IBM® WebSphere® Application Server

WAS V7 64-bit performance
Introducing WebSphere Compressed Reference Technology

J. Stan Cox, Aaron Quirk, Derek Inglis, Nikola Grcevski, Piyush Agarwal
stancox@us.ibm.com
ajquirk@us.ibm.com
inglis@ca.ibm.com
nikolag@ca.ibm.com
agarwalp@us.ibm.com

WebSphere Application Server Performance
Research Triangle Park, NC

Contents

Executive Summary

"Compressed references" reduce the size of address references on 64-bit IBM J9 for Java 6, offsetting increased memory footprint and the associated performance loss.

IBM® WebSphere® Application Server (WAS) version 7.0 introduces compressed reference (CR) technology to dramatically improve WAS performance on 64-bit platforms. IBM CR technology for Java™ allows 64-bit WAS deployments to allocate large Java heaps without the memory footprint growth and performance overhead generally incurred with larger, 64-bit address references. Using CR technology, WAS instances can allocate heap sizes up to 28GB with the same physical memory overhead as an equivalent 32-bit deployment.

WAS instances can allocate heap sizes up to 28 GB with the same physical memory overhead as an equivalent 32-bit deployment.

The CR technology provides this improvement by reducing the width of 64-bit Java heap references to 32 bits through an efficient bit shifting algorithm. These 4 byte (32-bit) compressed references are converted to 64-bit values and stored in processor registers in a "just in time manner" at runtime. WAS 64-bit deployments are thus able to reap the benefits of large Java heaps and 64-bit processor extensions while maintaining a comparable 32-bit memory footprint. Note that heap sizes below 4GB may not require any shift operations, since the effective virtual address range only requires 32 bits.

Applications on 64-bit WAS are able to achieve about ~95% of 32-bit performance using the same heap size.

64-bit WAS can provide dramatic performance improvements for applications that take advantage of large heaps. Historically, these performance gains were somewhat limited by the memory footprint overhead of 8-byte Java references. CR technology removes this limitation. For applications that do not require a large heap, the performance of WAS 64-bit with CR technology is generally within 95% of WAS 32-bit, when using the same heap size settings and equivalent hardware configurations. The minimal performance cost is incurred from the overhead of reference compression and decompression.

This paper provides details about CR technology along with the performance characteristics and expectations for WAS V7 32-bit and 64-bit with compressed references technology.

WAS and 64-bit computing

High performance 64-bit platforms are becoming ubiquitous in the enterprise. IBM has been a major player in the 64-bit world for many years, leveraging both AIX and Linux® on Power (1) hardware. Relatively new to the scene are x86-64 platforms from AMD/Intel, which support Windows and Linux in both native 32-bit and 64-bit modes. With the influx of 64-bit platforms, customers are faced with the important decision between 32 or 64 bit applications, and the impact this choice will have on performance.

Background: 64-bit in WAS V6

IBM began offering 64-bit WAS deployments with the release of WAS V6, providing two key advantages for high performance applications designed to utilize 64-bit platforms. These applications benefitted from increased Java™ heap size capacity, as well as Java generated code optimizations to fully leverage 64-bit performance extensions.

Increased heap capacity

Depending on the operating system (OS), WAS 32-bit provides a maximum heap size of roughly 1 to 3 gigabytes (GB). In contrast, 64-bit platforms can theoretically address as much as 16.8 million terabytes (TB), far beyond the current practical limits of physical memory. Since WebSphere/Java does not set an artificial limitation on the size of the heap below the OS dependant 64-bit addressing maximum, 64-bit deployments can easily contain the memory requirements of the most demanding applications.

Applications that take advantage of the large memory addressing capability of 64-bit can gain significant performance advantages. For example, caching vast amounts of data in main memory and thus avoiding latencies in accessing data from slower resources like database or disk can result in dramatic performance gains.

Performance Extensions

In addition to supporting larger Java heaps, 64-bit processors also provide hardware support for double precision (64-bit) mathematical computations and wider 64-bit integer and floating point registers. For example, x86-64 (2) provides eight extra general purpose registers that are only available in 64-bit mode. These features in 64-bit processors can lead to significant performance improvements on computationally intensive applications.

Challenges in V6

Applications capable of leveraging large Java heaps and performance extensions provided by 64-bit WAS V6 experienced unprecedented performance. However, many applications not designed for 64-bit execution experienced better performance on 32-bit platforms, due to the overhead of wider address references in 64-bit environments.

Each 64-bit Java object stored its address reference using 8 bytes, as opposed to 4 bytes in 32-bit deployment. With larger address references, a typical application's memory footprint increased by an average of 60-70% (3). Figure 1 illustrates object growth in a simple class by comparing 32-bit and 64-bit address representation.

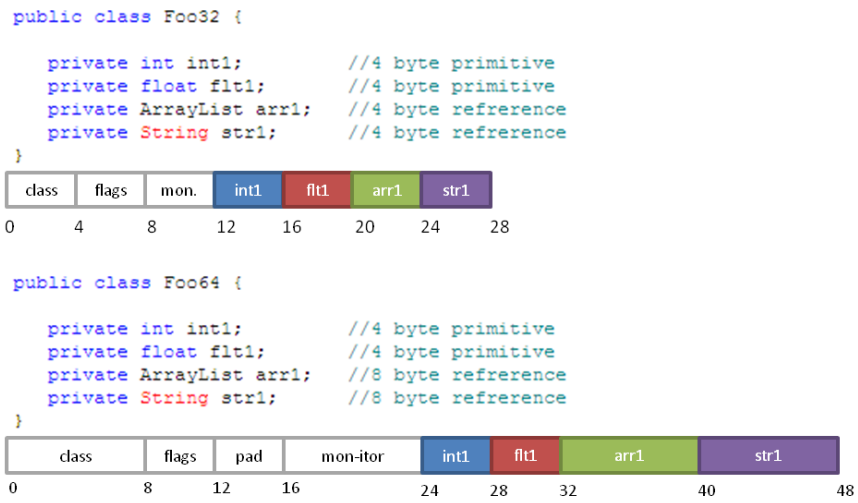


Figure 1 – Java object growth with 64-bit

In this example, `Foo64` (64-bit) occupies 71% more storage than `Foo32` (32-bit) by increasing from 28 to 48 bytes with the same class definition. If this growth represents a 64-bit application, WAS would require 71% more heap capacity than its 32-bit counterpart to maintain equivalent performance. Without increasing capacity, fewer objects can be stored on the heap and garbage collection (GC) must run more frequently in order to conserve space. Not only does 64-bit increase system memory requirements, but can also lead to lower cache and translation lookaside buffer (TLB) utilization due to collisions resulting from the larger 64-bit instruction sizes.

Each of these factors contributed to reduced performance in many applications migrating from 32-bit to 64-bit WAS V6 deployments. Consequently, IBM generally recommended that WAS V6 customers use a 32-bit release unless they were deploying applications specifically intended to leverage 64-bit capabilities (3).

WAS V6 64-bit performance is detailed in *IBM WebSphere Application Server: 64-bit Performance Demystified*, written in September 2007.

Compressed references: 64-bit in WAS V7

There have been dramatic improvements in WAS 64-bit performance since it was first introduced in V6. In addition to many Just-In-Time (JIT) compilation optimizations to improve 64-bit performance, the introduction of compressed reference (CR) technology allows WAS V7 64-bit to provide Java heap sizes up to 28GB, while maintaining the high throughput and smaller physical memory consumption found in 32-bit deployments. CR technology enables this behavior by reducing the width of 64-bit address references to 32 bits through an efficient bit shifting algorithm.

Performance with compressed references

CR technology enables 64-bit applications not designed to utilize large heaps or performance extensions to achieve similar throughput offered by 32-bit deployments. Analysis shows that a 64-bit application without CR yields only 80-85% of 32-bit throughput but with CR yields 90-95%. Depending on application requirements, CR can improve performance up to 20% over standard 64-bit.

The remaining 5-10% performance gap between 64-bit with CR and 32-bit is due to a number of factors:

- Compression and decompression of the references, requiring extra CPU cycles
- Uncompressed, 8-byte memory regions adding to the memory footprint (i.e. Java execution stack, garbage collector)
- The JVM runtime libraries and native Java class library code are still 64-bit
- Larger 64-bit instructions on some platforms resulting in increased instruction cache misses and instruction decode time

On the other hand, using full 8-byte address references for Java objects in standard 64-bit results in significantly greater physical memory consumption and higher cache/TLB miss ratios. These costly

disadvantages far outweigh the CR performance factors listed above, reflected in the performance gap between CR and standard 64-bit.

Compressed reference usage and supported platforms

Compressed references are enabled through the "`-Xcompressedrefs`" command line switch in the IBM J9 for Java 6 JVM. This option is set automatically in supported 64-bit releases of WAS V7, depending on whether the maximum Java heap size is less than 25GB. While CR can address up to 28GB, the 25GB value was chosen as a safe parameter for automatic enablement across disparate platforms. Details surrounding this value have been provided in the CR technical details. To explicitly disable CR, WAS can be started using the "`-Xnocompressedrefs`" option.

WAS V7 is supported on the majority of commercial 64-bit platforms (i.e. POWER, x86-64 from Intel/AMD, Sun SPARC, etc), however in order to use CR technology the WAS release must be packaged with the IBM J9 for Java 6 JVM. Please refer to the WAS V7 information center (4) for detailed and up-to-date software and hardware system requirements.

Planning for 64-bit migration

It's also important to note that 32 and 64-bit WAS installs can co-exist, allowing applications to run in a mixed mode on POWER and x86-64 platforms. This provides a simplified migration path for 32-bit applications running on supported platforms, where they are initially deployed on WAS 32-bit and gradually migrated to a 64-bit version.

Compressed reference technical details

The IBM J9 for Java 6 memory model requires addresses for 64-bit objects to be 8 byte aligned on all supported platforms. While the required alignment is necessary for performance considerations in 64-bit execution, it also creates an interesting side-effect that the three low order bits of every Java object address reference will always be zero.

With this side-effect in mind, it follows that 35 address bits can be stored non-destructively in only 32 bits using simple shift arithmetic. If every 35-bit address is shifted right by 3 when stored and shifted left when loaded, we can safely extend the 32-bit addressable range to 35 bits by taking advantage of the fact that the low order bits of Java heap

references will always be zero. The net effect is that through using compressed address representation, we can theoretically address up to 32 GB of virtual address space using only 32 bits.

When running 64-bit IBM J9 for Java 6 in CR mode, the JIT compiler detects the specified maximum heap size and applies appropriate compression and decompression. Namely, if the maximum heap address used by the Java JVM process is below 4GB, the shift operation on compression and decompression is avoided. Figure 2 provides a graphical representation of the address reference models used in each addressing mode.

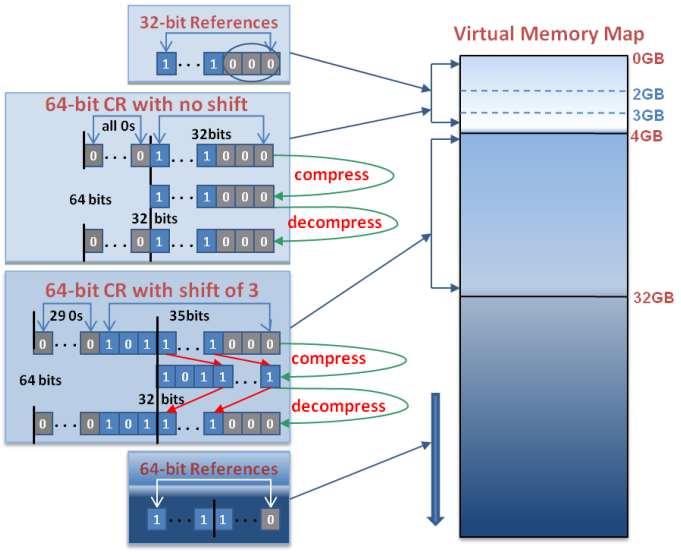


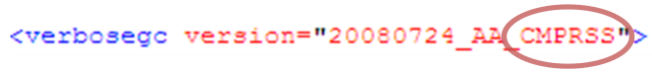
Figure 2 – Address reference models

The memory map to the right of Figure 2 indicates the virtual memory regions addressable by each address reference mode. While the theoretical limit for addressable space in CR mode is 32GB, certain OS memory allocation restrictions reduce the maximum heap size in practical application. The limiting factor is that operating systems reserve one or more regions of memory in each process's virtual address space, indicated by the dashed blue lines on the virtual memory map. Given that Java requires contiguous memory allocation, the heap is not able to achieve the full 32GB theoretical maximum, since the address space is broken up by static memory reservations.

For example, Linux allocates kernel memory at the 3GB mark of the virtual address range while Windows has system specific allocation at

the 2GB mark. In both cases, the Java heap is only able to occupy the virtual memory space following the OS reserved memory up to the 32GB mark. Since compression and decompression logic must be used for address spaces greater than 4GB, CR technology can only be used to satisfy memory requirements less than 28GB.

Considering that memory reservations vary by OS, the automatic CR enablement feature uses 25 GB as a safe limitation for maximum heap size. If necessary, it is possible to extend the 25GB limitation to 28GB by explicitly using the "`-Xcompressedrefs`" switch in the JVM command line arguments. In the case where CR is automatically enabled, verbose garbage collection (verbosegc) logs can be used to verify a 64-bit WAS deployment has successfully initialized the CR feature. When verbosegc is enabled, output is written by default to the native_stderr.log file in the profile's logs directory. Opening this file, an entry similar to the following will indicate "`CMPRSS`" at the end of the version string when CR is enabled:



In 32-bit and standard 64-bit addressing modes, the "`CMPRSS`" string is not present in the verbosegc log.

Performance analysis

It's clear that CR technology provides a distinct performance advantage over the standard 64-bit addressing mode. In order to validate and measure the expected improvement in a real world application, the WAS performance team used the open source DayTrader 1.2 (5) benchmark from the Apache Geronimo (6) project.

DayTrader 1.2 benchmark

DayTrader is an appropriate benchmark for this study because it contains a set of applications not specifically designed for 64-bit deployment. In the past, this type of application suffered significant performance loss because it did not reap the benefits of large Java heaps or 64-bit precision calculations. Results collected from the DayTrader benchmark show that 64-bit deployment with CR technology is now a viable option for these applications. Figure 3 is a general overview of the benchmark's design.

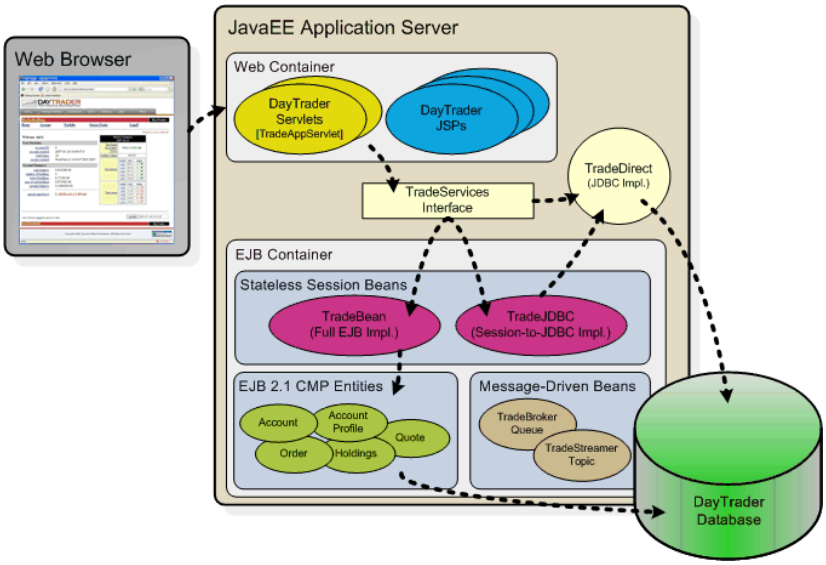


Figure 3 - DayTrader 1.2 architecture

DayTrader is a J2EE enterprise application modeling an online stock brokerage. Web users can login, view and modify their account, check stock quotes, buy and sell shares etc. This benchmark was originally developed by IBM as an end-to-end test suite for Web application server performance. IBM donated the benchmark to open-source and it now is developed and maintained under the Apache Geronimo project. All source code and benchmark artifacts are downloadable under the Apache open-source license.

Physical memory consumption

The following discussion presents data collected from DayTrader 1.2 over various operating systems and architectures supported by WAS V7. In order to understand the performance improvements enabled by CR, it's important to understand the differences between physical memory usage in 32-bit, 64-bit and 64-bit with CR. Figure 4 compares the effective heap consumption for each type of WAS deployment on several operating systems, each configured with a 1GB Java heap.

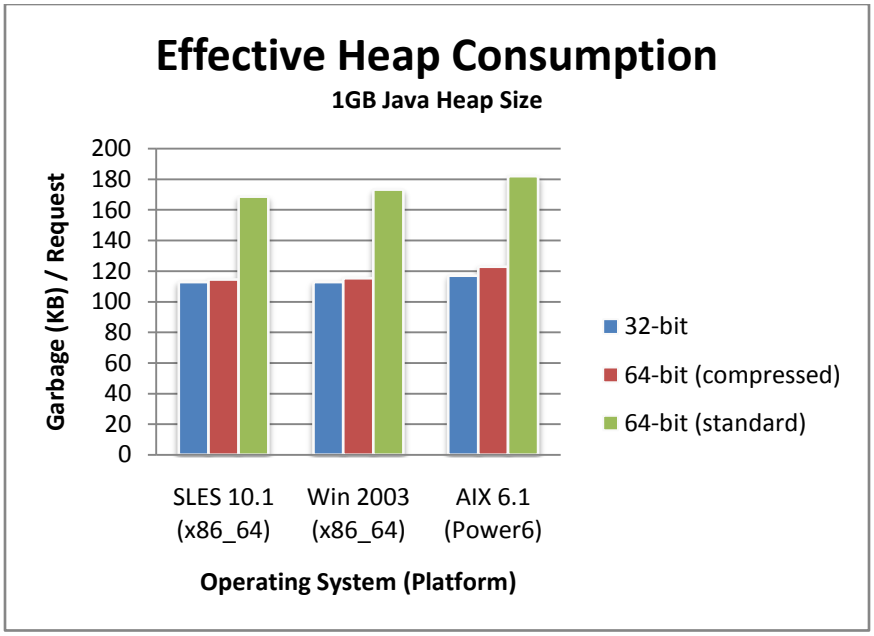


Figure 4 - Memory consumption with 1GB Java heap

Effective heap consumption is expressed as the average kilobytes (KB) of garbage generated per request. This metric approximates the effective Java heap growth introduced by 64-bit, making the assumption that actual footprint growth for all objects is equal to the growth of objects freed during garbage collection (GC). While the memory freed during GC represents short lived objects, persistent heap data experiences similar growth conditions.

Figure 4 shows that CR heap utilization is nearly equivalent to 32-bit, while usage increases by ~50% in standard 64-bit addressing modes across all platforms. Each value is computed by dividing the amount of heap memory freed over a discrete time span, by the number of requests completed during the time span. This expression simplifies to:

$$\frac{\sum KB\ Freed}{Request}$$

Verbose GC logging must be enabled to acquire the information required for this analysis, using the "`-Xverbose:gc`" JVM switch. Parsing the XML output from the verbosegc log provides detailed entries for each garbage collection. This data includes timestamp, bytes collected, post GC heap occupation, etc.

Throughput comparison

Spikes in memory growth are directly related to the performance regressions experienced by many applications migrating to 64-bit WAS deployments. The connection between performance and memory usage is demonstrated in the following chart, which contains corresponding throughput data from the scenarios in Figure 4. In order to account for hardware differences, values have been normalized as percentages of 32-bit throughput for each platform.

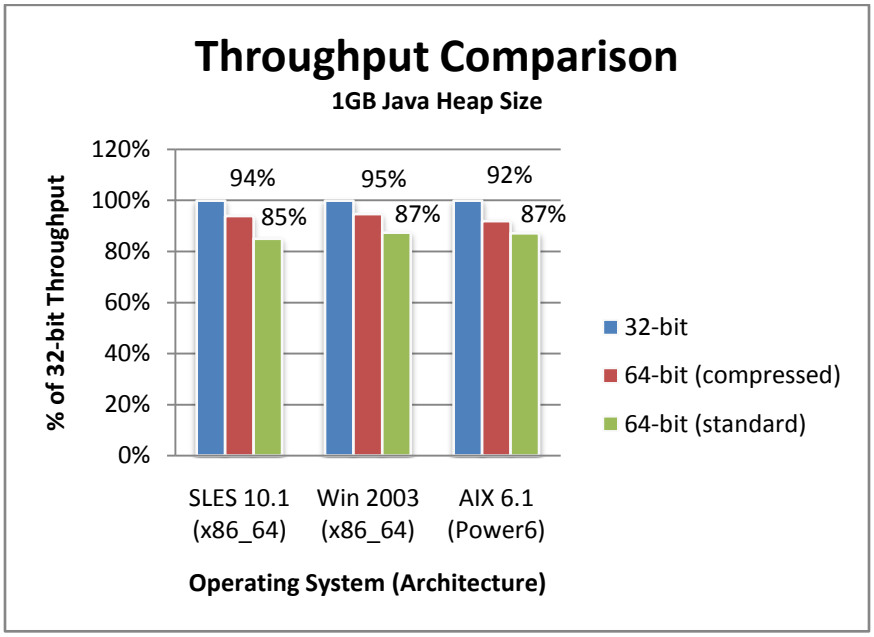


Figure 5 - Throughput with 1GB heap

The performance advantage provided by the CR deployment is clear. Assuming 32-bit throughput as the baseline for each platform, CR deployments are able to achieve ~95% of the available performance, compared to ~85% for standard 64-bit deployment. Recall that the 5% performance delta between 32-bit and CR is the result of compression / decompression operations, uncompressed regions in the Java execution stack, external calls to 64-bit user libraries, etc..

Detailed discussion for Linux on x86

Thus far, data has been limited to WAS deployments with 1GB Java heaps, however it's also very useful to discuss the performance characteristics of WAS deployments as the heap sizes grow. Using Novell SUSE Linux Enterprise Server (SLES) 10.1 on x86 hardware, memory footprint and throughput are explored in detail with varying

Java heap configurations. Figure 6 shows memory usage with heaps configured up to 12GB, according to the hardware limitations of the system under test.

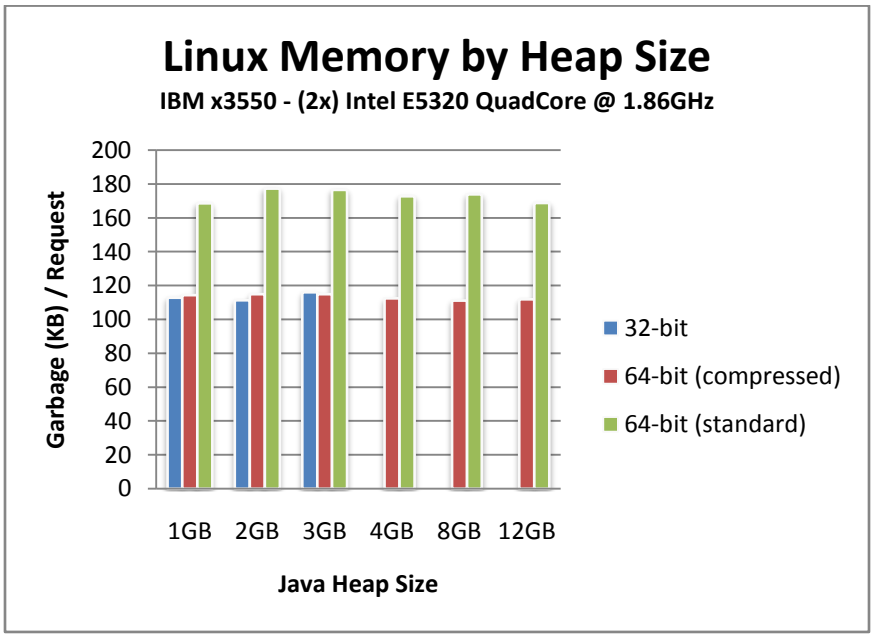


Figure 6 - Linux memory footprint by heap size

Since SLES uses a Linux kernel, the Java heap is limited to 3GB in 32-bit deployments. Within this range, effective heap consumption is nearly equivalent for 32-bit and 64-bit CR deployments. Looking further along the axis, we can determine that relative memory usage is consistent between the two 64-bit deployments regardless of heap size. Even as heap size approaches 12 GB, 64-bit CR deployments maintain their advantage in physical memory consumption over the standard 64-bit addressing mode.

Since memory consumption and throughput are directly related, CR is also able to maintain its performance advantage over 64-bit as the heap size grows. Figure 7 shows the corresponding throughput measurements for the heap configurations above.

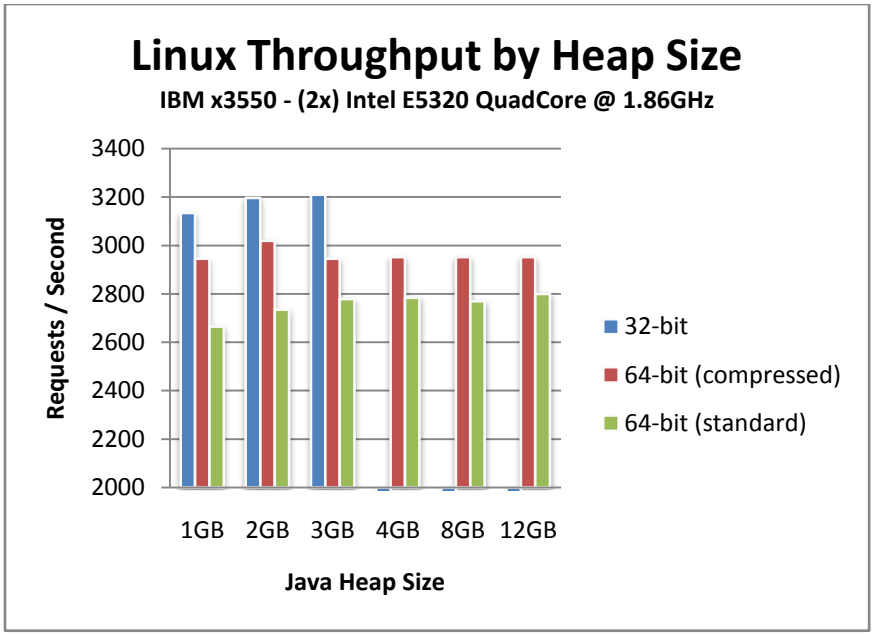


Figure 7 - Linux throughput by heap size

There are two important observations to take away from figure 7. First, the throughput for WAS deployments using CR is significantly faster than standard 64-bit for all heap sizes. Secondly, CR experiences a slight drop in relative performance between 2GB and 3GB, but remains constant from 3GB forward.

The slight performance shift (~2.5%) at 3GB occurs when throughput is reduced from 3020 requests/second to 2946 requests/second. This shift is a consequence of the Linux kernel's static memory reservation in the JVM's virtual address space. Recall from the CR details that since IBM J9 for Java 6 requires contiguous heap allocation, the effective heap's start offset must be moved past the kernel's reserved memory. Once the offset has been adjusted, the heap has a wide area of open memory to satisfy its contiguous requirement using the address range above the 3GB mark.

However, when the effective Java heap memory start offset is pushed above the 32-bit addressable memory range, the JIT compiler has to perform bit shift arithmetic for compressing and decompressing the 64-bit object references. The extra instructions for address compression and decompression cause increase in instruction path-length, which results in the slight overhead observed in Figure 7.

Each OS comes with its own set of memory reservations, which must be accounted for by the CR algorithm. Despite all limitations, the CR

technology is able to provide significantly faster 64-bit performance at each heap size, even taking into account OS specific differences.

Conclusions/Summary

Since the introduction of 64-bit technology in WAS V6, Java 5 applications were provided the ability to run with heaps much larger than the 3GB effective limit found on 32-bit platforms. The IBM Just-In-Time compilation technology for Java transparently leveraged the 64-bit performance extensions and generated optimized code.

WAS V7 introduces "compressed references" technology with IBM J9 for Java 6, which allows the JVM to offset the performance and memory footprint growth previously experienced by many applications on 64-bit platforms. The CR technology allows applications deployed on 64-bit WAS to take full advantage of all 64-bit features of the processor, such as native 64-bit arithmetic, additional CPU registers, etc. At the same time it allows the application to use heaps as large as 28GB, overcoming the 4GB limit imposed by 32-bit architectures.

Minimizing the performance gap between 32-bit and 64-bit WAS deployments, compressed reference technology provides customers with a viable single install solution on 64-bit platforms for both applications that leverage and those that do not leverage 64-bit address extensions.

References

1. **Utsler, Jim.** Linux on Power. *IBM.* [Online] IBM. http://www.ibm.com/servers/eserver/linux/power/powerarticle.pdf.
2. **Christian Zdebel, Simon Solotko.** the AMD64 Computing Platform. *AMD.* [Online] http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/30172C.pdf.
3. **J. Stan Cox, Piyush Agarwal, Nikola Grcevski, Hong Hua.** IBM WebSphere Application Server 64-bit Performance Demystified. [Online] IBM. ftp://ftp.software.ibm.com/software/webserver/appserv/was/64bitPerf.pdf.
4. **IBM Corporation.** WebSphere Application Server. *IBM Information Center.* [Online] IBM. http://www.ibm.com/software/webservers/appserv/was/library/.
5. **Apache Geronimo Project.** DayTrader. *Apache Geronimo v2.0 Documentation.* [Online] http://cwiki.apache.org/GMOxDOC20/daytrader.html.
6. —. Welcome to Apache Geronimo. *Apache Geronimo.* [Online] http://geronimo.apache.org/.

Additional Resources

Java Diagnostics Guide 6
http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp

Porting Guide – Moving Java Applications to 64-bit Systems
http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/64bitporting/64BitJavaPortingGuide.pdf

IBM, AIX, DB2, POWER Architecture, POWER3, POWER4, POWER4+, POWER5, POWER5+, POWER6, WebSphere and xSeries are trademarks of International Business Machines Corporation in the United States, other countries, or both

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

Microsoft, Windows, Windows 2003 and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Intel Xeon are registered trademarks or trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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