



INTEGRATE



BUILD



PORTAL



# BEA WEBLOGIC JROCKIT™: JAVA FOR THE ENTERPRISE

# COPYRIGHT

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.  
December 2003

## RESTRICTED RIGHTS LEGEND

This document may not, in whole or in part, be photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc. Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems, Inc.

## TRADEMARKS

BEA, Tuxedo, and WebLogic are registered trademarks and BEA WebLogic Enterprise Platform, BEA WebLogic Server, BEA WebLogic Integration, BEA WebLogic Portal, BEA WebLogic Platform, BEA WebLogic Express, BEA WebLogic Workshop, BEA Liquid Data for WebLogic, BEA WebLogic JRockit, and BEA WebLogic Java Adapter for Mainframe are trademarks of BEA Systems, Inc.

Intel® Xeon™ and Intel® Itanium® 2 are trademarks of Intel Corporation.

J2SE, J2EE, and Enterprise JavaBeans are trademarks of Sun Microsystems, Inc.

All other company and product names may be the subject of intellectual property rights reserved by third parties.

CWP0671E1203-1A

# CONTENTS

Introduction . . . . .	1
BEA WebLogic JRockit: Performance, Manageability, Simplicity . . . . .	1
Architecture of the BEA WebLogic JRockit JVM . . . . .	3
The Simple Path to Enterprise Application Development and Deployment . . . . .	4
Boosting Developer Productivity . . . . .	4
Runtime Efficiency Through Progressive Optimization . . . . .	5
“Zero-Overhead” Monitoring . . . . .	5
Dynamic Optimizing Code Generator . . . . .	6
Efficient Thread Management . . . . .	7
Adaptive Memory Management for Scalability and Reliability . . . . .	8
Adaptive Heap Management . . . . .	8
Adaptive Garbage Collection . . . . .	9
Optimizing Memory Management for Client and Server Environments . . . . .	12
Managing for Enterprise Performance, Scalability, and Reliability . . . . .	14
BEA WebLogic JRockit Monitoring and Management APIs . . . . .	14
BEA WebLogic JRockit Management Console . . . . .	15
BEA WebLogic JRockit Runtime Analyzer . . . . .	16
BEA WebLogic JRockit Performance and Scalability . . . . .	17
Bottom Line: Simply Superior Enterprise Java . . . . .	18
Availability . . . . .	19
About BEA . . . . .	19

# INTRODUCTION

Today's developers have seen an explosion of large-scale system development beyond the confines of the back-office and mainframe systems of three decades ago. The Java programming language has been a key factor in the creation of large-scale enterprise systems. It has evolved from a "write once, run anywhere" client-side language to the language of choice for large-scale enterprise applications. Java "building blocks" have helped to reduce application development time and complexity.

But now growing numbers of users and increasingly complex business requirements are pushing Java applications to their limits, and companies often have to spend large amounts of development time and resources achieving and maintaining performance, scalability, and reliability in their enterprise Java applications. Adding to these costs is the fact that many Java Virtual Machines (JVMs) are optimized by hardware vendors for their own proprietary architectures, so basic performance can come at a high price in hardware investments. To satisfy user, business, and financial requirements, developers need a simple, cost-effective way to ensure application performance, reliability, and scalability on low-cost, standards-based platforms.

## BEA WEBLOGIC JROCKIT: PERFORMANCE, MANAGEABILITY, SIMPLICITY

BEA WebLogic JRockit™ is the only Java VM designed specifically to simplify the task of developing and managing Java applications for large-scale enterprise-wide environments. With BEA WebLogic JRockit, developers do not need to know JVM internals to create performant, scalable, reliable applications. BEA WebLogic JRockit speeds application development through fast startup performance. It provides out-of-the-box application performance and scalability through *progressive optimization* features that enable the JVM to automatically deliver the best possible application performance without requiring a lot of manual configuration or tuning. Unique manageability features give developers the real-time visibility and control to ensure top application performance and health while delivering industrial-strength system stability and reliability under heavy user and transaction loads. And BEA WebLogic JRockit is the only JVM designed for top performance with open standards-based Intel platforms, which allows it to deliver the best price/performance and lowest total cost of ownership (TCO) for enterprise Java applications.

## BEA WEBLOGIC JROCKIT BOOSTS PERFORMANCE, RELIABILITY, AND DEVELOPER PRODUCTIVITY

### Simplified Development and Top Performance

- Development Optimization
- Progressive Optimization
- Thread Management
- Adaptive Memory Management

BEA WebLogic JRockit provides continuous, automatic performance improvement and scalability from development through initial deployment and day-to-day operations.

Minimizes application start-up time to speed up iterative development and testing.

Zero-overhead monitoring identifies areas for potential performance improvement, and dynamic code optimization continually improves runtime performance.

Specialized locking techniques improve performance for multi-threaded enterprise applications.

Adaptive memory management adjusts heap sizes and garbage collection techniques to meet changing application requirements.

### Unique Manageability

- Management APIs
- JRockit Management Console
- JRockit Runtime Analyzer

Gives developers and system administrators the visibility and control to ensure industrial-strength performance and reliability for enterprise applications.

Allows applications and third-party tools to manage JVM and application behavior at runtime without having to instrument bytecode.

Gives developers and system managers the visibility to monitor application behavior and to identify and resolve issues before they affect reliability or performance.

Provides detailed runtime information for problem diagnosis and performance improvement, without compromising run-time performance.

### Industry-leading Performance on Standards-based Intel Architecture

Proven superior performance on 32-bit and 64-bit Intel architectures lowers TCO of enterprise systems and offers greater flexibility in OS and hardware choices.

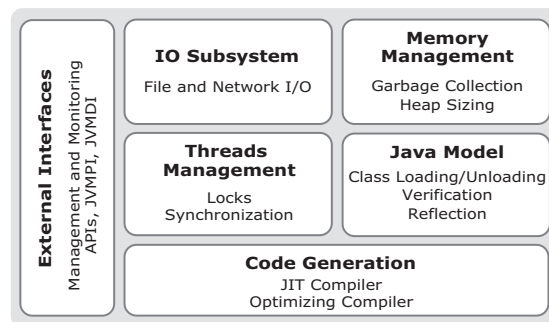
# ARCHITECTURE OF THE BEA WEBLOGIC JROCKIT JVM

Every subsystem of the BEA WebLogic JRockit JVM is designed to deliver superior performance, simplicity, and manageability for applications in large-scale, enterprise-wide deployments.

- The code-generation subsystem performs progressive optimization throughout the life of the application.
- Thread management is optimized to minimize synchronization between threads.
- Memory management is designed for efficient memory usage and application throughput throughout a running application.
- The Java model maintains an up-to-date view of system metadata. The BEA WebLogic JRockit JVM uses highly optimized algorithms to efficiently manage classes, fields and methods, as well as classloading and string handling for Java applications. The Java model also does a number of optimizations to ensure the efficiency of accessing various instance members using Java reflection.
- The I/O subsystem is well tuned and optimized for common Java file and network I/O activities.
- External management and monitoring APIs help developers fine-tune performance and ensure system health, and they offer extensibility through the integration of third-party tools.

FIGURE 1.

BEA WebLogic JRockit Architecture



# THE SIMPLE PATH TO ENTERPRISE APPLICATION DEVELOPMENT AND DEPLOYMENT

Developers can spend a lot of time and effort trying to understand and optimize runtime behavior of enterprise applications, but the JVM ultimately determines the runtime performance and has the ability to affect the application's behavior in real time. It can also give the developer insights and optimization choices that would not be apparent with traditional profiling tools.

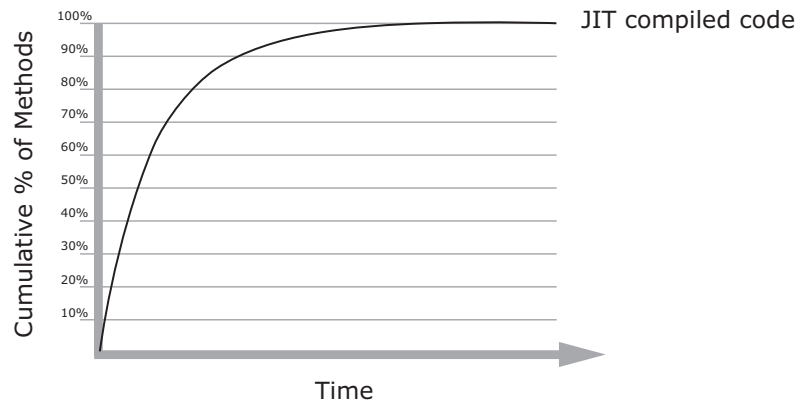
The BEA WebLogic JRockit JVM eliminates much of the time and effort and many of the stumbling blocks that developers have faced in achieving Java application performance. BEA WebLogic JRockit is the only JVM designed to allow developers to realize optimal application performance and scalability without tuning a single configuration parameter. It does this through *progressive optimization* and *adaptive memory management*, approaches whereby the JVM automatically adapts its own behavior based on the operating conditions of the application itself and the underlying environment — client versus server systems, concurrent users and memory requirements, and variations in the system resources such as available memory and CPUs — to deliver optimal performance, scalability, and reliability throughout the life of the application.

## BOOSTING DEVELOPER PRODUCTIVITY

Application development involves iterative redeployment of an application as developers create, debug, and improve their code. Aside from development tools, one of the most significant factors that can contribute to developer productivity is the amount of time it takes to build, deploy, and start the application.

Some JVMs minimize start-up time by interpreting Java code at runtime, even though compiled code executes much faster. BEA WebLogic JRockit balances startup time with runtime performance by compiling each method “just in time” (JIT) the first time it is used and then caching that compiled method onto the disk for subsequent automatic reuse. Thereafter, subsequent deployments or restarts of the application can automatically retrieve the compiled code from disk, shortening start-up time and boosting overall performance. Achieving optimal application performance is typically a large part of Java development efforts. So by helping to deliver both fast runtime performance through JIT compilation of all methods and fast startup time through code-caching of compiled methods, BEA WebLogic JRockit dramatically improves overall developer productivity. Code-caching has been introduced as an experimental feature and will continue to be enhanced. As shown in Figure 2, the BEA WebLogic JRockit JVM compiles and caches each method the first time it is encountered.

FIGURE 2.  
JIT Compilation of  
Methods Over Time



## RUNTIME EFFICIENCY THROUGH PROGRESSIVE OPTIMIZATION

The BEA WebLogic JRockit JVM combines the speed of compiled code with the benefits of adaptive performance technology through *progressive optimization*, a process of continual performance improvement from initial deployment through the life of the application. The JVM compiles each method the first time it is encountered, generating machine code with platform-specific optimizations such as a special fast register allocation unique to the IA64 architecture. The JVM then monitors the application as it executes and identifies the methods where the application spends the most time for more aggressive optimization. This approach eliminates many performance bottlenecks early in the life of a running application.

### “ZERO-OVERHEAD” MONITORING

The BEA WebLogic JRockit dynamic runtime environment uses a sophisticated, low-overhead sampling-based technique to identify areas for optimization. A “sampler thread” wakes at periodic intervals and checks the status of several application threads. It identifies what each thread is executing and notes some execution history. The information is tracked, and methods where the application spends most of its time are earmarked for optimization. Overhead for monitoring is typically only 1-2%.

Early in deployment of an application, BEA WebLogic JRockit monitors execution to identify areas for code optimization. As the runtime performance increases and stabilizes, the JVM monitors less and less frequently, further minimizing overhead and, thereby, maximizing performance.



If methods are added or changed or if changing application usage causes the application to spend more time in different methods, BEA WebLogic JRockit optimizes those methods further and monitors again until performance stabilizes.

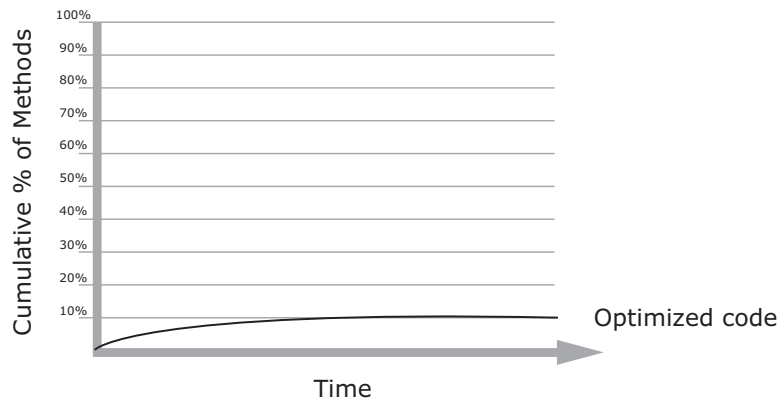
## DYNAMIC OPTIMIZING CODE GENERATOR

BEA WebLogic JRockit's dynamic optimizing compiler uses a number of techniques to increase the performance of frequently used methods. Some Java developers believe that JIT compilation cannot optimize Java effectively because of the “openness” of Java features such as dynamic type-casting and virtual method invocations. However, JRockit's progressive optimization overcomes this issue. Methods are JIT-compiled and efficient code is generated the first time they are called. Then runtime monitoring and dynamic optimization are used to further increase performance based on collected runtime information.

The most-used methods are recompiled with aggressive optimizations and replaced dynamically. Since method sizes tend to be small and scope is very important to the code scheduler, method inlining is used to prepare the code for further optimization. While this can be problematic in Java because of runtime identification of some calls, BEA WebLogic JRockit uses well-tuned heuristics to ensure that inlining provides substantial performance increases.

As shown in Figure 3, an application will typically spend 99% of its execution time in about 10% of its methods. The BEA WebLogic JRockit JVM monitors execution time in each method and targets most-used methods for aggressive optimization. Even if runtime behavior changes over the life of the application, the JVM will identify new methods needing optimization and dynamically optimize them to continuously improve performance.

FIGURE 3.  
Progressive Optimization of  
Methods Over Time



Dynamic code optimization not only increases performance over time, it can also optimize performance for different usage patterns. For example, an application system may have different needs throughout the day or the month as usage patterns change. The dynamic optimization approach ensures that methods suddenly turning into performance bottlenecks at later stages also will be optimized.

Object allocation in BEA WebLogic JRockit is also the responsibility of the code generator. Allocation is thread-local for small objects, meaning that each allocating Java thread has a dedicated area in which to allocate objects. Hence, no time is wasted on synchronization (waiting for locks). For optimized code, small object allocation is inlined. Large object allocation is typically used only for arrays of large or indeterminable size.

## EFFICIENT THREAD MANAGEMENT

The thread management part of BEA WebLogic JRockit is responsible for locks as well as the implementation of wait-primitives. The locks are used to implement the synchronized-keyword in Java. There are two kinds of locks in BEA WebLogic JRockit: *thin locks* and *fat locks*.

Thin locks are used where there has never been contention. Locking and unlocking of thin locks is an extremely fast operation. For single-CPU systems, locking is further optimized by reducing the extra locking primitives required on multiprocessor systems.

If contention on a thin lock is longer than  $x$  (where  $x$  is a sub-millisecond variable that is hardware-dependent), then the thin lock becomes a fat lock. Locking and unlocking of fat locks is slower than for thin locks, but still very fast. On multiprocessor systems, BEA WebLogic JRockit uses a special spin-lock facility that improves the performance of fat locks by spinning for a little while before going to sleep on a locked lock. This can eliminate thousands of CPU cycles worth of unnecessary sleep time, because the thread holding the lock, which may be running on another CPU, will typically release the lock during that initial spin cycle.

# ADAPTIVE MEMORY MANAGEMENT FOR SCALABILITY AND RELIABILITY

Memory management in Java can result in big performance problems, especially with the high user and transaction loads found in enterprise environments. But it also offers tremendous opportunity for performance optimizations. BEA WebLogic JRockit uses a number of mechanisms to automatically increase performance, scalability, and reliability by adapting memory management to suit application behavior and the runtime environment.

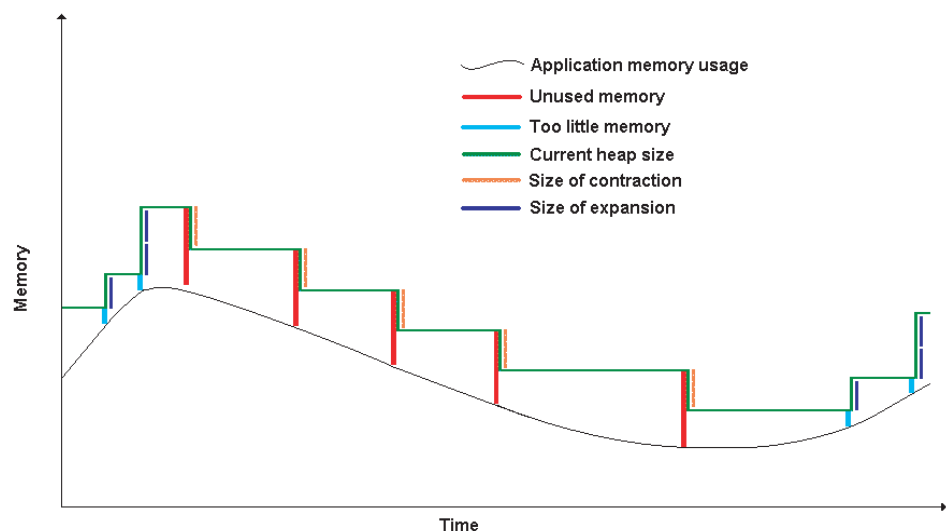
## ADAPTIVE HEAP MANAGEMENT

The issue of heap management is particularly critical in enterprise environments where users typically run multiple application instances simultaneously on the same system. Depending on how the JVM defines its heap size, at some point new instances of the JVM may have insufficient memory to maintain acceptable performance.

The BEA WebLogic JRockit JVM is specially designed to maintain application performance while accounting for overall system memory usage. Each BEA WebLogic JRockit JVM monitors its own memory utilization and dynamically increases or decreases the size of its own heap depending on the needs of its own application. For example, a sales processing application might need more memory during business hours, so it would increase its heap at those times and relinquish heap during non-business hours. However, an e-commerce application might capture additional memory during usage spikes at certain times of day, then release it during off-peak times.

As illustrated in Figure 4, the BEA WebLogic JRockit JVM automatically adapts heap size to meet changing conditions and application requirements.

FIGURE 4.  
Adaptive Heap Management



## ADAPTIVE GARBAGE COLLECTION

Garbage collection (the reclaiming of memory no longer referenced by objects) is a critical factor in Java application performance. Efficient use of memory increases performance and application scalability. On the other hand, the wrong garbage collection approach can be intrusive on application execution and seriously detract from overall system performance and reliability under load. Some applications require the highest possible application throughput and can tolerate periodic garbage collection pauses, while others need consistency and can sacrifice some throughput in order to minimize pause times.

BEA WebLogic JRockit's memory management system offers a selection of garbage collection strategies tailored for different types of applications and environments, as well as an adaptive mode that uses runtime analysis to dynamically adjust the garbage collection strategy to best fit the performance and behavioral requirements of the application.

The BEA WebLogic JRockit garbage collection system uses the following approaches in various combinations to create runtime efficiency during garbage collection:

***Parallel garbage collection*** optimizes throughput by taking advantage of multi-CPU machines to speed up garbage collection. The application is paused temporarily while all the available CPUs are used by the garbage collector to quickly reclaim memory from “dead” (unreferenced) objects.

***Generational garbage collection*** keeps recently allocated objects in a “nursery” until they have survived a certain length of time. The garbage collector periodically sweeps the nursery, removing dead objects and promoting live objects out of the nursery into the long-lived object space. This approach increases the number of pauses due to garbage collection, but the average pause time and, often, the total pause time is significantly reduced because the most frequent garbage collection activities are performed for a memory area smaller than the entire Java heap.

***Single-spaced (non-generational) garbage collection*** configures the Java heap into a single, contiguous space for the allocation of all objects. This approach results in fewer, but longer garbage collection pauses than with generational garbage collection because the entire Java heap has to be traversed to evacuate the dead objects during every garbage collection cycle.

***Concurrent garbage collection*** performs memory reclamation in a background process, resulting in slightly reduced application throughput. However, the number of garbage collections is much reduced, resulting in fewer and shorter pause times because, unlike the parallel garbage collector which stops all application threads during the collection cycle, the concurrent garbage collector conducts part of the collection in the background and does not block the application threads for the entire collection cycle. Concurrent garbage collectors can collect garbage in the background on one or more CPUs while the application continues to run on other CPUs.

The variety of approaches provides the most efficient garbage collection for a range of applications and environments. For example, as shown in Figure 5, with an increasing workload, the parallel garbage collection strategy delivers a higher application throughput compared to a concurrent strategy. However, the parallel strategy also results in higher pause times because there are more dead objects to be collected while the application is suspended during the garbage collection process.

If an application has plenty of heap available and needs to minimize pause times, concurrent garbage collection is a good choice. Figure 5 shows how concurrent garbage collectors keep pause times to a minimum. Concurrent garbage collectors are well-suited to handle very large heaps because, in contrast to the parallel mode, the pause time does not grow with the heap size, but rather depends more on the amount of live data in the heap. Concurrent garbage collectors are also very good for batch-oriented, single-threaded applications running on multi-CPU machines, because they can collect garbage in the background on one or more CPUs while the application continues to run on the other CPUs, thus minimizing the number of garbage collections and the associated pause times. In this situation, garbage collection becomes virtually overhead free for the application.

As the workload increases, parallel garbage collection increases throughput, but also increases pause times. On the other hand, concurrent garbage collection minimizes pause times, but throughput is somewhat less than with parallel garbage collection.

FIGURE 5.  
Parallel vs. Concurrent Garbage Collection

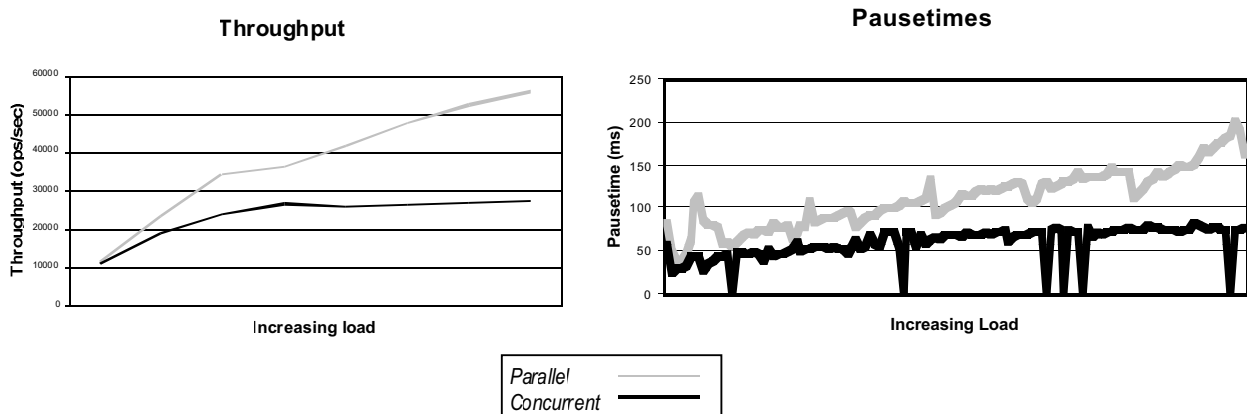
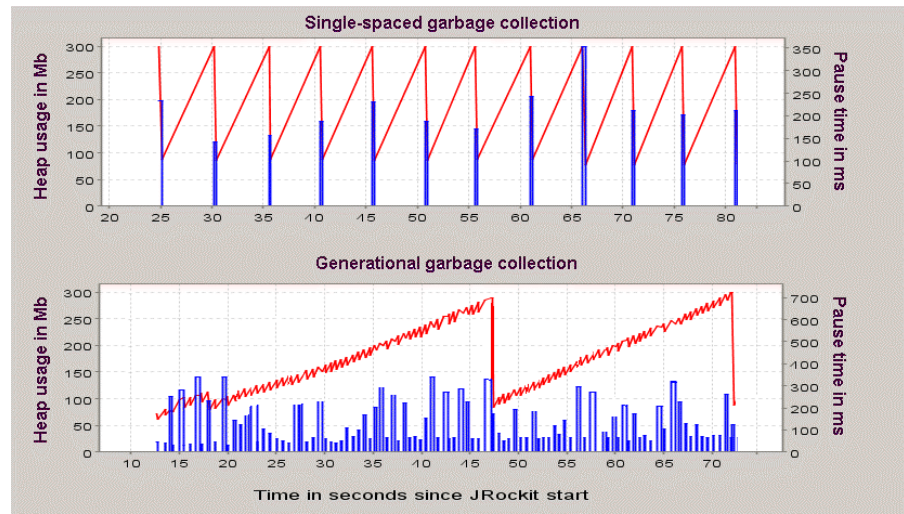


FIGURE 6.  
Single-Spaced vs.  
Generational Garbage Collection



Single-spaced (non-generational) concurrent garbage collection is good for applications that create a lot of long-lived objects, because garbage collection can run infrequently and in the background. Figure 6 shows how single-spaced garbage collection minimizes the number of pauses for an application where heap usage is fairly predictable.

For applications with a lot of short-lived objects, Figure 6 shows how the generational mode results in very frequent garbage collection, but also helps keep heap usage within bounds while keeping pause times as short as possible.

In the graphs in Figure 6, red lines represent memory usage and blue lines represent garbage collection pause times. Generational garbage collection results in fewer whole-heap collections. Generational garbage collection is useful for applications that use a lot of short-lived data. Single-spaced (non-generational) garbage collection should be used for applications with a lot of long-lived objects.

Choosing the best garbage collection for a given application can be complex, since the application behavior can change over time while the application is running. BEA WebLogic JRockit eliminates this complexity by allowing the developer or system administrator to select adaptive garbage collection mode. In fact, BEA WebLogic JRockit is the only JVM with a self-adapting garbage collector that actually switches garbage collection strategies during runtime, automatically choosing the garbage collection algorithm best suited to the current running application. The developer specifies the behavior that is most important for the particular application — minimal pause times or highest throughput — and the adaptive garbage collector automatically configures itself to deliver those characteristics. If the application currently (or

FIGURE 7.  
Adaptive Garbage Collection



temporarily) needs a nursery, the adaptive garbage collector will identify the need and create a nursery. If garbage collection pauses become too long for the application, the adaptive garbage collector will adjust the garbage collection algorithm to prevent long pauses.

In the upper graph of Figure 7, the garbage collection system adapts its algorithm to achieve minimum pause times for the running application, switching between a single-spaced strategy and a generational heap with a nursery. In the lower graph where maximum throughput is desired, the garbage collection system shifts from a generational heap with a nursery to a single-spaced algorithm that would deliver the best throughput.

This unique feature of the BEA WebLogic JRockit JVM simplifies the developer's task and allows applications to achieve an optimal balance between the smallest possible pause times and the highest possible throughput (ability to reclaim used memory). Developers no longer have to spend large amounts of time and effort configuring and tuning the JVM to achieve the desired levels of performance.

## OPTIMIZING MEMORY MANAGEMENT FOR CLIENT AND SERVER ENVIRONMENTS

In order to achieve optimal application performance, it is important that the application starts out with the right amount of memory. Too small a heap will result in out-of-memory errors. Too large a heap could result in long garbage collection pauses or slow overall system performance because other applications are starved for memory.

BEA WebLogic JRockit allows the developer to set initial memory allocation for optimal performance in client development or server environments. By default, BEA WebLogic JRockit configures its heap size and nursery size for a server environment, automatically sizing the heap and nursery according to the number of CPUs in the system and the total system RAM. If the developer or system administrator starts the JVM with the “-client” option, BEA WebLogic JRockit configures its heap size and nursery size optimally for a Java applet in a browser or a single-user Swing application running on a single-CPU PC with a minimum system memory of 128 MB. So in client development mode, the JVM starts with a smaller heap and a pause-time sensitive to a garbage collector.



# MANAGING FOR ENTERPRISE PERFORMANCE, SCALABILITY, AND RELIABILITY

The JVM has the front-row seat on application behavior at runtime, but the Java developer has the business perspective and the ultimate responsibility for application performance. With BEA WebLogic JRockit's unique performance management tools, the JVM is no longer a "black box". The BEA WebLogic JRockit Management Console, Runtime Analyzer, and Monitoring and Management APIs give developers and system administrators an unparalleled level of real-time visibility and control, enabling them to tune application performance and ensure system health through changing usage patterns and business conditions.

## BEA WEBLOGIC JROCKIT MONITORING AND MANAGEMENT APIS

BEA WebLogic JRockit provides a unique capability to monitor and manage the JVM and Java application activity at runtime without introducing a noticeable overhead affecting performance and operation. The BEA WebLogic JRockit Monitoring and Management APIs give applications and multiple external application management tools a consistent and non-contentious means to interface with the JVM and gather runtime information on the application without having to instrument the application byte code.

### Monitoring APIs

These APIs allow for manual and/or programmatic data collection during runtime. Among other capabilities, the Monitoring APIs provide data collection for:

- Monitoring and diagnosis of method-level application operating conditions. The APIs measure frequency and time spent in monitored methods. They also monitor for exceptions.
- Monitoring JVM operating conditions such as garbage collection mode and heap utilization
- Monitoring operating system and hardware operating conditions such as memory availability and CPU utilization

### Management APIs

The Management APIs allow applications or external tools to manually or programmatically modify the runtime characteristics of the application or JVM. Among other capabilities, the Management APIs provide the ability to:

- Modify JVM operating conditions such as heap size, GC parameters, and CPU binding without having to restart the JVM
- Modify application methods such as forceCompilation
- Monitor garbage collection events to identify trends towards overly long GC times
- Gather complete information about threads and frames

## BEA WEBLOGIC JROCKIT MANAGEMENT CONSOLE

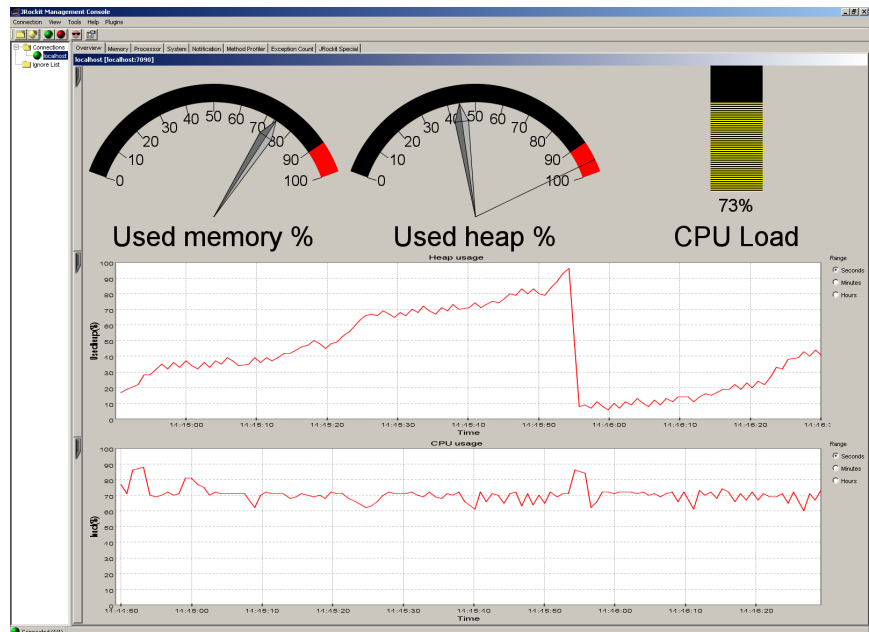
The BEA WebLogic JRockit Management Console (JMC) uses the underlying BEA WebLogic JRockit Monitoring and Management API infrastructure, to give developers and system administrators real-time visibility and control over the inner workings and behaviors of multiple JVM instances over a network. BEA WebLogic JRockit is the only JVM to provide this level of manageability and application visibility. Specifically, it provides up-to-date information on CPU utilization, GC pause times, heap utilization, the number and state of threads, and other runtime behavior such as time spent in individual methods. Thread stack dumps can also be captured through the console. The BEA WebLogic JRockit Management Console gives hands-on control over runtime behavior such as garbage collection parameters. Rule-based alerts and notification of exceptions and boundary conditions such as excessive heap utilization help developers and system administrators to identify and understand application behavior problems and correct them before they cause catastrophic failure.

The BEA WebLogic JRockit Management Console gives visibility and control over JVM and application behavior at runtime as shown in Figure 8.

Among other capabilities, the BEA WebLogic JRockit Management Console also provides:

- Persistent storage of monitored data for off-line analysis
- The ability to programmatically trigger invocation of Java classes from external applications based on notification rules set within the console

FIGURE 8.  
BEA WebLogic JRockit  
Management Console

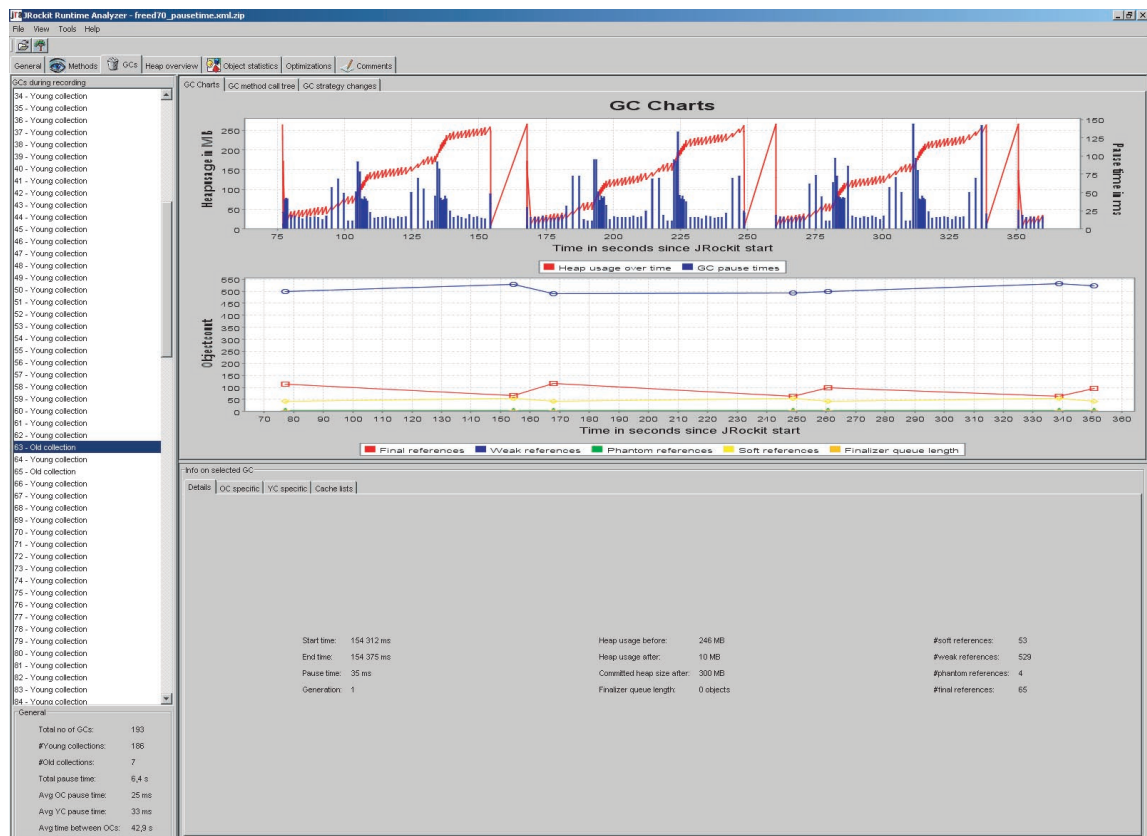


# BEA WEBLOGIC JROCKIT RUNTIME ANALYZER

The BEA WebLogic JRockit Runtime Analyzer takes advantage of BEA WebLogic JRockit's built-in monitoring framework to help developers view and analyze the behavior of applications in production environments. In an offline environment, developers can use the information collected by BEA WebLogic JRockit to analyze an application's runtime behavior and diagnose potential performance-related conditions.

Figure 9 illustrates how the BEA WebLogic JRockit Runtime Analyzer helps developers view and analyze an application's runtime behavior.

FIGURE 9.  
BEA WebLogic JRockit Runtime Analyzer



# BEA WEBLOGIC JROCKIT PERFORMANCE AND SCALABILITY

The BEA WebLogic JRockit JVM leads the industry in performance and scalability. Its progressive optimization provides continuous performance improvements, while adaptive memory management ensures top runtime efficiency and deployment simplicity. Built-in monitoring allows the JVM to maximize its own performance, and runtime management tools enable the developer to fine-tune performance through changing business conditions and application workloads.

In addition to these features, BEA WebLogic JRockit is optimized for performance on systems built from industry-standard Intel-based servers, from 32-bit Intel® Xeon™ processor-based systems to servers using the latest 64-bit Intel® Itanium® 2 processors. BEA WebLogic JRockit is the fastest JVM on the market for IA32 platforms, and the only viable JVM for IA64 platforms. Because the architecture is developed and certified to 100% of the Java 2 Standard Edition (J2SE) specifications, it provides greater flexibility to choose hardware, OS, and middleware vendors. This standards-based optimization allows companies to quickly scale their enterprise application infrastructure while reducing component and operating costs.

When applications are deployed on Intel® Itanium® 2 processor-based platforms with a high performing Java Virtual Machine such as BEA WebLogic JRockit, the Java programming language becomes the ultimate deployment platform for large-scale, server-side, enterprise-class applications. These applications, which typically require large data sets, derive substantial benefit from 64-bit computing by taking advantage of the large amounts of available memory to reduce time-consuming disk swapping. BEA WebLogic JRockit's ability to utilize the larger memory space on IA64 platforms also helps companies to cost-effectively scale their enterprise applications to meet future requirements.

BEA WebLogic JRockit continues to demonstrate superior application performance and price/performance as measured through standardized benchmarks.

## **SPECjbb2000**

SPECjbb2000 is an industry standard benchmark for evaluating the performance of server-side JVMs. The Standard Performance Evaluation Corporation (SPEC) has created the benchmark to measure scalability of JVMs on multi-CPU servers. BEA WebLogic JRockit is designed to scale linearly across multiple CPUs, and has been shown to outperform other JVMs in CPU scalability on this benchmark. More information about the benchmark can be found at SPEC's Web site (<http://www.spec.org>).

### **SPECjAppServer2002**

SPECjAppServer2002 is an Enterprise JavaBeans™ benchmark designed to measure the scalability and performance of J2EE™ application servers and containers. Since all J2EE servers run on top of a JVM, the JVM is implicitly benchmarked along with the application server. BEA WebLogic JRockit demonstrates exceptional performance and scalability on large servers. For more information on the benchmark, refer to the SPEC Web site (<http://www.spec.org>).

## **BOTTOM LINE: SIMPLY SUPERIOR ENTERPRISE JAVA**

BEA WebLogic JRockit is simply the best JVM for enterprise applications. It gives Java developers a straightforward and simple way to achieve performance from the start of development through the lifetime of a Java application. Its unique features, including progressive optimization, adaptive performance technology and runtime manageability, ensure the speed, transaction capacity, scalability, and reliability required for enterprise Java applications, while optimizations on industry-standard platforms reduce enterprise system costs. With BEA WebLogic JRockit, developers spend less time figuring out how to make the application perform and more time helping the business to perform.

# AVAILABILITY

The BEA WebLogic JRockit JVM is included as part of BEA WebLogic Enterprise Platform™ and BEA WebLogic Server™. In addition, it is available for download at <http://commerce.bea.com/index.jsp> for the following environments on Intel Architecture platforms:

- Microsoft Windows (IA32 and IA64)
- Red Hat Enterprise Linux (IA32 and IA64)
- SuSE Linux ES (IA32 and IA64)

For current platform support, please refer to <http://edocs.bea.com>.

# ABOUT BEA

BEA Systems, Inc. (Nasdaq: BEAS) is the world's leading application infrastructure software company, providing the enterprise software foundation for more than 15,000 customers around the world, including the majority of the *Fortune* Global 500. BEA and its WebLogic® and Tuxedo® brands are among the most trusted names in business. Headquartered in San Jose, Calif., BEA has 77 offices in 31 countries and is on the Web at [www.bea.com](http://www.bea.com).



BEA SYSTEMS, INC.  
2315 North First Street  
San Jose, CA 95131 U.S.A.  
Telephone: +1.408.570.8000  
Facsimile: +1.408.570.8901  
[www.bea.com](http://www.bea.com)

CWP0671E1203-1A