Deterministic Garbage Collection: Unleash the Power of Java with Oracle JRockit Real Time

An Oracle White Paper
August 2008

ORACLE

Deterministic Garbage Collection: Unleash the Power of Java with Oracle JRockit Real Time

INTRODUCTION

The Java virtual machine (JVM) is the cornerstone of the Java platform—the technology responsible for Java's hardware and operating system independence. Although it is an essential component of enterprise Java applications, the hardworking JVM has remained largely anonymous, with little distinction made between the many versions on the market. But all JVMs are not created equal.

Oracle JRockit Real Time incorporates a number of unique, industry leading technological advances that push Java into the domain of real-time systems and the realm of application debugging and memory leak control. One of its most important advances is *deterministic garbage collection:* an automatic memory management technique that minimizes transaction latency.

The deterministic garbage collection capabilities in Oracle JRockit Real Time enable the use of Java technology in markets where it was previously impractical, including financial services and telecommunications.

Java is a "garbage-collected" language; in other words, objects that are no longer referenced must be periodically cleared out so that processing can continue. Automated, traditional garbage collection processes are highly unpredictable, placing high demand on the JVM and causing erratic pause-and-response times. The deterministic garbage collection capabilities in Oracle JRockit Real Time smooth these performance spikes to deliver faster, more-reliable performance that enables the use of Java technology in markets where it was previously impractical.

UNDERSTANDING GARBAGE COLLECTION

Efficient memory use increases application performance and stability. Garbage collection as a form of memory management greatly influences Java application performance. In the process, it performs two basic activities:

- Determines which objects in memory are or are not being used
- Reclaims the memory being consumed by inactive, discarded objects

Improper handling of garbage collection inhibits application execution and seriously detracts from system performance and reliability. Some applications require the highest-possible application throughput and can tolerate periodic garbage collection pauses. Others cannot; they demand consistency, sacrificing some amount of throughput to minimize pause times.

TRADITIONAL GARBAGE COLLECTION

Java is certainly not the first programming language to rely on garbage collection, but it is probably the most widely used. The benefits are clear: increased reliability, decoupling of memory management from class interface design, and less developer time spent chasing memory management errors. This results in a faster time to market because less time is spent on debugging and development.

Garbage collection is the process of reclaiming unused memory to increase application performance. Although beneficial overall, traditional garbage collection techniques have unpredictable performance impacts.

However, garbage collection is not without its costs: unpredictable performance impacts, pauses, and configuration complexity, among others. Figure 1 illustrates what happens when the garbage collection method used by the JVM does not have the capability to dynamically set pause times. The top line is of key concern. This line shows the long pause times of the garbage collector. Each time the collector pauses to clean out the unused objects (the garbage), the application experiences a delay in returning a rapid response. When performed multiple times in the course of normal operation, this delay affects both application performance and service-level agreements.

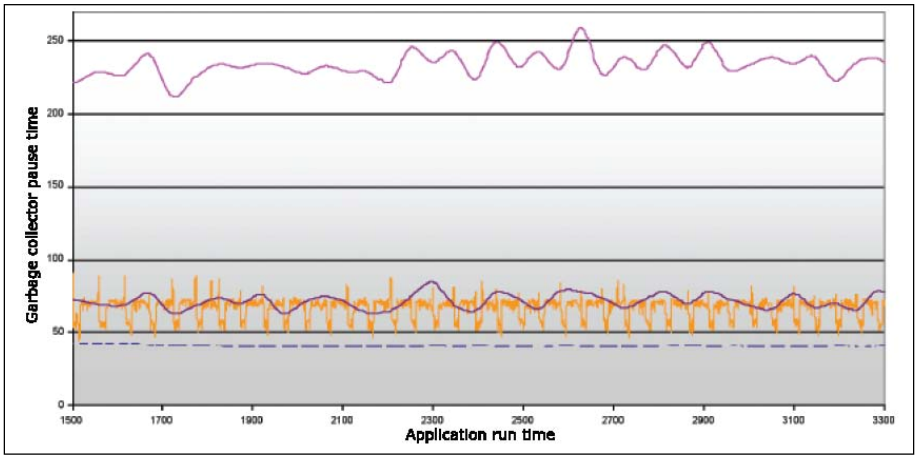

Figure 1: Application using typical garbage collection technology (measured in milliseconds)

One of the more commonly used garbage collection methods is parallel garbage collection, as shown in Figure 2. In a parallel garbage collection strategy, the pause times are less frequent, but involve longer periods of time.

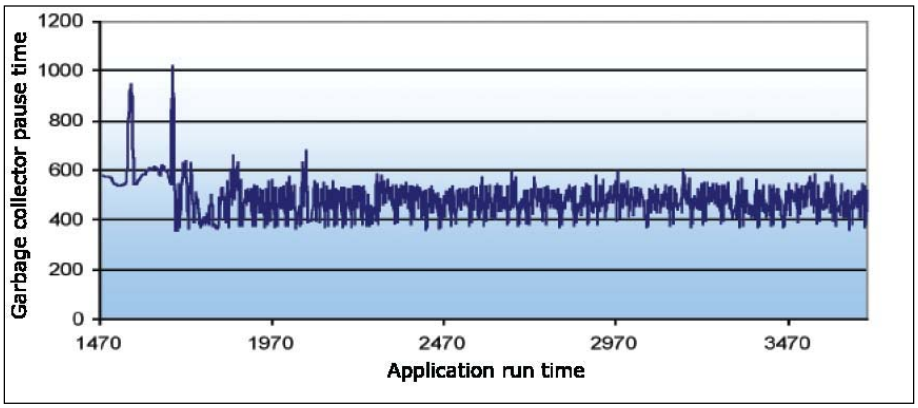

Figure 2: Application using parallel garbage collection technology (measured in milliseconds)

DETERMINISTIC GARBAGE COLLECTION

Ideally, garbage collection implementation would be completely invisible: there would be no collection pauses and no CPU time lost to garbage collection. Unfortunately, there are no ideal garbage collectors, but the deterministic garbage collection functionality in Oracle JRockit Real Time represents a significant improvement over traditional methods.

Deterministic garbage collection is the ability to specify and maintain a maximum pause time for the memory system with a high level of confidence. It is designed to deliver short, predictable pause times with minimal manual tuning.

Deterministic garbage collection is the ability to specify and maintain a maximum pause time for the memory system with a high level of confidence. Designed to deliver short, predictable pause times with minimal manual tuning, it helps in situations where there is a continuous querying of events or where it is necessary to find correlations over streams and time periods in real time or near real time.

The memory management system of Oracle JRockit Real Time offers an array of garbage collection strategies tailored for different applications and environments. It also offers an adaptive mode that uses runtime analysis to dynamically adjust the garbage collection strategy and tuning parameters to best fit the performance and behavioral requirements of the application.

Figure 3 shows how deterministic garbage collection can smooth out the spikes in collection and pause times. In this example, the maximum pause time is set at 50 milliseconds and all the garbage collection activity takes place beneath that threshold. By using many short garbage collection pauses, the duration of each garbage collection period is kept to a minimum, thereby improving performance.

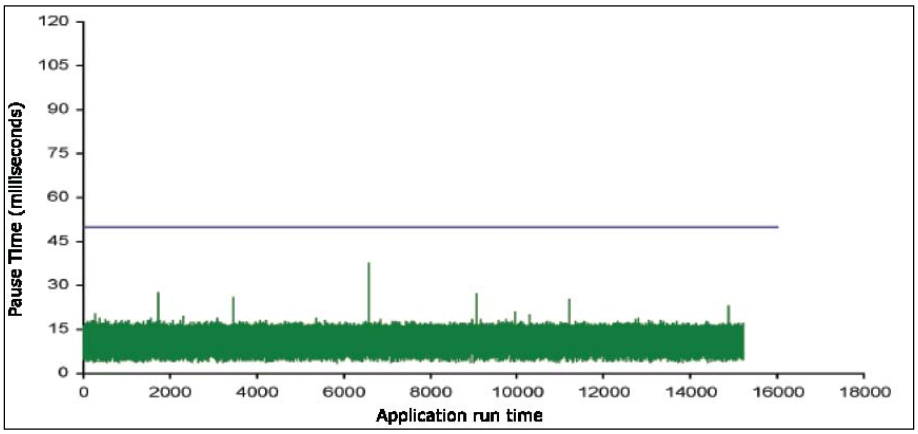

Figure 3: Application using deterministic garbage collection

Figure 4 compares the pause times between deterministic garbage collection and parallel garbage collection. The long pause times for the parallel garbage collector can be fatal for applications that are sensitive to delays, such as real-time trading applications. With the deterministic garbage collector, pause times are guaranteed to be kept to a minimum, even when running applications with gigabyte-size heaps. The short, frequent pauses in the deterministic garbage collector ensure that an application will not experience time-outs like those of the parallel garbage collection method, which occur when the JVM using parallel collection pauses while purging unused items from memory.

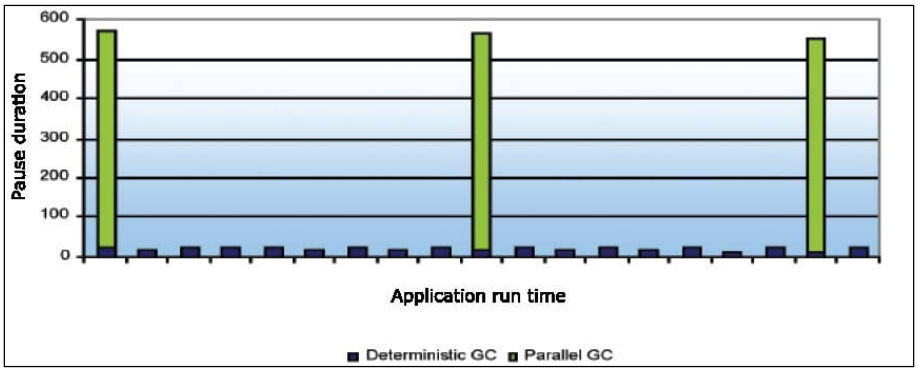

Figure 4: Comparison of deterministic garbage collection and parallel garbage collection pauses

REAL-TIME ENTERPRISE JAVA: THE PROMISE OF DETERMINISTIC GARBAGE COLLECTION

Many enterprises have noticed productivity gains in development and maintenance when switching to Java and want to expand its use to the greatest extent possible. However, several markets that rely upon high-speed, high-volume applications have been unable to use Java due to unpredictable garbage collection, including

- **Financial services.** Response times of less than 20 milliseconds are typical for trade-processing applications because every millisecond of downtime translates into lost revenue. Decreasing the amount of downtime maximizes the number of trades that can take place, and because the trade execution

takes place sooner, it also enables the maximum number of trades to be completed.

- **Telecommunications.** Responses times between 50 and 100 milliseconds are typical for telecom infrastructures. The rapid response times maximize the number of calls that can be set up in a given time period. Delayed response times result in more dropped calls and a higher frequency of busy signals.
- **Gaming.** Betting and betting exchange transactions are inhibited by long system response times as a result of dropped frames when rendering to the screen.

With the availability of deterministic garbage collection, customers can now consider the multiple benefits of moving expensive C/C++ legacy systems to less expensive, more flexible Java-based systems.

Due to these demanding response times, Java has not been a viable option because these time metrics could not be guaranteed. As a result these sectors are dominated by applications written in C/C++. With the availability of deterministic garbage collection, customers can now consider the multiple benefits of moving expensive legacy systems to less expensive, more flexible Java-based systems.

Figures 5 and 6 show the performance gains that deterministic garbage collection can bring to real-time systems. These examples illustrate the response times for a financial services trading application using different platforms and methods of garbage collection.

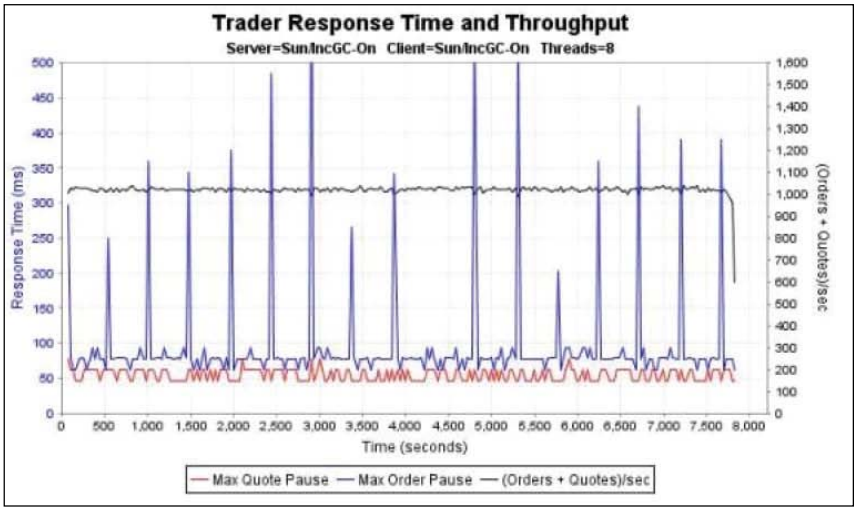

Figure 5: Trader response time and throughput using Sun JVM; shows unpredictable spikes that surpass the maximum order pause on several occasions

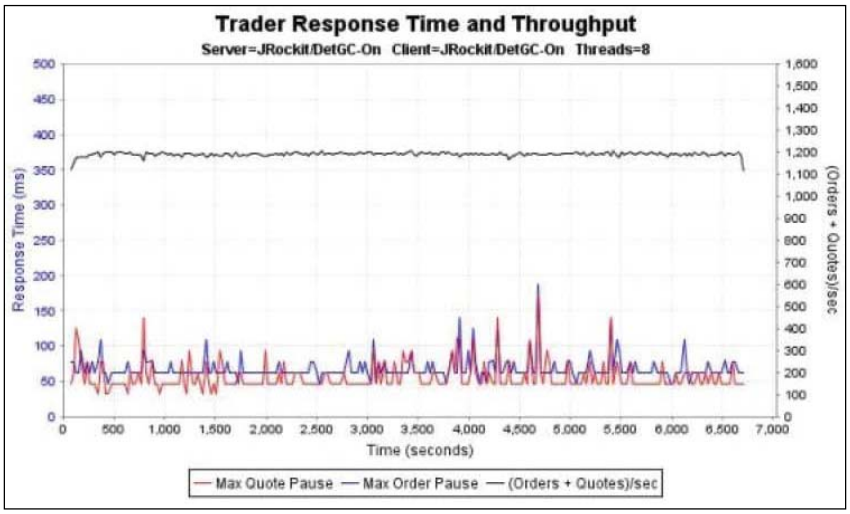

Figure 6: Trader response time and throughput using Oracle JRockit Real Time and deterministic garbage collection; no pause exceeds the threshold and most under 100 milliseconds

PROGRESSIVE OPTIMIZATION

Oracle JRockit Real Time provides continuous performance improvement in real time—from initial deployment through the life of the application.

Progressive optimization complements adaptive memory capabilities to further enhance performance. Oracle JRockit Real Time provides continuous performance improvement in real time—from initial deployment through the life of the application. It automatically adapts its behavior to the operating conditions of the application and the underlying environment to deliver optimal performance, scalability, and reliability. The solution compiles each method the first time it encounters it, generating machine code with platform-specific optimizations. For more-aggressive optimization, it then monitors an application as it executes and identifies the methods on which it spends the most time. This approach eliminates many performance bottlenecks early on, and continues to do so throughout the life of the application.

PLATFORM UBIQUITY

Oracle JRockit Real Time continues to demonstrate superior application performance and price-to-performance ratios as measured by a series of industry standard benchmarks.[1] Oracle JRockit Real Time has set numerous SPECjbb2000 and SPECjbb2005 performance records[2] on Intel and AMD platforms. It is optimized for performance on systems built from industry standard Intel- and AMD-based servers, from 32- and 64-bit Intel Xeon and AMD Opteron processor-based systems to servers using the 64-bit Intel Itanium 2 processors. It is also effective on Sun SPARC systems.

[1] For various benchmark results, please refer to www.spec.org.
[2] For SPECjbb benchmark results, please refer to www.spec.org/jbb2000/results/jbb2000.html and www.spec.org/jbb2005/results/jbb2005.html.

SPARC customers can now go from the top of the application stack into the operating system using Oracle-supported Java technology. With Oracle JRockit Real Time on Solaris SPARC, Sun users have a choice of JVMs optimized for their needs and can take advantage of real-time deterministic garbage collection without additional hardware investments.

When applications are deployed on these Intel, AMD, or Sun processor-based platforms with Oracle JRockit Real Time, the Java programming language becomes the ultimate deployment platform for large-scale, server-side, enterprise-class applications.

When applications are deployed on any of these processor-based platforms with Oracle JRockit Real Time, the Java programming language becomes the ultimate deployment platform for large-scale, server-side, enterprise-class applications—making it cost effective for organizations to scale enterprise applications and remain competitive.

CONCLUSION

Java's promise of "write once, run anywhere" is made possible through industrywide adoption of JVMs that run on a variety of operating systems and hardware chipsets. Unfortunately, not all JVMs are created equal, and Java performance on various platforms has suffered from the limitations of previous JVM implementations. These limitations have restricted the use of Java in certain sectors due primarily to memory management and garbage collection processes that were not designed to minimize transaction latency.

With its deterministic garbage collection capabilities and other enhanced features, including progressive optimization, Oracle JRockit Real Time enables enterprises that depend on low-latency, real-time applications to capture the benefits of Java.

Deterministic Garbage Collection: Unleash the Power of Java with Oracle JRockit Real Time
August 2008

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com