by limited resources
- Which resource is limited?
- Applications may be
  - CPU bound
  - I/O bound
  - Space bound
  - “Lock bound” (contended)
How to decide which it is?

- **CPU bound**
  - CPU utilization consistently high
- **I/O bound**
  - CPU utilization not consistently high
- **Lock bound**
  - CPU utilization not consistently high
- **Space bound**
  - Any of the above!

These heuristics aren't precise enough, so tools are required to guide diagnosis.
Diagnosing CPU bound

- 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 without algorithms research!
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 than you expect.

“Why is so much of my profile in calls to this trivial little method?”
two ways to work out what code your application is doing

- trace and profiling

**Trace**

- Does not require specialist tools (but is better with them)
- Records every invocation of a subset of methods
- Gives insight into sequence of events
- In the simplest case, System.out.println

**Profiling**

- Samples all methods and provides statistics
Method profiling

- Validate algorithm performance
  - “Where is my application spending its time?”
  - “Am I getting benefit from that?”
- Identify methods where application is spending lots of time
  - Why are they being called? Can calls be reduced?
  - Can algorithm be optimized?
- Identify branches where time is being spent
IBM Health Center

IBM Monitoring and Diagnostic Tools for Java – Health Center is new performance tooling from IBM

- Live monitoring of applications
- Capabilities in a number of areas
  - Method profiling
  - Garbage collection
  - Locking
  - Configuration
- Visualisation and recommendations
Profiling with the Health Center

- Very low overhead profiling
  - Sampling profiler
  - Integrated into VM
  - No bytecode instrumentation
IBM Monitoring and Diagnostic Tools for Java – Health Center
IBM Java method trace

- Traces any Java methods
- Instrumentation-free, and no extra code required
- No fancy GUI, but very powerful
- Detailed information:
  - Entry and Exit points, with thread information and microsecond time stamps

Not overhead-free, but lower overhead than equivalent function implemented in Java
Controlling what is traced

- Can select methods on package, class or method name:
  - Package: methods=\{java/lang/\*\}
  - Class: methods=\{java/lang/String.\*\}
  - Method: methods=\{HelloWorld.main\}
  - Also ! operator and combination allowed:
    - methods=\{java/lang/\*,!java/lang/String\*\}
  - Possible to create huge volume of output, so use sensible method specifications!
Triggering events

- Can request certain actions occur when chosen methods are entered or exited
- Actions such as coredump, javadump, etc.
- Actions such as enabling trace!
- Can cause action to occur on n’th instance of trigger condition
- Can specify how many times the action occurs
- Multiple trigger types and actions can be specified
-Xtrace:print=mt,methods={myapp/MyTime*},resumecount=1,trigger=method{myapp/MyTime.main,resume,suspend}

- Only real time (79ms) is in the call to MyTime.test()
- Could now drill down into MyTime.test()
Profile Stock History
Benchmarking Stock History
Benchmark

- Process by which we will measure and investigate performance
- Use “the box” as a guide
- Define Actors
- Monitor system under load starting with hardware and working up
  - identify dominate consumer of the CPU
  - eliminate layers as you go

Diagram:
- Actors: Usage patterns
  - Application: Locks, external systems
    - JVM/OS: Memory, Hardware management
      - Hardware: CPU, Memory, Disk IO, Network
Preliminary Steps

- Encode usage patterns into a load test harness
  - Apache Jakarta-JMeter
  - Grinder as well as many commercial offering
- Configure benchmarking environment
  - mirror production
  - box tells us, change a layer, change the problem
- ensure adequate hardware for test harness
  - harness should not be the bottleneck
Preliminary Steps

- Review performance targets
  - response times for given load levels
- Plan on how to bring test to steady state
- Enable appropriate level of monitoring
  - too much will affect baseline results
    - system counters
    - garbage collection
    - external system performance
Establish a Workload

- Fixed amount of work measure the time
- Fixed amount of time measure the work
- Fixed load measure response time
Benchmarking Steps

Workflow used to guide our activities

do {
    baselinePerformance = application.baseline();
    profilingResults = application.profile();
    application.fixUsing(profilingResults);
    while (application.failsQA)
        application.debug();
    performance = application.baseline();
    user.setHappy(performance.meets(requirements));
} while (user.isUnhappy()) && (user.hasMoney());
Stumbling points

- Noise
- Randomization and caching
- Randomization and access patterns
- Complex usage patterns
- Complex system interactions
- Stubbing out external systems
  - mocks
public MockCreditAuthorizationServiceProvider( long someMeanValue,  
        long someRejectionRate) {
    this.setMeanServiceTime( someMeanValue);
    this.setRejectionRate( someRejectionRate);
}

public void setMeanServiceTime( long meanServiceTime) {
    this.serviceDistribution = new Exponential( meanServiceTime);
}

public void setRejectionRate( long rejectionRate) {
    this.percentDistribution = new java.util.Random();
    this.rejectionRate = (double)rejectionRate / 100.0;
}

public void authorize(AuthorizationRequest request) {
    try {
        Thread.sleep( this.serviceDistribution.nextLong());
    } catch (InterruptedException e) {}  

    if ( this.percentDistribution.nextDouble() > this.rejectionRate)  
        request.authorize();
}
Apache Jakarta-JMeter

Simulating Actors
Closed Systems

- Characterized by
  - limited number of users
  - user re-joins the line as soon as it is finished
  - source of artificial latency
  - sell throttling
  - difficult to regulate request arrival rates (thread starvation)
Open Systems

Characterized by

- unlimited number of users
- users arrive according to some schedule
- possible to flood a system
Open or Closed

- Closed harness when number of users is fixed
  - call center
- Open harness when number of users is unlimited
  - e-Commerce sites
Benchmarking Stock History

Stock History

hpq

<table>
<thead>
<tr>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adjusted Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-03-07</td>
<td>47.31</td>
<td>48.17</td>
<td>46.74</td>
<td>47.31</td>
<td>12587400</td>
<td>47.31</td>
</tr>
<tr>
<td>2008-03-06</td>
<td>48.02</td>
<td>48.80</td>
<td>47.69</td>
<td>47.78</td>
<td>16083800</td>
<td>47.78</td>
</tr>
<tr>
<td>2008-03-05</td>
<td>48.59</td>
<td>48.85</td>
<td>47.80</td>
<td>48.31</td>
<td>12472600</td>
<td>48.31</td>
</tr>
<tr>
<td>2008-03-04</td>
<td>47.30</td>
<td>48.52</td>
<td>46.99</td>
<td>48.45</td>
<td>16189900</td>
<td>48.45</td>
</tr>
<tr>
<td>2008-03-03</td>
<td>47.62</td>
<td>47.88</td>
<td>47.14</td>
<td>47.60</td>
<td>12177100</td>
<td>47.60</td>
</tr>
<tr>
<td>2008-02-29</td>
<td>48.11</td>
<td>48.62</td>
<td>47.61</td>
<td>47.77</td>
<td>14187000</td>
<td>47.77</td>
</tr>
<tr>
<td>2008-02-28</td>
<td>48.61</td>
<td>48.80</td>
<td>48.16</td>
<td>48.55</td>
<td>15779600</td>
<td>48.55</td>
</tr>
<tr>
<td>2008-02-27</td>
<td>49.00</td>
<td>49.49</td>
<td>48.65</td>
<td>48.94</td>
<td>14493400</td>
<td>48.94</td>
</tr>
<tr>
<td>2008-02-26</td>
<td>47.90</td>
<td>49.69</td>
<td>47.64</td>
<td>49.20</td>
<td>20432300</td>
<td>49.20</td>
</tr>
</tbody>
</table>
IBM Garbage Collection
What is garbage collection?

- Garbage collection is automatic memory management
  - Memory which has been dynamically allocated but which is no longer in use is reclaimed without intervention by the application
  - Only memory which is unreachable is freed
Garbage collection - not just

“Garbage collection” is a shorthand for automatic memory management
“Garbage collection” is a shorthand for automatic memory management.

Memory management involves a lot more than just reclaiming unused memory.
“Garbage collection” is a shorthand for automatic memory management

Memory management involves a lot more than just reclaiming unused memory

Memory management includes

- freeing memory
- allocating memory
- arranging memory
“Garbage collection” is a shorthand for automatic memory management

Memory management involves a lot more than just reclaiming unused memory

Memory management includes

- freeing memory
- allocating memory
- arranging memory

All of these are important and must be considered when choosing and tuning garbage collection policies
Garbage collection - not just a

- Garbage collection pauses are easily identifiable pauses when the application is prevented from doing 'real' work, so garbage collection is often considered to be a necessary evil.

- Garbage collection can provide performance benefits:
  - Faster freeing of memory
  - Faster memory allocation
  - Faster memory access
Freeing memory

- Even without garbage collection, freeing memory takes time
  - The mean malloc/free overhead in C applications ranges between means of 7% and 20%, depending on the allocator. For one application the overhead of malloc/free was 53%
  - Some collectors can free memory much more quickly than free() can
  - When well tuned, the cost of freeing with a copying collector can be less than 1 instruction per object
Allocating memory

Allocating memory takes time
Allocating memory

- Allocating memory takes time
- Allocating memory is particularly slow when
  - The heap is fragmented
  - Multiple threads are contending for allocation locks on the heap
Allocating memory takes time

Allocating memory is particularly slow when

- The heap is fragmented
- Multiple threads are contending for allocation locks on the heap

Garbage collection can help with both of these problems
Allocating memory takes time

Allocating memory is particularly slow when

- The heap is fragmented
- Multiple threads are contending for allocation locks on the heap

Garbage collection can help with both of these problems

Fragmentation

- Rearrange objects on contention to ensure no lengthy free-list searches are required
Allocating memory

- Allocating memory takes time
- Allocating memory is particularly slow when
  - The heap is fragmented
  - Multiple threads are contending for allocation locks on the heap
- Garbage collection can help with both of these problems
  - Fragmentation
  - Contention
  - Batch-allocates chunks of heap to threads so they don't
Accessing memory

- Not all memory access is equally fast
- Garbage collection can speed up memory access by rearranging objects in memory
- Since memory access is one of the main things an application does, this can make a big performance difference
- This is an ongoing area of research:
  - Using the JIT to profile of hot fields and then rearrange objects on collection gave throughput improvements of up to 45% (Huang, 2004)
What a cache is, and why you need it

Memory access is very slow compared to instruction processing.

To prevent memory access from being a serious bottleneck, memory caches are added.

Most modern systems have a hierarchy of caches of increasing speed and decreasing size.

Access to objects already in the cache is far faster than pulling an object into the cache.

Improvements in processor speed continue to outpace improvements in memory speed, so caches are becoming increasingly important.
Locality

- When an object is loaded into the cache its neighbours are also loaded into the cache.
- This makes relative positions of objects important to performance.
- Spatial locality describes how spatially close objects are to objects which are accessed at similar times.
- An application will go much faster if objects which tend to be accessed around the same time are located near one another.
- Locality can be improved.
Garbage collection and the

- Garbage collection can hinder or help interaction with the cache
Garbage collection and the

- Garbage collection can hinder or help interaction with the cache
- Cache pollution
- Depending on the algorithm, garbage collection may visit quite a lot of memory during a collection
- This means the cache won't be right for the application because it will be full of stuff the garbage collection just visited
Garbage collection and the cache

- Garbage collection can hinder or help interaction with the cache

  - Cache pollution

    - Depending on the algorithm, garbage collection may visit quite a lot of memory during a collection
    - This means the cache won't be right for the application because it will be full of stuff the garbage collection just visited

  - Compaction

    - Means objects are closer to their neighbours and more
Garbage collection and the cache

- Garbage collection can hinder or help interaction with the cache

- Cache pollution

- Depending on the algorithm, garbage collection may visit quite a lot of memory during a collection

- This means the cache won't be right for the application because it will be full of stuff the garbage collection just visited
Garbage collection and the

- **Compaction**
  - Means objects are closer to their neighbours and more likely to be in the cache

- **Re-arrangement**
  - Means objects are closer to their friends and more likely to be in the cache at the right time
Why provide more than one

Most JVMs provide several policies for different requirements and workloads

None of the policies are bad, but the default is not necessarily best in every circumstance

They differ in the following:

- When and how is the work done?
- What happens to garbage?
- How is the heap laid out?
When and how is the work

- **Stop-the-world**
  - All application threads are stopped during garbage collection

- **Incremental**
  - Garbage collections are divided into smaller partial collections
  - Reduces application pauses

- **Concurrent**
  - Work appears to happen concurrently with application work
What happens to garbage?

- **Free-list collectors**
  - The heap is searched for unreachable objects which are added to a list of free space
  - New objects are allocated from the free list
  - When the heap becomes fragmented, it is compacted by rearranging objects

- **Copying collectors**
  - Reachable objects are copied to fresh heap; what's left is garbage
  - Collecting garbage is free!
How is the heap laid out?

Flat heap

Everything is in one unstructured area
How is the heap laid out?

- Heap with large object area
  - Very large objects are kept away from normal objects
  - Large objects are expensive to allocate
  - Large objects are expensive to compact
Generational Heap Layout

- Exploits the observation that most objects die young
- Divides the heap into generations
  - Younger generations are collected more frequently
  - If a copying collector is used, collecting the young generations is very fast since collecting dead objects is free
But which policy is best?

- It depends!
- Depends on
  - Application characteristics
  - Rates of object creation and death
  - Size of objects
  - Patterns of object access
  - Size of required heap
  - System characteristics
Opt throughput policy

What kind of policy is this?

- Stop-the-world
- Free-list
- Flat heap

Default policy

Has an advantage if there is lots of spare heap
Optavgpause policy

What kind of policy is this?

Concurrent
Free-list
Flat heap

Shorter pauses than optthruput
But small pauses are not guaranteed
Compaction causes long pauses
If the heap is too small pauses may be very long
Gencon policy

What kind of policy is this?

- Generational
- Copying collection in nursery (youngest generation)
- Concurrent free list collection in tenured area (older generation)

- Combines small pauses with excellent throughput
- Allocation is very fast
- Has an advantage in small/restricted heaps
- Suitable when rate of data churn is high
The relative impact of

- Even when a garbage collector spends a lot of time paused, application performance may be better
The relative impact

- Even when a garbage collector spends a lot of time paused, application performance may be better.

Why?

- Garbage collection is not just garbage collection!
The relative impact

- Even when a garbage collector spends a lot of time paused, application performance may be better

- Why?
  - Garbage collection is not just garbage collection!
  - Investing more time in collection can give big wins for allocation and access

- Example: compacting the heap.

- Collectors which compact the heap
Enabling verbose gc

- Relatively lightweight logging, so usually has minimal performance impact
- Enable with `-verbose:gc`
- Different vendors may also have extra options.
- For example:
  - IBM JRE provides `-Xverbosegclog:file` to send the logs directly to a file
  - As the verbose gc logs are XML in 1.5.0 and higher, this avoids mixing the logs in with stdout and is a good idea
IBM Monitoring and Diagnostic Tools for Java – GC and Memory Visualizer (formerly known as EVTK) is a verbose GC analysis tool.
GC and Memory Visualizer

- Handles verbose GC from all versions of IBM JVMs
  - 1.4.2 and lower
  - 5.0 and higher
  - zSeries
  - iSeries
  - WebSphere real time
- ... and Solaris platforms
- ... and HP-UX platforms
Health Center and GC

- Health Center can diagnose and help solve GC problems
- GC visualisations
- Recommendations

A screenshot of the IBM Monitoring and Diagnostic Tools for Java - Health Center showing various metrics and graphs related to garbage collection.
The GC and Memory
GC and Memory Visualizer

- Analyses heap usage, heap size, pause times, and many other properties
- Compare multiple logs in the same plots and reports
- Many views on data
  - Reports
  - Graphs
  - Tables
- Can save data to
  - HTML reports
IBM Support Assistant

- Hosting for Serviceability Tools across product families
- Automatic problem determination data gathering
- Assist with opening PMR’s and working with IBM Support
Differences

Health Center

- Live monitoring (agent on VM startup)
- Integrated with other performance functionality

GC and Memory Visualizer

- Reads verbose GC
- Allows comparison of logs
- Greater platform and version support
- Greater depth of GC information available
Scaling concerns

- How does your application footprint scale with the number of users?

- Linear scaling will be bad unless

  - Restricted number of users
  - Not desirable in most commercial scenarios
  - Unlimited heap size
  - Not possible at all
Assessing Footprint

- Is the footprint what you expect?
- If not, why not?
  - Excessive caching
  - Excessive cloning
  - Bloated object structures
- Solution may be to reduce application's memory usage rather than increase the heap size
- Sometimes the solution may be to increase application's memory usage if it's using less than expected

“If my footprint’s that small then I can cache all that stuff and speed up my application”
Don't forget native memory

- Java applications use – and may leak - native memory
- Low occupancy is no guarantee an application is not space bound.
- Native memory use is not logged in verbose GC
- Memory pressure and even OutOfMemory errors may occur even though there is lots of room in the heap
- GCMV can visualise native memory collected with platform-specific scripts
Visualizer

- Health Center
  - Live monitoring (agent on VM startup)
  - Integrated with other performance functionality

- GC and Memory Visualizer
  - Reads verbose GC
  - Allows comparison of logs
  - Greater platform and version support
  - Greater depth of GC information available
IBM Monitoring and Diagnostic Tools for Java – GC and Memory Visualizer (formerly known as EVTK) is a verbose GC analysis tool.

- Handles verbose GC from all versions of IBM JVMs
  - 1.4.2 and lower, 5.0 and higher
  - zSeries, iSeries
  - WebSphere real time
- ... and Solaris platforms
- ... and HP-UX platforms
GC and Memory Visualizer

- Analyses heap usage, heap size, pause times, and many other properties
- Compare multiple logs in the same plots and reports
- Many views on data
  - Reports
  - Graphs
  - Tables
- Can save data to
  - HTML reports, JPEG pictures, CSV files
Health Center and GC

Health Center can diagnose and help solve GC problems

GC visualizations

Recommendations

The application seems to be using some quite large objects. The largest request which triggered an allocation failure and was recorded in the verbose gc log was for 4766 KB.

The mean occupancy is 61%. This is close to optimal, so you do not need to tune your heap size.

The memory usage of the application does not indicate any obvious leaks. 2 sources: idle.

Mean garbage collection pause (ms) 8.38
Rate of garbage collection 18:23 MB/minute
Largest memory request (KB) 4766
GC Node unknown
Time spent in garbage collection pauses (%) 99.5
Proportion of time spent paused (%) 99.5
Mean interval between collections (minutes) 0.03
Number of collections 204
Mean heap unusable due to fragmentation (MB) 14.0
GC and Memory Visualizer

[Diagram showing memory usage and garbage collection metrics]
IBM Support Assistant

- Hosting for Serviceability Tools across product families
- Automatic problem determination data gathering
- Assist with opening PMR’s and working with IBM Support
Sun Garbage Collection
Sun Generational Spaces

Young

{ Eden
   Survivor 1
   Survivor 2
}

Old

{ Tenured
   Perm
}
Object Life-cycle

Young

Eden
Survivor 1
Survivor 2

Old

Tenured
Perm
Sun Generational

Young

Eden

Survivor 1

Survivor 2

Old

Tenured

Perm
G1 Collector

- 1 1/2 collector
- Generational
  - evacuation
- Parallel and Concurrent
- Pause prediction model
  - guideline for the collector, not a guarantee
G1 Spaces

- Parallel
- Young - green
- Old - blue
- Large objects stored in contiguous spaces
G1 Young Collection

- 1 Megabyte segments
- Concurrent mark
- Evacuation pause
G1 Old Collection

- Concurrent mark
- remark pause
- Per region liveliness calculation
  - reclaim empty regions
- Low liveliness regions evacuated and reclaimed by young collector
  - hence 1 1/2 collector
G1 Old Collection

- No old space collector
- 1 1/2 collection
G1 Remembered Sets

- If you move objects you have to know how to find them
- C pointer bug
- Remember set traces evacuated root objects
- References will follow the trail and then update themselves
- Allows single regions sweeps
What’s Leaking?
Memory leaks in Java?

Memory leaks happen when objects which are no longer required still use up memory.

Two kinds of memory leak:

- Losing a reference to an object which is no longer in use
- Holding on to a reference for an object which is no longer in use
Memory leaks in Java?

- Garbage collection eliminates the first kind, but not the second
- Memory leaks are never desirable
  - Long running applications with memory leaks will eventually crash
  - Short-lived applications with memory leaks may still suffer performance degradations
A heap dump is a **snapshot of objects that are alive** at one point in time.

- contains:
  - Objects: fields, references, primitive values, …
  - Classes: class loader, super class, static fields, …

A heap dump does **not** contain
- where the objects have been created
- which objects could be garbage collected
Types of Dumps

- VM dumps differ between vendors (e.g. IBM and Sun)
- Different dumps within vendors
  - Java (small)
  - Heap (medium)
  - System (large)
- Diagnostic Tooling Interface for Java (DTFJ) hides differences between dumps from different VMs
- DTFJ is being standardized via JSR 326
- DTFJ support for IBM System dumps only
Types of Dumps

- Diagnostic Tooling Interface for Java (DTFJ) hides differences between dumps from different VMs
- DTFJ is being standardized via JSR 326
- DTFJ support for IBM System dumps only
  - Need a VM option to get a system (= process) dump on OutOfMemoryError
  - System dumps are post-processed with JExtract
Eclipse Memory Analyzer

- Standalone application built on Eclipse's Rich Client Platform
- Designed to analyze very large heaps
- Simplifies memory analysis
- Extensible
- Supports hprof and IBM dumps
- Free for download at www.eclipse.org/mat
IBM Support for Memory

- Adapter for MAT available from IBM to read the dumps
- Adapter uses DTFJ to open the dump & read the object/class data, then passes the data into MAT
- MAT then displays data just like an HPROF dump
Getting the Memory Analyzer

- Memory Analyzer @ Eclipse:
  - www.eclipse.org/mat

- IBM DTFJ-adapter for Eclipse Memory Analyzer:

- Memory Analyzer Wiki @ SAP:
  - www.sdn.sap.com/irj/sdn/wiki?path=/display/Java/Java+Memory+Analysis

- Blogs:
  - dev.eclipse.org/blogs/memoryanalyzer
Using Eclipse Memory Analyzer to discover what's eating memory
Using Eclipse Memory Analyzer to discover what's eating memory
Lock Profiling
Diagnosing lock bound

In multi-threaded applications, threads will fight over shared resources

Some synchronization is necessary to maintain functional correctness

Excessive synchronization can cause significant application delays

As the number of threads increases, the performance impact of locking will increase

Diagnose and identify needlessly contended locks to eliminate bottlenecks

A contended lock is the opposite of a contented lock!
Identifying contended locks

- Functional superset of IBM's Java Lock Analyzer (JLA)
- Provides profiling data on monitors used in Java applications and the JVM:
  - Counters associated with contended locks
  - Total number of successful acquires
  - Recursive acquires
  - Frequency with which a thread had to block waiting on the monitor
  - Cumulative time the monitor was held.
  - Class of the monitor object
Health Center for Locking

- Health Center includes a Locking perspective
- Allows points of contention which are preventing scaling to be identified
What do the bars mean?

- The Health Center provides very detailed information on locking and synchronization in the table below the chart.
- In most cases the chart will be enough.
- The height of the bar indicates how often threads were blocked waiting for the lock.
- The color of the bar indicates what fraction of the attempts were unsuccessful.
Demo title
Puzzle
Thanks for your attention!

(references)