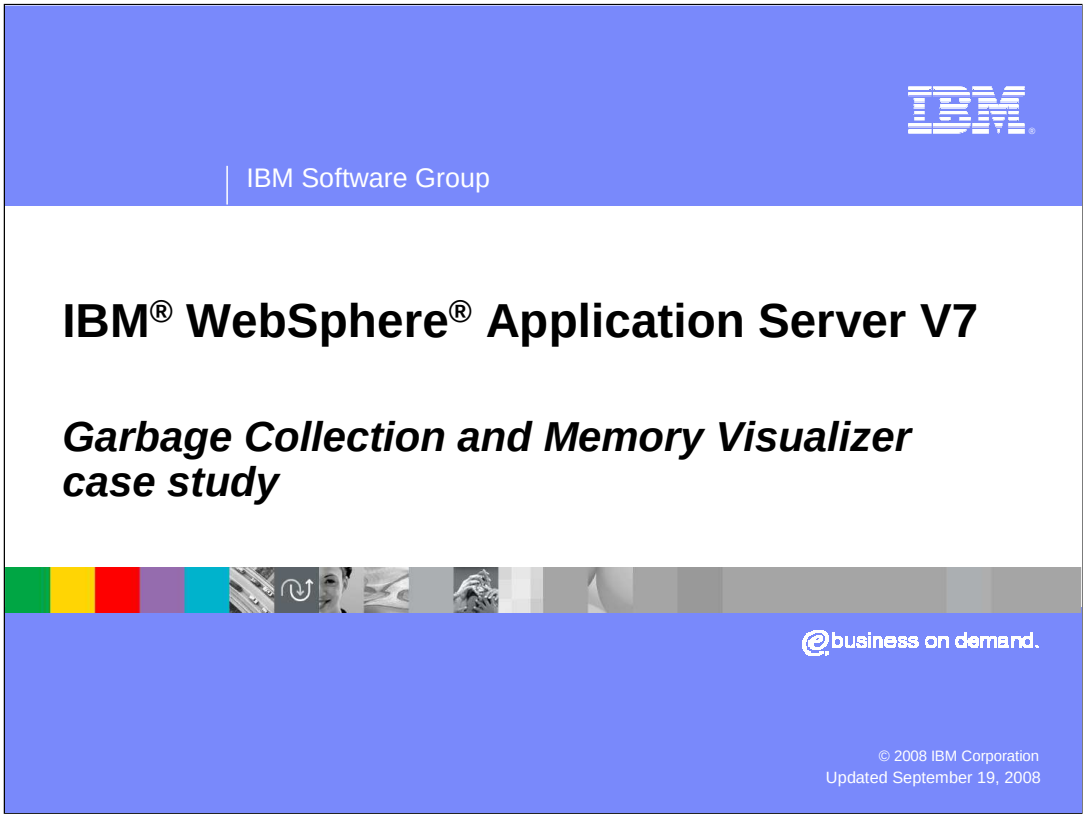
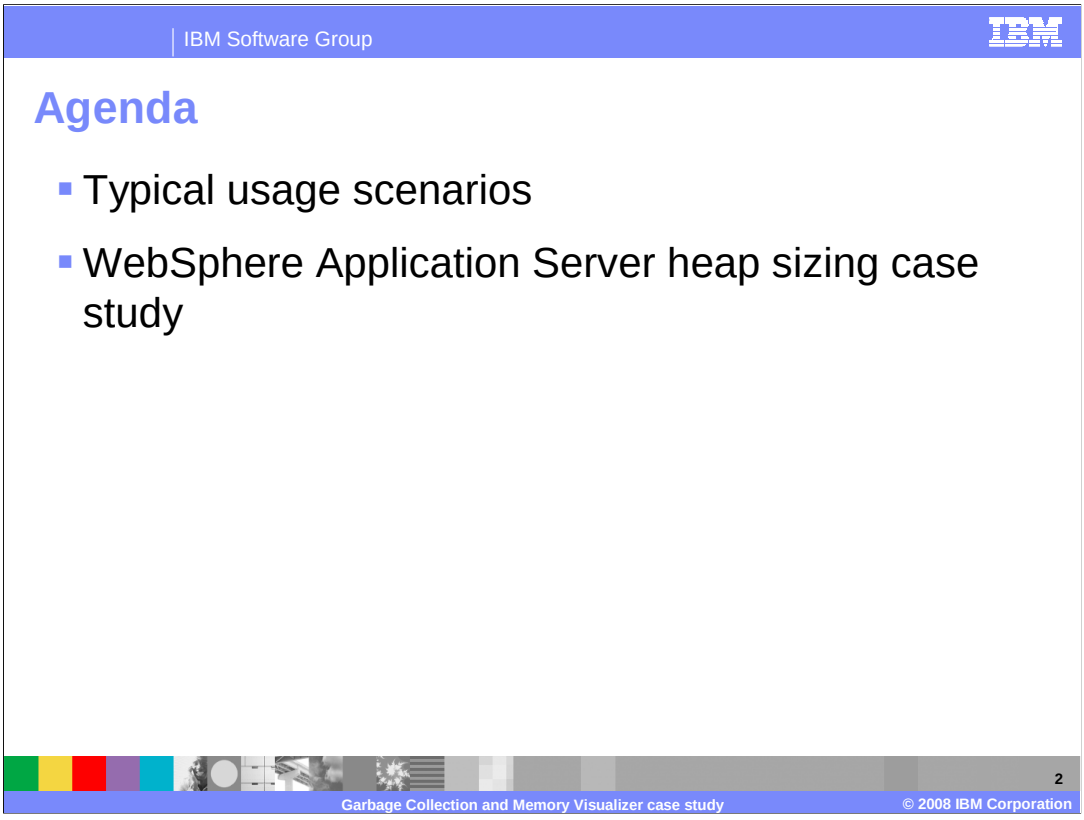


This presentation will walk you through a case study that shows how to use data provided by the Garbage Collection and Memory Visualizer to help diagnose a heap sizing problem.

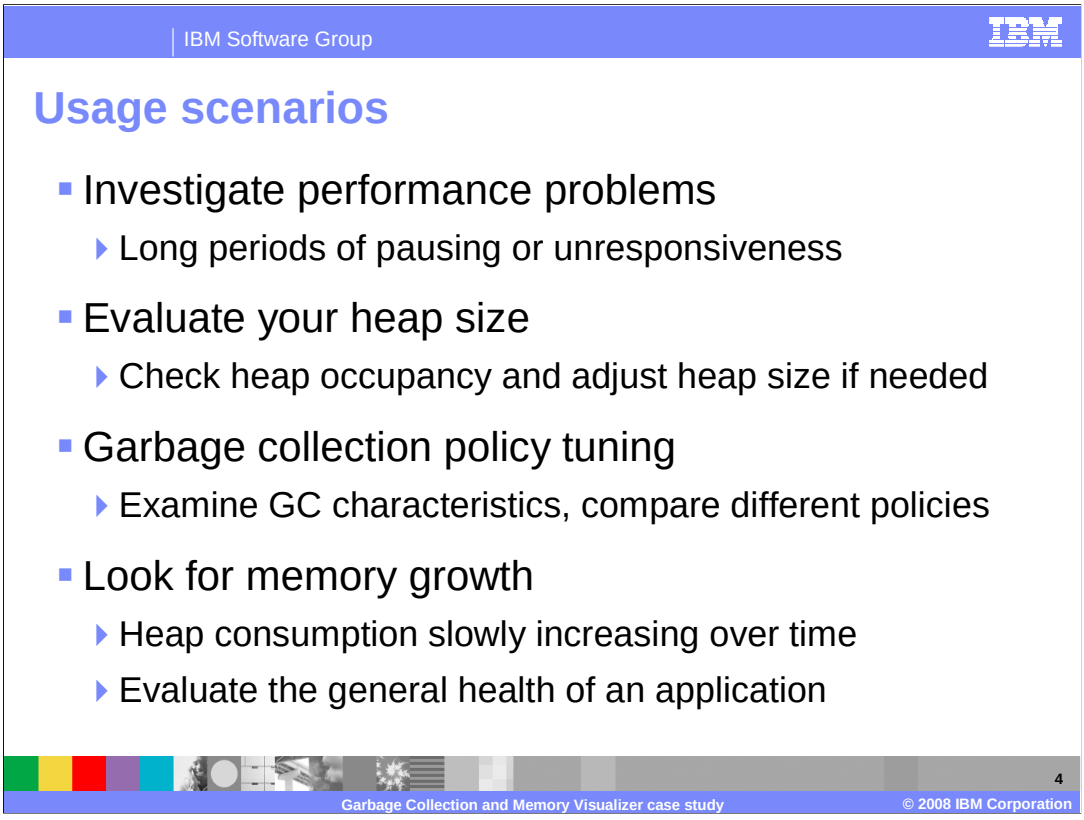


The first portion of the presentation will briefly describe some situations where you might want to use the Garbage Collection and Memory Visualizer. The second part of the presentation will use the Garbage Collection and Memory Visualizer to explore several different garbage collection characteristics captured in the verbose garbage collection logs from a test run of the WebSphere Application Server that is acting a little sluggish. You will see that the garbage collection data can provide insight into why the application is running poorly, and how to fix it.

Section

Usage scenarios

Garbage collection data analysis can be useful in helping you understand your Java application behavior in many settings. This first section of the presentation describes a few scenarios in which you might want to do some GC data analysis, using the Garbage Collection and Memory Visualizer.



When experiencing poor Java application performance, people sometimes have a tendency to jump to the conclusion that the garbage collector is to blame, but that is often not the case. One way to check whether garbage collection might be bogging your application down is by examining the logs. Is your application having to spend a large percentage of its time collecting garbage? Are you noticing period of unresponsiveness in your application that correspond to long garbage collection pause times in your logs? If so, you can probably tune your garbage collection policies to get better behavior. If you have concerns about your overall heap size, check out the Report tab portion of the GC and Memory Visualizer. It contains information on your heap utilization and offers recommendations for adjusting your heap size. To see how using different garbage collection policies will affect your application, gather logs for test runs using the different policies, and then load those logs into a GC and Memory Visualizer workspace. View and compare their characteristics on a single plot. To check the overall health of your application, use the GC and Memory Visualizer to examine your application's heap consumption over a long period of time. If your heap usage is slowly creeping up over time, you might have a memory leak in your application. If you suspect a memory leak, IBM provides another tool called the Memory Dump Diagnostic for Java that can help you isolate the source of the leak. The Memory Dump Diagnostic for Java tool is also available as a plug-in for IBM Support Assistant.

Section

Heap utilization case study

The next section of the presentation will show you how to use the Garbage Collection and Memory Visualizer to understand why an application is acting sluggish. In this example, you will see that there is not enough heap space allocated to the application.

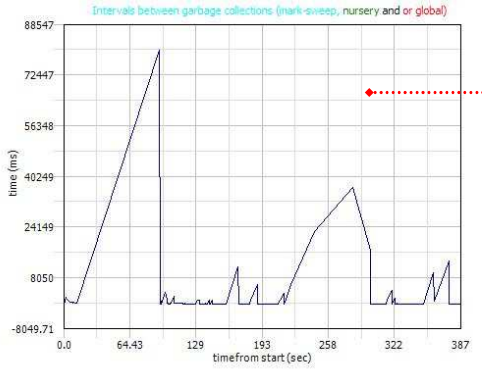
Scenario description

- Running WebSphere Application Server with typical usage
  - Using the administrative console
  - Deploying applications
  - Accessing applications
- Notice that performance is sluggish
  - Applications taking a long time to load
  - Errors showing up in the administrative console

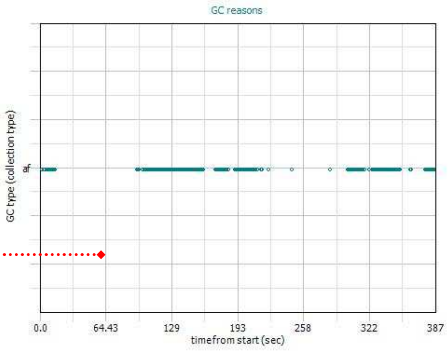
This problem scenario is taken from an artificially constrained test run of the WebSphere Application Server. The maximum heap size was manually restricted to 60 megabytes in order to force heap utilization issues. This is in no way a recommended configuration; the default minimum heap configuration in the application server is 256 megabytes. Still, it is useful to restrict the heap for learning purposes.

In this case, the application server was running with typical usage – working in the administrative console, deploying applications, and accessing Web applications. The performance of the application server was sluggish, applications were taking a long time to load, and error screens started showing up in the administrative console. The next several slides show GC and Memory Visualizer graphs using a verbose garbage collection log from this test run.

Garbage collection trigger graphs



This graph shows intervals between garbage collection cycles. Notice that it shrinks to near 0ms for extended periods (horizontal portions of the graph)



Each dot in this graph represents a garbage collection cycle. All of these cycles ran for reason 'af' – allocation failure. Notice that the dot concentration lines up with the trigger graph.

The graph in the upper left of the display shows the intervals between garbage collection cycles. As the virtual machine is starting, there are several garbage collection cycles close together, and then a fairly long pause before additional cycles kick in around 80 seconds into the test run. From that point on, there are several portions of the graph that essentially look like horizontal lines, which means that the pause time between GC cycles has shrunk to near zero. During these periods, the virtual machine is constantly trying to collect garbage and is not free to perform the normal work of the application, and it is likely that a user might experience sluggish performance and slow application response times. Garbage collection can be triggered by a variety of events, including a forced system call or an allocation failure. The graph in the lower right of the display shows that all of the garbage collection cycles in this test run were the result of allocation failures, shown by the reason code 'af' in the graph. Each dot in the trigger graph represents one garbage collection cycle. Notice that there are portions of the graph where the dots are highly concentrated, which indicates that many garbage collection cycles are occurring in a short period of time. The sections of high concentration in the 'GC reasons' graph correspond to the sections of the top graph with near zero intervals between garbage collection cycles.

Heap usage and occupancy recommendation

Occupancy recommendation

Summary

Occupancy recommendation

The mean occupancy is 98%. This is a bit high, so you may improve collection times by increasing your heap size.

Summary

Largest memory request (bytes)	1409048
Allocation failure count	563
GC Mode	optthruput
Proportion of time spent in garbage collection pauses (%)	34.99
Concurrent collection count	0
Proportion of time spent unpaused (%)	65.01
Forced collection count	0
Number of collections	298
Mean interval (sec)	0.62
Mean pause (ms)	217

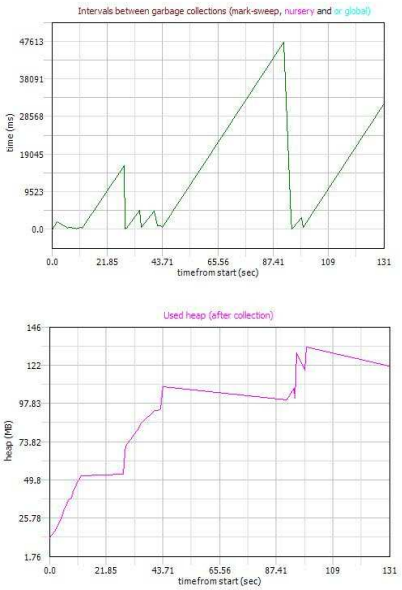
The summary report shows that mean heap occupancy is 98% and that the application is spending over a third of its time doing garbage collection

This graph shows heap usage after garbage collection; it jumps up to around 60M and stays there

The graph in the lower portion of the display shows the amount of used heap after each garbage collection cycle. Recall that, in this testing scenario, the heap has been artificially constrained to 60 megabytes, and notice that the used heap jumps up to 60 after running for about two minutes, and then stays there for the duration of the test. The summary report in the Garbage Collection and Memory Visualizer provides garbage collection summary statistics and recommendations on heap sizing. In this case, the mean heap occupancy for the test run is 98%, which is very high. The application is having to spend over one third of its processing time trying to do garbage collection due to allocation failures. The recommendation in the summary report is to increase the heap size.

Results after increasing the heap size

- Increased the maximum heap size from 60M to 256M
  - 256M is the default maximum heap size for WebSphere Application Server
- Mean heap occupancy shrank to 64%
- Spent less than 3% of processing time in garbage collection
- Reduced average pause time by more than half
  - From 217 ms to 98 ms
- Server performance no longer appeared sluggish



After increasing the maximum heap size from 60 megabytes back to the default value of 256 megabytes, the overall health of the application's garbage collection profile improved substantially. The mean heap occupancy shrank from 98% to 64%; the time spent doing garbage collection was cut from almost 35% to 3%; and the average pause time for a garbage collection cycle was reduced by more than half. The GC and Memory Visualizer line plots from the "healthy" test run look much better. The top graph shows the intervals between garbage collection cycles, and in this case, the graph has a typical saw-edge pattern. Unlike in the poor performing test run, here there are no periods where the intervals shrink to near zero for an extended period. Similarly, the used heap profile is much healthier than in the constrained test run.

Section

Summary and references

This section contains a summary and links to reference materials.

Summary

- The Garbage Collection and Memory Visualizer can be used to analyze Java application behavior
  - Performance concerns, garbage collection policy tuning, heap size evaluation, overall application health
- Use multiple datasets and characteristics to investigate issues

The Garbage Collection and Memory Visualizer is useful for analyzing Java application behavior in a variety of situations, from identifying potential performance bottlenecks to understanding the impacts of different garbage collection policies to tuning your heap size. The tool allows you to display over forty verbose garbage collection data characteristics, so when you are using it, try to take advantage of multiple sets of data to gain better insight into your application's behavior.

References

- IBM Support Assistant
  http://www.ibm.com/software/support/isa/
- Diagnostics Guide
  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp
- developerWorks® articles about garbage collection
  http://www.ibm.com/developerworks/java/library/j-ibmjava2/
  http://www.ibm.com/developerworks/java/library/j-ibmjava3/

Here are some links to additional resources.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASv7_GCMVCaseStudy.ppt

This module is also available in PDF format at: ../WASv7_GCMVCaseStudy.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

developerWorks IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.