BEA WebLogic Real Time™

Introduction to WebLogic Real Time

Version 1.0
January 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

CHAPTER 1

Overview of WebLogic Real Time

This section contains information on the following subjects:

What is WebLogic Real Time?

WebLogic Real Time provides lightweight, front-office infrastructure for low latency, event-driven applications. For companies in highly competitive environments where performance is key and every millisecond counts, WebLogic Real Time provides the first Java-based real-time computing infrastructure.

For example, for certain types of applications, particularly in the Telecom and Finance industries, stringent requirements are placed on transaction latency. When these applications are written in Java, the unpredictable pause times caused by garbage collection can have a profound and potentially harmful affect on this latency.

For this reason, WebLogic Real Time's proprietary JRockit JVM introduces "deterministic" garbage collection, a dynamic garbage collection priority that ensures extremely short pause times and limits the total number of those pauses within a prescribed window. Such short pauses

can greatly lessen the lessen the impact of the deterministic garbage collection when compared to running a normal garbage collection.

WebLogic Real Time includes the following components:

- BEA WebLogic Server 9.1®
- BEA JRockit® 5.0 R26, with deterministic garbage collection for minimal transaction latency

  Note: For WebLogic Real Time 1.0, this requires acquiring a special JRockit license, as described in "Installing and Updating License Files" in the WebLogic Real Time *Installation Guide*.

- Spring 1.2.6 Framework for BEA

For more information about the key components of the WebLogic Real Time, please refer to "Architecture" on page 2-1.

For a listing of the hardware and software configurations supported by WebLogic Real Time, see "WebLogic Real Time 1.0 Supported Configurations" on page 1-3.

Example Use Cases

These use cases provide examples of how WebLogic Real Time can provide solutions for high-performance environments with response-time sensitive applications.

Derivative Exchange Defies Arbitrage Traders

An investment arm of a large retail bank provides an exchange for derivatives of European securities. It is an over-the-counter (OTC) request-for-quote and execution system (but provides no settlement and clearing services). A broker submits a request for a quotation and includes the investment identifier and quantity. The system accepts the quotation and applies certain business rules. Depending upon the investment identifier and market conditions, the request is routed to a particular third-party market-maker who then calculates and provides the bid and ask price for the derivative. The response is returned to the broker via the OTC exchange. The broker can then execute the trade of the derivative through a subsequent request, which is routed via the OTC exchange to the appropriate market maker.

The complication with this arrangement is that arbitrage traders can take advantage of the latency delay in the bank's OTC exchange infrastructure because the arbitrage trader can measure the latency that occurs during the small period in which the request for quotation is handled. In a fast moving market, price changes of the derivative may occur within this latency period. This

presents an opportunity for an arbitrage trader to take advantage of inefficiency in the marketplace and expose the investment bank to intolerable risk.

The investment bank requires a very high performance-driven software infrastructure, such as WebLogic Real Time. It requires that the latency of the OTC exchange be extremely low. Specifically, to combat arbitrage traders, the latency of the exchange's infrastructure must be less than the latency of the arbitrage traders' infrastructure. In this way, the arbitrage traders' data becomes stale before the exchange's, and therefore is not actionable.

Competition-Beating Risk Calculation Infrastructure

A large investment bank is a market-maker for fixed income securities. A request-for-quote (RFQ) is received from an inter-dealer market electronic communication network (ECN), such as TradeWeb. This RFQ would have been submitted to a number of entities. To be competitive, it is vital that the quotation is returned as quickly as possible with the best possible price. Therefore, a minimum amount of latency is necessary to ensure that the investment bank wins customers, or at least, the latency is less than that of the organization's competitors.

During the quotation process, a risk and pricing model is executed to determine the quote price to provide to the customer. Because of the complexity of these calculations, they are currently performed overnight. The result is a stratum of at least four grades of risk advisories that govern fixed rate securities prices. Note that there is at least a twelve-hour lag in these risk calculations. This leads to a risk window since the calculations are stale even at the start of next-day business. To lower this risk, and potentially provide better rates to customers, a real-time risk and pricing calculator would be required. WebLogic Real Time provides a latency-adverse infrastructure to make this feasible.

WebLogic Real Time 1.0 Supported Configurations

WebLogic Real Time 1.0 is certified to be compatible with the following platforms and software:

- "List of Supported Operating System Configuration" on page 1-3
- "List of Supported Software" on page 1-4

List of Supported Operating System Configuration

Table 1-1 lists the operating system and hardware configurations on which BEA supports WebLogic Real Time 1.0.

Table 1-1 Operating System and Hardware Supported by WebLogic Real Time 1.0

Vendor and Operating System	Version	Hardware	WebLogic Real Time 1.x
Microsoft Windows Server 2003	• Standard • Enterprise • Datacenter	x86	1.0
Microsoft Windows XP	SP2	x86	1.0
Red Hat Enterprise Linux	3.0 AS, ES, WS	x86	1.0
	4.0 AS, ES, WS	x86	1.0

List of Supported Software

Table 1-2 lists the software supported by WebLogic Real Time 1.0.

Table 1-2 Software Supported by WebLogic Real Time 1.0

Software	Version
WebLogic Server 9.1	9.1
BEA JRockit	5.0 R26.0.0 Build: R26.0.0-189_CR256719_CR257184-54965-1.5.0_04-20051219-*
Spring Framework	1.2.6

WebLogic Real Time Support

You are entitled to support if you have a support agreement with BEA.

Terminology

Table 1-3 defines the terms and acronyms used this document:

Table 1-3 Terminology

Terms	Definition
Real-time	A level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process (for example, to present visualizations of the weather as it constantly changes).
Latency	An expression of how much time it takes for data to get from one designated point to another.
Throughput	The amount of work that a computer can do in a given time period.
Deterministic garbage collection	Short, predictable pause times for memory heap garbage collection, which is the process of clearing dead objects from the heap, thus releasing that space for new objects.

CHAPTER 2

Architecture

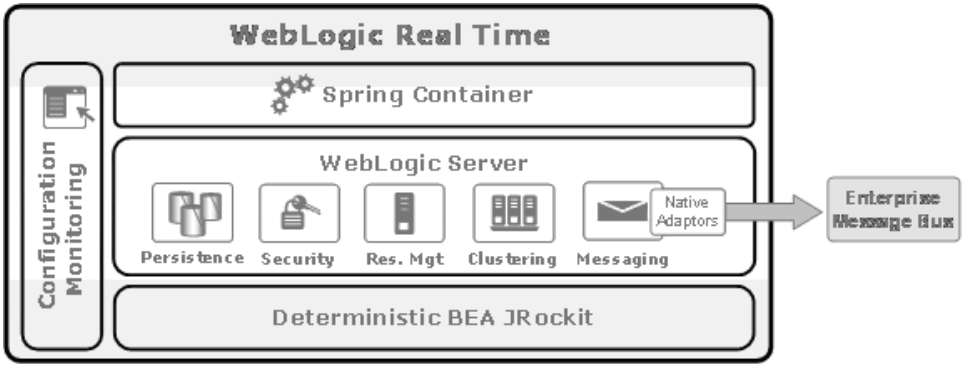
This chapter describes the WebLogic Real Time Server architecture.

Key Components

WebLogic Real Time provides lightweight, front-office infrastructure to allow WebLogic Server 9.1 and BEA JRockit JDK 5.0 with Deterministic Garbage Collection, in combination with a BEA optimized Spring 1.2.6 container, to be used in latency-sensitive, real-time environments, such as the financial services industry.

The components of WebLogic Real Time are illustrated in Figure 2-1.

Figure 2-1 WebLogic Real Time Components



WebLogic Server 9.1

WebLogic Server 9.1 is a scalable, enterprise-ready Java Two Enterprise Edition (J2EE) application server. The WebLogic Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications based on Service Oriented Architectures (SOA). SOA is a design methodology aimed at maximizing the reuse of application services.

The WebLogic Server 9.1 complete implementation of The Sun Microsystems J2EE 1.4 specification provides a standard set of APIs for creating distributed Java applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems. End-user clients access these applications using Web browser clients or Java clients.

In addition to the J2EE implementation, WebLogic Server enables enterprises to deploy mission-critical applications in a robust, secure, highly available, and scalable environment. These features allow enterprises to configure clusters of WebLogic Server instances to distribute load, and provide extra capacity in case of hardware or other failures. Diagnostic tools allow system administrators to monitor and tune the performance of deployed applications and the WebLogic Server environment itself. You can also configure WebLogic Server to monitor and tune application throughput automatically without human intervention. Extensive security features protect access to services, keep enterprise data secure, and prevent malicious attacks.

For more introductory information on WebLogic Server 9.1, see Introduction to BEA WebLogic Server and BEA WebLogic Express.

BEA JRockit 5.0 Deterministic Garbage Collection

Memory management relies on effective *garbage collection*, which is the process of clearing dead objects from the heap, thus releasing that space for new objects. WebLogic Real Time uses a dynamic "deterministic" garbage collection priority (`-Xgcprio:deterministic`) that is optimized to ensure extremely short pause times and limit the total number of those pauses within a prescribed window.

For certain types of applications, particularly in the Telecom and Finance industries, stringent requirements are placed on transaction latency. When these applications are written in Java, the unpredictable pause times caused by garbage collection can have a profound and potentially harmful affect on this latency.

However, shorter deterministic pause times do not necessarily equal higher throughput. Instead the goal of the deterministic garbage collection is to lower the *maximum* latency for applications that are running when garbage collection occurs. Such shorter pause times should lessen the impact of the deterministic garbage collection compared to running a normal garbage collection.

For more information on the deterministic garbage collector, see "Minimal Transaction Latency (The "Deterministic" Garbage Collector)" in the BEA JRockit *Memory Management Guide*.

Note: For WebLogic Real Time 1.0, this requires acquiring a special JRockit license, as described in "Updating Your BEA JRockit License for the Deterministic Garbage Collector" in the WebLogic Real Time *Installation Guide*.

The BEA Spring 1.2.6 Framework

The Spring 1.2.6 Framework for BEA WebLogic Real Time is a lightweight application framework that greatly simplifies the development effort over traditional J2EE development. Spring has proven itself to be very flexible, easy to use, and capable of working in highly latency-sensitive environments. It allows the use of Plain Old Java Objects (POJOs) in the place of EJBs, still allows access to RASP capabilities of an application server, and enforces modular and reusable coding practices.

The Spring 1.2.6 Framework for BEA is fully supported with WebLogic Real Time 1.0 and can be downloaded from the WebLogic Real Time Product download page at `http://commerce.bea.com/showproduct.jsp?family=SPRING&major=1.2.6&minor=0`.

Spring 1.2.6 is also distributed with the WebLogic Real Time CD-ROM kit when the media is ordered.

For more information on the Spring framework for BEA, see:

- Spring Integration with WebLogic Server on BEA's dev2dev

  Note: The Spring download link at the end of this white paper is for Spring 1.2.5, which is not the version supported with WebLogic Real Time 1.0.

- Spring Applications Reference in *Developing Application with WebLogic Server*
- Spring Framework
- Interface21

CHAPTER 3

Tuning WebLogic Real Time

This section contains information on the following subjects:

Basic Tuning Guidelines

This section provides some basic tuning guidelines for configuring WebLogic Server, J2EE, and JMS applications for response-time sensitive applications.

Basic Environment and Application Tuning

- Ensure that CPUs are not maxed out on servers or clients.
- Highly active Java clients may require running "deterministic" garbage collection as well.
- There may be a "warm-up" period of one-to-three minutes before response times achieve desired levels.
- For server-side EJBs, MDBs, and servlets ensure that there are enough concurrent instances configured to respond immediately to client requests (if all instances are active, this is a sign that client requests are queuing up behind each-other on the server).

JMS Application Tuning

Use these guidelines when using WebLogic JMS applications with WebLogic Real Time.

- Consider using asynchronous consumers rather than synchronous consumers.

  For more information on JMS consumers, see "Best Practices for Application Design" in *Programming WebLogic JMS*.

- Tune all JMS connection factory Messages Maximum settings to 1. This can potentially provide better latency at the expense of possibly lowering throughput. Similarly, configure your MDBs to refer to a custom connection factory with the following settings:

  – Messages Maximum = `1`

  – XA Connection Factory Enabled = `enabled`

  – Client Acknowledge Policy = `ACKNOWLEDGE_PREVIOUS`

  For more information on configuring JMS connection factories, see "Configure connection factories" in the *Administration Console Online Help*.

- For consumers of non-persistent messages from queues, consider using the WebLogic JMS `WLSession NO_ACKNOWLEDGE` extension.

- Ensure that your Spring JMS Templates leverage resource reference pooling (otherwise, they negatively impact response times as they implicitly create and close JMS connections, sessions, and producers once per message).

  Note: Resource reference pooling is not suitable if the target destination changes with each call, in which case change application code to use *regular* JMS and cache the JMS connections, sessions, producers, and consumers.

JVM Tuning for Real-Time Applications

These tuning suggestions can further improve performance and decrease pause times when using the deterministic garbage collector.

- Setting the minimum heap size (`-Xms`) smaller or the maximum heap size (`-Xmx`) larger affects how often garbage collection will occur and determines the approximate amount of live data an application can have. See "-X Command-line Options" in the BEA JRockit *Reference Manual*.

- Increasing the page size (`-XXlargePages`) can increase performance and lower pause times by limiting cache misses in the translation look-aside buffer (TLB). See "-XX Command-line Options" in the BEA JRockit *Reference Manual*.
- Try to limit the amount of reference objects that are used, such as `Weak-`, `Phantom-`, and finalizers. These references require special handling, and if they occur in large numbers then pause times can become longer than 30ms.
- Running on slower hardware, with a different heap size and/or with more live data may break the deterministic behavior. In these cases, you might need to increase the pause time target by using the `-XpauseTarget` option. See "-X Command-line Options" in the BEA JRockit *Reference Manual*.
- Try adjusting the garbage collection trigger (`-XXgctrigger`) to limit the amount of heap space used. This way, you can force the garbage collection to trigger more frequent garbage collections without modifying your applications. The garbage collection trigger is somewhat deterministic, since garbage collection starts each time the trigger limit is hit. See "Adjust the Garbage Collection Trigger" in the BEA JRockit *Configuration and Tuning Guide*.

  Note: If the trigger value is set to low, the heap might get full before the garbage collection is finished, causing even longer pauses for threads since they have to wait for the garbage collection to complete before getting new memory. Typically, memory is always available since a portion of the heap is free and any pauses are just the small pauses when the garbage collection stops the Java application.

More Tuning Information

This section contains pointers to additional performance and tuning information.

JRockit JVM

- BEA JRockit Garbage Collection Models contains information on all of the JRockit garbage collection options.
- Tuning the JRockit JVM provides information on tuning the JRockit JVM.

WebLogic Server

- *WebLogic Server Performance and Tuning* contains information on monitoring and improving the performance of WebLogic Server applications.
- *Best Practices for Application Design* in *Programming WebLogic JMS* provides design options for WebLogic Server JMS, application behaviors to consider during the design process, and recommended design patterns.