



IBM Software Group

# WebSphere Application Server V6.1 Problem determination guide

## Introduction to out-of-memory problems



@business on demand.

© 2009 IBM Corporation  
Converted to video May 18, 2015

This unit covers the problem determination techniques associated with out of memory problems in WebSphere® Application Server.

## What is a `java.lang.OutOfMemory` error?

- Java™ virtual machine error that indicates a condition where not enough free memory is available to allocate an object
- Can be caused by:
  - ▶ Undersized heap
  - ▶ Memory fragmentation
  - ▶ Memory leak in the Java code
  - ▶ Not enough space in the native memory

When the Java virtual machine, or JVM, tries to allocate an object and it fails, it runs garbage collection to free up heap space from objects that are no longer being used. If the object cannot be allocated after garbage collection, the JVM creates an `OutOfMemoryError`.

## Memory leak in the Java code

- No matter what the JVM maximum heap size is set to, the heap will still run out of space.
- Increasing the maximum heap size only causes the problem to take longer to occur.
- One or more objects are taking up a high percentage of the JVM heap:
  - ▶ A few large objects
  - ▶ Thousands of instances of small objects
- Memory leaks can also occur in native code

Memory is not explicitly allocated and deallocated in Java, like in C code, but it is still possible to create a memory leak. One example is to save an object into some type of collection. If the collection is a class object, and the class always stays loaded, the object will never be removed from the collection. If objects are continuously added to the collection, the collection can grow until it consumes a significant portion of the Java heap. The misuse of object caches is a common cause of memory leaks seen in applications running in WebSphere. Another commonly seen misuse of cache in WebSphere is the HTTP maximum Sessions parameter in WebSphere. HTTP Sessions can be written to store very large objects. If the HTTP maximum sessions parameter is set too high, then it will again consume a large portion of the JVM heap.

## Not enough native memory

- Insufficient memory available in the native memory segment
- There is more than sufficient space in the JVM heap, but the allocation still fails
- The JVM is not necessarily trying to allocate a large object but rather memory is not available in the native memory space



When the Java Virtual Machine is unable to find enough contiguous memory in the heap for object allocation, and it has already called the garbage collection routine, it will then attempt to grow the JVM heap. In order to grow the JVM heap the JVM must request, and be granted, system memory from the operating system. If the operating system is unable to provide memory to the JVM for heap expansion, the JVM will throw an out of memory. This can also occur within Java native interface code.

## The native heap

- Allocated using **malloc()** and therefore subject to memory management by the OS
- Used for virtual machine resources like class loader and garbage collector
- Used to underpin Java objects like threads and classes
- Used for allocations by JNI code
- Size can not directly be controlled. To increase this memory space decrease the Java heap size.



The native heap is the portion of the memory for the JVM process that does not include the Java heap. This space is used for holding data for the JVM's internal components such as the garbage collector and class loader. Additionally, native memory is used to underpin or support Java data structures like threads and classes. The size of the native memory cannot be directly controlled. It is basically total memory minus the maximum Java heap size, and is further bounded by operating system constraints and libraries..

## Using TPV to anticipate an OutOfMemory error

- Tivoli® Performance Viewer (TPV) runs in the WebSphere administrative console
- Uses Performance Monitoring Infrastructure (PMI) to capture information about the WebSphere run time.
- Provides graphical display of the Captured PMI Data



The Tivoli Performance Viewer is embedded within the WebSphere Administrative Client. It uses the Performance Monitoring infrastructure to capture information about the WebSphere runtime, such as the JVM heap size.

## OutOfMemory indicators of the javacore file

- Look to see if the dump was due to an OutOfMemory condition

```
1TISIGINFO      Dump Event "throw" (00000010) Detail
                "java/lang/OutOfMemoryError" received
```

- Check MEMINFO subcomponent output (values in hexadecimal)

```
0SECTION      MEMINFO subcomponent dump routine
NULL          =====
1STHEAPFREE    Bytes of Heap Space Free: 4bca320
1STHEAPALLOC   Bytes of Heap Space Allocated: c000000
```

- Check GC History

```
1STGCHTYPE     GC History
3STHSTTYPE     05:15:28:807904721 GMT j9mm.81 - J9AllocateIndexableObject()
                returning NULL! 167772176 bytes requested for object of class 101189C8
```



When looking at a javacore file there are some key eye-catchers that indicate an OutOfMemory condition. One such eye-catcher is in the dump event section. The javacore should indicate very clearly that the dump was created due to an OutOfMemoryError.

## How to obtain a verboseGC log

- Verbose GC is an option provided by the JVM run time
- Enables a garbage collection log
  - ▶ Interval between collections
  - ▶ Duration of collection
  - ▶ Compaction required
  - ▶ Memory size/memory freed/memory available
- Typically writes to native\_stderr
  - ▶ Varies depending on platform and WebSphere version
  - ▶ Some overhead because of disk I/O, but typically minimal unless thrashing

The verbose garbage collection or verbosegc output is the diagnostic data used to identify what type of OutOfMemoryError condition is occurring on the system.



## OutOfMemory: interpret verboseGC output

- Inspect the allocation failure
  - ▶ Find the allocation failure that caused the OutOfMemoryError
  - ▶ Check the size of the object to be allocated
  - ▶ Determine the size of the Java heap
  - ▶ Check to see what percentage of the heap is free
- Look for the pattern in previous allocation failures
  - ▶ Do you continuously get allocation failures that cause the JVM to increase the heap?
  - ▶ Is this an isolated allocation failure caused by a request for a large object?

Once you locate the OutOfMemoryError in the log, you want to first examine the allocation failure that caused the event to occur. You need to check the size of the object to be allocated, and the current size of the Java heap. You can also observe what percentage of the heap is currently free.

## Error messages output showing dump events

```
JVMDUMP006I Processing Dump Event "throw", detail
"java/lang/OutOfMemoryError" - Please Wait.

JVMDUMP007I JVM Requesting Heap Dump using
'C:\Temp\Dumps\heapdump.20070715.173835.1012.phd'

JVMDUMP010I Heap Dump written to
C:\Temp\Dumps\heapdump.20070715.173835.1012.phd

JVMDUMP007I JVM Requesting Java Dump using
'C:\WASv61\profiles\AppSrv01\javacore.20070715.173
835.1012.txt'

JVMDUMP010I Java Dump written to
C:\WASv61\profiles\AppSrv01\javacore.20070715.1738
35.1012.txt

JVMDUMP013I Processed Dump Event "throw", detail
"java/lang/OutOfMemoryError"
```

The example on this slide shows output from the verbose garbage collection log and shows different JVM-related diagnostic messages. In this case both a heapdump and a javadump were configured to be created if an OutOfMemory exception was thrown. Search the native\_stderr.log output for "Heap Dump or "Java Dump" and look for the failure entry directly preceding the output.

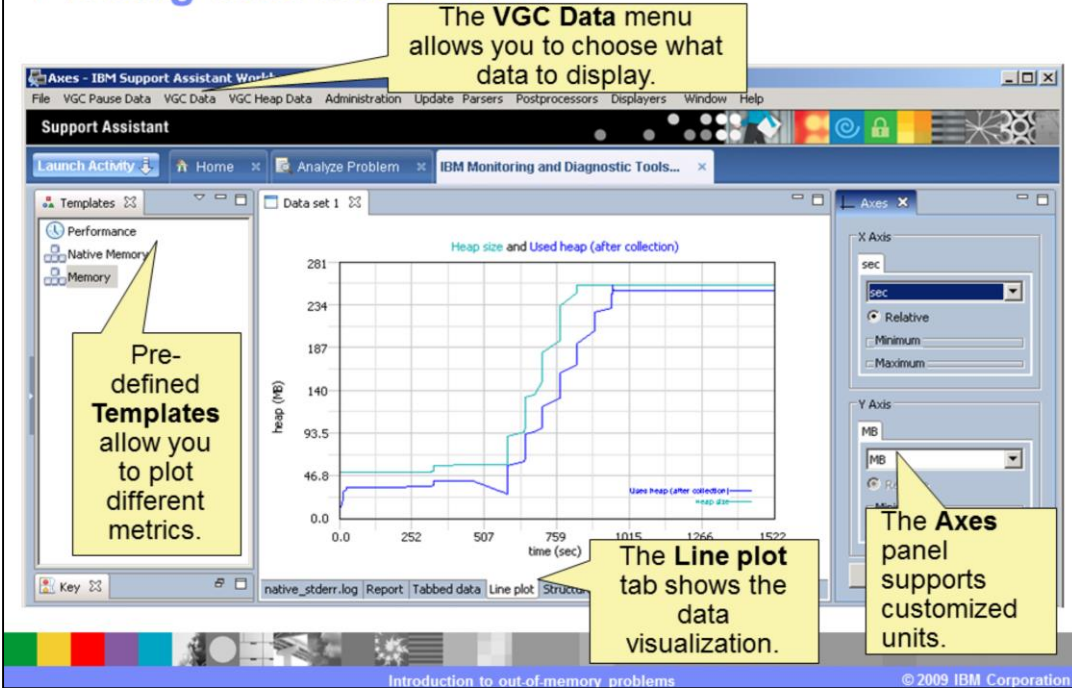
## Garbage Collection and Memory Visualizer overview

- The Garbage Collection and Memory Visualizer (GCMV) is a visualizer for verbose garbage collection output
  - ▶ The tool parses and plots verbose GC output and garbage collection traces (-Xtgc output)
- Launched from the IBM Support Assistant
- The GCMV provides
  - ▶ Raw view of data
  - ▶ Line plots to visualize a variety of GC data characteristics
  - ▶ Tabulated reports with heap occupancy recommendations
  - ▶ View of multiple datasets on a single set of axes
  - ▶ Ability to save data as an image (jpeg) or comma separated file (csv)



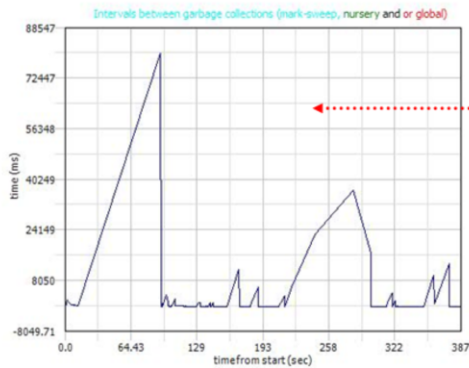
The Garbage Collection and Memory Visualizer tool is included with the IBM Support Assistant and provides reporting and visualizing for a verbose garbage collection log.

## Plotting data with GCMV



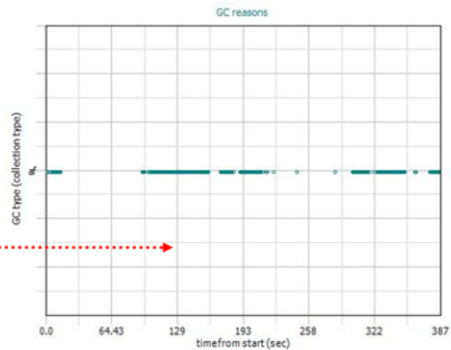
The screenshot displays all the major components of the graphical interface. Pictured are the line plot graphic view, the templates view, and the graph axes configuration view.

## Garbage collection trigger graphs



This graph shows intervals between garbage collection cycles.

Each dot in this graph represents a garbage collection cycle. All of these cycles ran for reason "a" – allocation failure.



The graphs on this slide depict a sample graph showing the intervals between garbage collections on the top. The graph on the bottom of the slide shows a graph displaying the reason for each garbage collection.

# Heap usage and occupancy recommendation

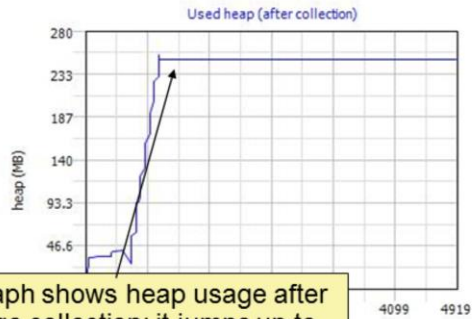
## Tuning recommendation

- ❌ The mean occupancy is 86%. This is high, so you may improve application performance by increasing your heap size. Increasing the heap size should reduce the garbage collection overhead from its current reported level of 0%.
- ⚠️ 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 5120016 bytes.
- +
- +
- i The recommended command line is `-mx830m`.

The summary report shows that the mean heap occupancy is 86% and that the application is using large objects.

### Summary

Mean garbage collection pause (ms)	329
Proportion of time spent in garbage collection pauses (%)	0.94
Number of collections	141
Largest memory request (bytes)	5120016
Proportion of time spent unpaused (%)	99.1
Allocation failure count	140
Forced collection count	0
GC Mode	optthroughput
Mean heap unusable due to fragmentation (MB)	1.78
Concurrent collection count	0
Full collections	0
Total amount tenured (bytes)	26633
Rate of garbage collection	0.381 MB/sec
Mean interval between collections (sec)	35.1



The graph shows heap usage after garbage collection; it jumps up to around 256 MB and stays there.

GCMV provides a summary report after parsing and analyzing the data contained in a verbose garbage collection log. The summary report makes recommendations for JVM command-line arguments that can better optimize the execution of the JVM.

## Comparing verbose GC between different JVMs

- Multiple verbose GC logs can be imported into GCMV and compared

Variant	native_stderr	native_stderr_oom
Mean garbage collection pause (ms)	290	329
Proportion of time spent in garbage collection pauses (%)	0.39	0.94
Number of collections	25	141
Largest memory request (bytes)	131088	5120016
Proportion of time spent unpaused (%)	99.6	99.1
Allocation failure count	25	140
Forced collection count	0	0
GC Mode	optthruput	optthruput
Mean heap unusable due to fragmentation (MB)	2.21	1.78
Concurrent collection count	0	0
Full collections	0	0
Total amount tenured (bytes)	711	26633
Rate of garbage collection	0.268 MB/sec	0.381 MB/sec
Mean interval between collections (sec)	78.3	35.1

Variant	Tuning recommendation
native_stderr	<p><b>w</b> The mean occupancy is 55%. This is high, so you may improve application performance by increasing your heap size. Increasing the heap size should reduce the garbage collection overhead from its current reported level of 0%.</p> <p><b>i</b> The number of collections increased by 100% in the last third of the log compared to the middle third. However, the change in the heap usage was 0%, which suggests that an increase in application activity or fragmentation rather than a memory leak may be the problem. If the workload is not constant then the change in the frequency of collections may be nothing to worry about.</p> <p><b>i</b> The recommended command line is <code>-mx3m -Xmn10.25</code>.</p>
native_stderr_oom	<p><b>w</b> The mean occupancy is 86%. This is high, so you may improve application performance by increasing your heap size. Increasing the heap size should reduce the garbage collection overhead from its current reported level of 0%.</p> <p><b>w</b> 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 5120016 bytes.</p> <p><b>+</b> The memory usage of the application does not indicate any obvious leaks.</p> <p><b>i</b> The recommended command line is <code>-mx30m</code>.</p>

As previously mentioned, GCMV will take two garbage collection logs and compare them, providing feedback on tuning operations. The comparison feature is invaluable when it comes to JVM tuning operations.



## Memory Dump Diagnostic Tool for Java

- Memory Dump Diagnostic Tool for Java
  - ▶ Available in IBM Support Assistant
- Analyzes Java heap dumps
  - ▶ Identifies data structures that are likely causes of memory growth (leaks) and their relationships
  - ▶ Comparison of primary and secondary heap dumps
  - ▶ Provides a graphical display of the Java object tree
- Handles binary and text dump formats



The Memory Dump Diagnostic Tool for Java, or MDD4J, provides good analysis and interpretation of heapdump files. There are two forms of analysis that can be performed. Single heap dump analysis from an `OutOfMemoryError` will help in diagnosing the root cause of the crash. Multi-heapdump analysis from multiple heapdumps taken some time apart can help in determining the cause of a memory leak.



## MDD4J analysis results

- When the analysis completes, there are several views of the heap contents that can be explored.

The screenshot displays the 'Memory Dump Diagnostic for Java' application. The title bar reads 'Memory Dump Diagnostic for Java' and the subtitle is 'Leading edge heap dump analysis for today's memory problems.' The interface includes a menu bar with 'Open', 'View', and 'Help'. Below the menu bar, there are several tabs: 'Results Overview' (selected), 'Suspect View', 'Type View', 'Instance View', and 'Tree View'. The main content area shows an 'Overview' section with the following text: 'Use the following tabs to review the analysis:' followed by a bulleted list:

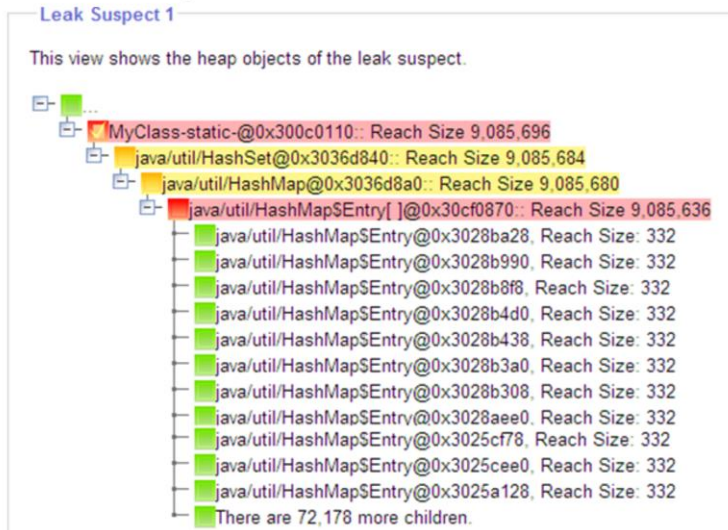
- The 'Suspect View' tab to see tables of:
  - leak suspects(not every heap contains a leak suspect)
  - types with highest number of instances
  - packages with highest number of instances(cumulative for all types from each package)
- The 'Type View' tab to see a table that lists all types in the heap
- The 'Instance View' tab to see a table that lists all object instances in the heap
- The 'Tree View' tab to see each leak suspect, within its ownership chain, in a tree form  
Each tree is navigable. Additional help on using the trees can be found in the 'Help' tab under the 'Trees' sub tab.

The footer of the application window contains the text 'Introduction to out-of-memory problems' and '© 2009 IBM Corporation'.

The Analysis Summary page gives a quick summary of information extracted from the heap dump. This information includes the size of the memory dumped to file, and the number of objects contained in memory. It's important to verify that the heap size dumped to file is correct, otherwise you are looking at a truncated heap dump which will not contain useful information.

## MDD4J: leak suspect tree view

- For each leak suspect you can expand a tree representing the ownership chain of the object.



The Suspects tab is where MDD4J shows you who is potentially leaking the memory.

## MDD4J: type view

- The data type and number of all objects in the heapdump is displayed in the Object Types table.

### Object Types

The table below lists all of the objects by type in the heapdump. Click on the column heading to sort the column for the entire table.

Type Name	Instance count	Array Instance count
unknown	0	106,848
boolean	0	0
byte	0	0
char	0	0
short	0	0
int	0	0
long	0	0
float	0	0
double	0	0
unknown static	0	0
MyClass static	0	0
sun/misc/Launcher\$AppClassLoader	1	0
sun/misc/Launcher\$ExtClassLoader	1	0
java/lang/Thread	5	2
java/lang/ref/Finalizer\$FinalizerThread	1	0

The Explore Context and Contents tab shows the ownership context of selected suspects. Users can select nodes listed on the drop-down list, and graphically explore the ownership context to identify objects consuming significant memory.

## MDD4J: instance view

- A list of all object instances along with their *size*, *reach*, and *number of children* is displayed.

### Object Instances

The table below lists information about all of the objects in the heapdump. Click on the column heading to sort the column for the entire table.

Class name	Object kind	Number of children	Size (bytes)	Reach size (bytes)	Address
unknown[ ]	primitive array	0	327,440	327,424	0x30070200
MyClass static	class	1	24	9,085,696	0x300c0110
java/util/HashSet	object	1	16	9,085,684	0x3036d840
unknown[ ]	primitive array	0	15,688	15,674	0x300c0210
sun/misc/Launcher\$AppClassLoader	object	13	104	565,455	0x300c3f58
java/net/URLClassLoader\$ClassFinder	object	2	24	51	0x300f2f38
sun/misc/Launcher\$ExtClassLoader	object	14	104	563,763	0x300c3fc0
java/lang/ClassLoader\$Finalizer	object	1	16	4	0x300f3050
java/util/Hashtable	object	1	48	107	0x300e7188
java/util/Hashtable	object	1	48	87	0x300e7118
java/security/cert/Certificate[ ]	array	0	16	3	0x3036d8d8
java/util/Vector	object	1	32	63	0x300e70b8
java/util/HashMap	object	1	56	95	0x300e7048
java/util/Vector	object	1	32	63	0x300f3098
java/util/Hashtable	object	1	48	408	0x300f3000

The Browse tab allows you to traverse the object tree looking for significant drops in memory usage. On the left, you can see the details of the highlighted object in the tree on the right. The key information is the Total Reach Size, which tells you how much memory is being used by the highlighted object, and all of the referenced objects below it in the tree. You can identify the leaking object by traversing down the tree until you go from a parent to one of its children, and the Total Reach Size drops significantly.

## Some useful Web addresses

- IBM Virtual Machine for Java Platforms diagnosis documentation:

▶ <http://www.ibm.com/developerworks/java/jdk/diagnosis/>

- IBM JRE and JDK forum:

[http://www.ibm.com/developerworks/forums/dw\\_forum.jsp?forum=367&start=0&thRange=15&cat=10](http://www.ibm.com/developerworks/forums/dw_forum.jsp?forum=367&start=0&thRange=15&cat=10)

- Memory leak detection and analysis in WebSphere Application Server

[http://www.ibm.com/developerworks/websphere/library/techarticles/0606\\_poddar/0606\\_poddar.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0606_poddar/0606_poddar.html)

- Garbage collection policy and tuning article on developerWorks®

▶ <http://www.ibm.com/developerworks/java/library/j-ibmjava3/>



This slide contains several useful Web links that you can use to learn more about garbage collection tuning and memory leak detection.

# Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:  
developerWorks Tivoli WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>

Java, JDK, JNI, JRE, JVM, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

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.

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.

© Copyright International Business Machines Corporation 2009. All rights reserved.

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.

