Oracle® JRockit Real Time

Introduction

Release 4.0

E15071-02

April 2010

This document contains background on Oracle JRockit Real Time and instructions for its use.

ORACLE®

Oracle JRockit Real Time Introduction, Release 4.0

E15071-02

Primary Author: Edwin Spear

Contributor: Robert Ottenhag

Contents

Preface

This document contains background on Oracle JRockit Real Time and instructions for its use.

About the Document

The document contains the following chapters:

- Chapter 1, "Overview", which provides basic background information on Oracle JRockit Real Time.
- Chapter 2, "Getting Started with JRockit Real Time", which describes how to start using Oracle JRockit Real Time.
- Chapter 3, "Tuning Real Time Applications for Deterministic Garbage Collection", which provides guidelines for tuning applications for the Oracle JRockit JVM deterministic garbage collector that is included with JRockit Real Time.
- Chapter 4, "Using JRockit Real Time with Other Oracle Products", which describes the tasks necessary for using JRockit Real Time with other Oracle products.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

Oracle JRockit 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, Oracle JRockit Real Time 4.0 provides a Java-based real-time computing infrastructure.

This section contains information on the following subjects:

1.1 Benefits of JRockit Real Time

Certain types of applications, particularly in those in the Telecom and Finance industries, place stringent requirements on transaction latency. When written in Java, these applications can experience unpredictable pause times caused by garbage collection. This can have a profound and potentially harmful affect on transaction latency.

For this reason, JRockit Real Time features *deterministic garbage collection*, a dynamic garbage collection priority that ensures extremely short pause times. Such short pauses can greatly lessen the impact of the deterministic garbage collection when compared to running a normal garbage collection.

1.2 JRockit Real Time Compatibility

JRockit Real Time is fully compatible with Oracle JRockit JDK R28.0 and all applications certified with the latter will work on the former without need for additional certification. This means that all Oracle applications supported by Oracle JRockit JDK R28.0 (whether based upon J2SE 5.0 or Java SE 6) are also supported with JRockit Real Time.

JRockit Real Time also supports standalone Java applications running on Java SE 6 and J2SE 5.0 runtime environments, as well as Spring Framework-based applications, as described in Section 1.4, "Software Components."

JRockit Real Time performance still depends upon application type and size, so you will need to verify that you applications meet the base requirements on hardware, heap size and other metrics for optimal performance.

1.3 Sample Use Cases

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

1.3.1 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 JRockit 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.

1.3.2 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. JRockit Real Time provides a latency-adverse infrastructure to make this feasible.

1.4 Software Components

JRockit Real Time supports Java applications running on such Oracle products as (but not limited to) Oracle WebLogic Event Server 2.0, Oracle WebLogic Server 10.0 (or

higher), and Oracle WebLogic Server 8.1. It also supports standalone Java applications running on Java SE 6 and J2SE 5.0 runtime environments.

JRockit Real Time includes the following software components:

- Oracle JRockit JDK 6 R28.0
- Oracle JRockit JDK 5.0 R28.0
- Deterministic Garbage Collection
- Oracle JRockit Flight Recorder

1.4.1 Oracle JRockit JDK 6 R28.0

The Oracle JRockit JDK 6 R28.0 is certified to be compatible with Java SE 6 (update 17). This version includes the Deterministic Garbage Collector for dynamic garbage collection priority that ensures extremely short pause times, as described in Section 1.4.3, "Deterministic Garbage Collection". It also includes the JRockit Flight Recorder, which provides internal metrics that are useful for profiling the Oracle JRockit JVM (see Section 1.4.4, "Oracle JRockit Flight Recorder") and the JRockit Memory Leak Detector.

For a listing of the hardware and software configurations supported by JRockit Real Time, see Section 1.5, "Supported Configurations for JRockit Real Time".

1.4.2 Oracle JRockit JDK 5.0 R28.0

The Oracle JRockit JDK 5.0 R28.0 is certified to be compatible with J2SE 5.0 (update 22). The 5.0 R28.0 JVM includes the Deterministic Garbage Collector for dynamic garbage collection priority that ensures extremely short pause times, as described in Section 1.4.3, "Deterministic Garbage Collection". It also includes the JRockit Flight Recorder, which provides internal metrics that are useful for profiling the Oracle JRockit JVM (see Section 1.4.4, "Oracle JRockit Flight Recorder") and the JRockit Memory Leak Detector.

For a listing of the hardware and software configurations supported by JRockit Real Time, see Section 1.5, "Supported Configurations for JRockit Real Time".

1.4.3 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. JRockit Real Time uses a dynamic "deterministic" garbage collection priority (`-Xgc: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 such applications.

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 the *Oracle JRockit Diagnostics and Tuning Guide*, available on the Oracle Technology Network.

For standalone or Spring-Based Java applications, enable the Deterministic Garbage Collector by doing one of the following:

- Enter the `-Xgc:deterministic` option from a Java command line.
- Use the sample startup scripts, `startRealTime (.cmd/.sh)`, that demonstrate how to start the Oracle JRockit JVM with deterministic garbage collection enabled.

1.4.4 Oracle JRockit Flight Recorder

At its most basic, the JRockit Flight Recorder is a rotating buffer of diagnostics and profiling data that is always available, on demand. You might consider it a sort "time machine" that enables you to go back in time to gather diagnostics data leading up to an event. The data stored in the rotating buffer includes JVM and application events. Since JFR is always-on using the default configuration will not result in any performance overhead.

Assuming you are running a Flight Recorder-compliant version of Oracle JRockit (that is, version R28.0.0 or later), you can see the content of a flight recordings on the Flight Recorder GUI in the Mission Control 4.0 Client. The GUI shows what has been recorded, current recording settings, and runtime parameters. The GUI is comprised of a series of tabs that aggregate performance data into logical groups.

1.5 Supported Configurations for JRockit Real Time

Oracle JRockit Real Time is supported on the same configurations (hardware and platform) as Oracle JRockit JVM R28, with the exception of Windows Itanium and Linux Itanium configurations: it is not supported on those configurations. For a complete list of supported configurations, see the Oracle JRockit JDK R28 Certification Matrix at:

http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html

2

Getting Started with JRockit Real Time

This section contains basic "getting started" information about Oracle JRockit Real Time 4.0. It contains these subsections:

2.1 Version Support

JRockit Real Time is bundled with the following versions of the Oracle JRockit JDK:

- Oracle JRockit JDK 6 Update 17
- Oracle JRockit JDK 5.0 Update 22

2.2 Starting JRockit Mission Control

To start JRockit Mission Control, use the following procedure.

1. Ensure that your `JAVA_HOME` environment variable points to the root folder of the Oracle JRockit JDK included in JRockit Real Time.

   Use this command for Windows platforms (This example assumes that you are using the Windows Command Prompt—DOS—or compatible command shell and have selected the default product installation directory):

   ```
   set JAVA_HOME=%ProgramFiles%\JRockit Real Time \wlrt<wlrt_version>-<java_
   version>\bin
   ```

   Use this command for Linux and Solaris platforms (This example assumes that you are using the UNIX bash shell or compatible command shell and have selected the default product installation directory):

   ```
   export JAVA_HOME=$HOME/wlrt<wlrt_version>-<java_version>/bin
   ```

2. Open up a command window.
3. Run the `jrmc` executable file, located in the `%JAVA_HOME%\bin` directory:

   Windows: > `%JAVA_HOME%\bin\jrmc`

   Linux: > `${JAVA_HOME}/bin/jrmc`

3

Tuning Real Time Applications for Deterministic Garbage Collection

This section contains the following guidelines for tuning your applications for the Oracle JRockit JVM deterministic garbage collector that is included with Oracle JRockit Real Time 4.0.

> **Note:** For more information on adjusting other non-standard start-up commands available with JRockit, see the *Oracle JRockit Command-Line Reference* on the Oracle Technology Network.

3.1 Basic Environment Tuning

Use these guidelines for configuring your environment to use JRockit Real Time.

- *Ensure that CPUs are not at maximum capacity out on servers or clients*

  If an application takes a majority of the CPU, then the deterministic GC performance may actually degrade the average latency. The reason is that deterministic GC will do continuous GC and the GC will be competing with the application for CPU cycles. It is best that the CPU is not fully utilized to get the best latency. A best practice is to run your benchmarks at various loads (with and without deterministic GC) to determine the optimal load.

- *Too many active threads can cause increased latency due to context switching*

  The "sweet-spot" number is generally one thread per virtual CPU (i.e., counting dual-core and HyperTransport as separate CPUs), but leaving one CPU free for background GC work. However, if you make external calls (e.g., to a database), then it does make sense to allocating a few extra threads to utilize idle cycles.

  For information on tuning JRockit garbage collection threads, see Section 3.5.9, "Adjust the Amount of Garbage Collection Threads for Processors".

3.2 Basic Application Tuning

Use these guidelines when designing your applications for JRockit Real Time.

- Understand your application code and how to measure latency.
- Avoid making synchronous calls to slow back-office systems as part of a transaction as this defeats the purpose of real-time. Conversely, make sure any non-critical calls are handled asynchronously through work thread pools, or by using JMS.
- Minimize memory allocation. If possible, allocate and free memory for a single transaction in a *chunk* as this helps avoid fragmentation of the Java heap. Also, minimize the amount and size of your objects.
- Control memory utilization by avoiding rampant memory allocation and allocating many large arrays.
- Free all objects as soon as possible; otherwise, objects that become unreferenced during a garbage collection might still be marked alive if they where referenced when the DetGC marked all live objects.
- Avoid long critical sections in your code, as synchronized blocks of Java code may cause a transaction to block.
- Avoid long linked structures; the deterministic GC needs to iterate through these objects.
- If transactions span more than one highly-active JVM, each such JVM may need to run Deterministic GC. For example, if a transaction is initiated by a Java client JVM, and the transaction includes both JMS server and J2EE server operations, all three JVMs may require Deterministic GC to reliably meet maximum latency criteria.

3.3 J2EE Application Tuning

Use these guidelines when tuning your J2EE applications for JRockit Real Time.

- 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).
- Make sure that resource pools contain enough instances so that threads are not forced to wait for resources. In J2EE for example, tune the EJB `max-beans-in-free-pool` property and tune thread pool sizes

3.4 JMS Application Tuning

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

- Consider using asynchronous consumers rather than synchronous consumers.

  For more information on JMS consumers, see "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 the Administration Console Online Help.
- For consumers of non-persistent messages from queues, consider using the WebLogic JMS WLSession `NO_ACKNOWLEDGE` extension.
- For improved non-persistent messaging performance, consider using the WebLogic JMS "One-Way Send Mode" feature available with WebLogic Server 9.2. See "Tuning WebLogic JMS" in *WebLogic Server Performance and Tuning*.

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.

3.5 JVM Tuning for Real-Time Applications

These tuning suggestions can further improve performance and decrease pause times when using the JRockit JVM deterministic garbage collector. For more information on the deterministic garbage collector, see the *Oracle JRockit Performance Tuning Guide* available on the Oracle Technology Network.

3.5.1 Allow For a Warm-up Period

There may be a *warm-up period* before response times achieve desired levels. During this warm-up, JRockit JVM will optimize the critical code paths. The warm-up period is application and hardware dependent, as follows:

- For smaller applications (in terms of amount of Java code) with high loads that are running on fast hardware, there may be a warm-up period of one-to-three minutes.
- For large applications (in terms of amount of Java code) with low loads that are running on slow hardware (in particular, most SPARC hardware), there may be a warm-up period of approximately thirty minutes.

3.5.2 Adjust Min/Max Heap Sizes

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. To begin with, try using the following heap sizes:

```
java -Xms1024m -Xmx1024m -XgcPrio:deterministic -XpauseTarget=30
```

For more information, see "-XX Command-line Options" in the *Oracle JRockit Command Line Reference*.

3.5.3 Increase or Decrease Pause Targets

- If you specify `-Xgcprio:deterministic` without the `pauseTarget` option, it will be set to a default value, which in this release is 30 milliseconds.
- 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 default pause time target (30 milliseconds) by using the `-XpauseTarget` option. The maximum allowable value for the `pauseTarget` option is currently 5000 milliseconds.
- Conversely, if you want to test your application for the lowest possible pause time, you can lower the default `-XpauseTarget` value down to a minimum value. In this release, the minimum value is 10 milliseconds.

For more information, see "-XX Command-line Options" in the *Oracle JRockit Command Line Reference*.

3.5.4 Set the Page Size

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 Oracle JRockit *JVM Command-Line Reference*.

3.5.5 Determine Optimal Load

Do not be overcautious in terms of load. The deterministic garbage collector can handle a fair amount of load without breaking its determinism guarantees. Too little load means the JVM's optimizer and GC heuristics have too little information to work with, resulting in sub-par performance. A best practice is to run your benchmarks at various loads (with and without deterministic GC) to determine the optimal load.

3.5.6 Analyze GC With JRockit Verbose Output

JRockit JVM verbose output normally doesn't incur a measurable performance impact, and is quite useful for analyzing JVM memory and GC activity. Table 3–1 defines recommended verbose options for analyzing JVM memory and GC activity.

Table 3–1 JRockit JVM Verbose Output Options

Option	What it does...
`-Xverbose:opt,memory,memdbg, gcpause,compact,license`	For GC and memory analysis.
`-Xverboselog:verbose-jrockit.log`	Redirects verbose output to the designated file.
`-Xverbosetimestamp`	Prints a formatted date before each verbose line.

3.5.7 Limit Amount of Finalizers and Reference Objects

Try to limit the amount of Finalizers and reference objects that are used, such as `Soft-`, `Weak-`, and `Phantom-` references. These types require special handling, and if they occur in large numbers then pause times can become longer than 30ms.

3.5.8 Adjust the Garbage Collection Trigger

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 the *Oracle JRockit Performance Tuning Guide* available on the Oracle Technology Network:

> **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.

3.5.9 Adjust the Amount of Garbage Collection Threads for Processors

With the variety of sophisticated processing hardware currently available (HyperTransport, Strands, Dual Core, etc.), the JRockit JVM may not be able to determine the appropriate number of GC threads it should start. The current recommendation is to start one thread per physical CPU; that is, one thread per chip not per core. However, having too many garbage collection threads could affect the latency of applications since more threads will be running on the system, which might saturate the CPUs, and thus affect the Java application. Conversely, setting them too low could increase the mark phase of the GC, since less parallelism is possible. For example, on a dual core Intel Woodcrest machine with four cores the recommended number of GC threads is two, which is the same as the number of processors in the machine.

To see how many garbage collection threads that the JRockit JVM uses on your machine, start the JRockit JVM with `-verbose:memdbg` and then check for the following lines that are printed during startup:

```
[memdbg ] number of oc threads: <num>
[memdbg ] number of yc threads: <num>
```

If necessary, adjust the number of GC threads using the `-XXgcthreads:<# threads>` parameter.

For more information, see "-XX Command-line Options" in the *Oracle JRockit Command Line Reference*:

3.6 More Tuning Information

For additional performance and tuning information, see the *Oracle JRockit Performance Tuning Guide* on the Oracle Technology Network.

4

Using JRockit Real Time with Other Oracle Products

Oracle JRockit Real Time 4.0 is fully compatible with Oracle JRockit JVM R28.0 and is therefore supported by Oracle when used with Oracle products that are supported with Oracle JRockit JVM R28.0. This chapter describes the tasks necessary for using JRockit Real Time with other Oracle products. It includes information on the following subjects:

4.1 Getting Started

To use JRockit Real Time with an Oracle product, you must download and install the Oracle product and JRockit Real Time separately and then make the following configuration changes:

- Update the Oracle product start script to use the `WLRT JAVA_HOME`.
- Remove any `-Xgc` and `-Xgcprio` parameters from the Java command line.
- Add `-Xgcprio:deterministic` and `-Xpausetarget=nnms` (where `nn` is the desired pause target) to the Java command line

Selecting a lower pause target will result in shorter garbage collection pauses, with the caveats outlined in "Tuning the Pause Target" in the *Oracle JRockit Diagnostics and Troubleshooting Guide* on the Oracle Technology Network.

Here are some recommendations for the most common combinations:

4.1.1 Oracle WebLogic Server

Oracle WebLogic Server can be enabled to use JRockit Real Time either by following the generic instructions above or by using a preconfigured domain template. Oracle has prepared templates for the following Oracle WebLogic Server versions:

- Oracle WebLogic Server 8.1 domain template.
- Oracle WebLogic Server 10.0 domain template.

Oracle has verified that larger J2EE applications (including SPECjAppServer2004) works well with a 30ms pause targets. Lower pause targets might be possible for smaller J2EE applications or faster hardware.

4.1.2 WebLogic Event Server

The Oracle WebLogic Server has been extensively optimized for use with JRockit Real Time, so a good starting point for the pausetarget is 10 ms. Oracle has verified that small WLEvS applications on recent x86 hardware work with pausetargets down to 3 ms. For more information on tuning Oracle WebLogic Server with JRockit Real Time, see the Oracle WebLogic Server documentation.