BEA JRockit JDK

Tuning the BEA JRockit JVM

JRockit 5.0 Service Pack 2
June 2005

Contents

CHAPTER 1

Introduction to Tuning BEA JRockit JVM

There are some ways that you can improve performance of your BEA JRockit JVM, even though most of the tuning that is needed is done automatically when the JVM adapts to its underlying hardware and to the applications running on it. For example, you can control how much Java memory you want the JVM to use. This guide will help find ways to where it is possible to tune JRockit so that it will increase performance and work better for your application and system.

This section includes information on the following subjects:

- How BEA JRockit is Tuned
- JVM Tuning Terminology
- What You'll Find in Tuning BEA JRockit JVM

How BEA JRockit is Tuned

BEA JRockit JVM has a number of non-standard startup parameters, called `-X` options, that allow you to better tune the JVM for your specific application. This guide documents the different startup parameters and what you need to know about setting them to be able to tune the JVM to ensure optimal performance for your application.

JVM Tuning Terminology

Before continuing, there are some terms you should understand. You may already be familiar with some of the terms, especially if you have read any other documents about garbage collectors.

Garbage collector

The garbage collector is the key to effectively managing BEA JRockit's memory system, which is the ultimate goal of JVM tuning. Garbage collection is the process of clearing dead objects from the heap, thus releasing that space for new objects.

Application throughput

The garbage collector is optimized for application throughput. This means that the garbage collector works as effectively as possible, giving as much CPU resources to the Java threads as possible. This may, however, cause non-deterministic pauses when the garbage collector stops all Java threads for garbage collection.The throughput priority should be used when non-deterministic pauses do not impact the application's behavior.

Pause time

The garbage collector is optimized to limit the length of each garbage collection pause where all Java threads are stopped for garbage collection. This may result in lower application throughput, as the garbage collector uses more CPU resources in total than when running with the throughput priority. The pause time priority should be used when the application depends on an even performance. Use `-Xpausetarget` to set a target length for the garbage collection pauses.

What You'll Find in Tuning BEA JRockit JVM

This guide is divided into these sections:

- Tuning BEA JRockit JVM describes the basic tuning parameters for the JVM. The instructions in this section describe default and optimal heap and nursery settings and how to use them to tune the JVM.
- Basic Tuning Tips and Techniques contains some helpful hints for maximizing system performance by tuning BEA JRockit to provide either optimal memory throughput or minimal garbage collection pause times.

CHAPTER 2

Tuning BEA JRockit JVM

To provide the optimal out-of-the-box experience, BEA JRockit JVM comes with default values that try to adapt automatically to the specific application you are running on which you are running BEA JRockit JVM. Tuning BEA JRockit JVM is accomplished by using extended options—or `-X` command line options that you enter at startup. The `-X` options are exclusive to BEA JRockit JVM and can differ greatly between JVMs on the market. Use them to set the behavior of BEA JRockit JVM to better suit the needs of your Java application.

This section describes how to use these options to tune BEA JRockit. It includes information on the following subjects:

- Setting the Heap Size
- Setting the Thread Stack Size

Note: If BEA JRockit behaves in some unexpected way, please consult the BEA JRockit Developers FAQ or you can also search for solutions to your problem in the BEA JRockit developer newsgroup.

Setting the Heap Size

System performance is greatly influenced by the size of the Java heap available to the JVM. This section describes the command line options you use to define the initial and maximum heap sizes and the size of any nursery that would be required by the generational garbage collectors. It also includes key guidelines for help you determine the optimal heap size for your BEA JRockit implementation.

Setting the Initial and Minimum Heap Size

```
-Xms<size>[k|K][m|M][g|G]
```

`-Xms` sets the initial and minimum size of the heap. For this, BEA recommends that you set it to the same size as the maximum heap size; for example:

```
-java -Xgcprio:throughput -Xmx:64m -Xms:64m myClass
```

Default

`-server` mode: will set the initial and minimal heap to 25% of the amount of free physical memory in the system, up to 64 MB and a minimum of 16 MB.

`-client` mode: will set the initial and minimal heap to 16 MB.

To improve start-up performance, set `-Xms` to *at least* the approximate amount of live data. If `-Xms` isn't set, or is set too low, frequent garbage collections can slow startup until JRockit has grown the heap.

To get a fixed heap size—for example, if you want a controlled environment—set `-Xms` and `-Xmx` to the same value.

Setting the Maximum Heap Size

```
-Xmx:<size>[k|K][m|M][g|G]
```

`-Xmx` sets the maximum size of the heap.

The default maximum heap size is a dynamic value determined by the amount of free physical memory in the system. If `-Xmx` is not set, the Java heap can grow up to the lesser of 75% of the total physical memory or 1 GB unless there is risk that growing the heap will cause paging. Paging may still occur, but will be avoided as much as possible.

Default

`-server` and `-client` modes: will set the total physical memory to the lesser of 75% of the total physical memory or 1 GB.

Setting a low maximum heap (`-Xmx`) compared to the amount of live data can affect performance by forcing JRockit to perform frequent garbage collections.

Encountering Out of Memory Errors

There are two types of out of memory errors in JRockit:

- The result of failed object allocation
- The result of failed allocation of native memory for internal use within JRockit

1. When the out of memory is caused by a failed object allocation, the maximum heap size (`-Xmx`) needs to be increased.
2. When the out of memory is caused by failed allocation of native memory, it means that the JRockit process uses too much memory in total. This limit is determined by the operating system. When this happens, it may help to decrease the maximum heap size (`-Xmx`) to free more memory resources for the rest of the JRockit process.

Paging (Page Faults)

Paging may cause severe performance problems for your application and long garbage collection pauses. To avoid paging, do not set `-Xmx` to more than 75% of the physical memory of the system. Also, remember to account for the memory usage of other applications intended to run simultaneously with JRockit, as these impact memory availability.

When verbose memory outputs are enabled (`-Xverbose:memory`) a warning will be printed when there are many page faults during garbage collection.

If the amount of free memory in the system varies widely, you might not want to set `-Xmx` at all. This will prevent JRockit from growing the heap when there is too little memory in the system. Be aware that this will throw an OutOfMemoryError if object allocation fails with the current heap size and the heap cannot grow without causing paging.

Paging may occur even if `-Xmx` isn't set. JRockit will not shrink the heap if more than half the heap is filled with live data. Thus, JRockit might not always be able to shrink the heap if the amount of free memory is reduced after JRockit has been started; for example, when another application is started.

Setting the Size of the Nursery

```
-Xns:<size>[k|K][m|M][g|G]
```

`-Xns` sets the size of the young generation (nursery) in generational garbage collectors. Optimally, you should try to make the nursery as large as possible while still keeping the garbage collection pause times acceptably low. This is particularly important if your application is creating a lot of temporary objects.

Note: To display pause times, include the option `-Xgcpause` when you start BEA JRockit JVM.

The maximum size of a nursery *cannot* exceed 95% of the maximum heap size.

Default

`-server` mode: the default nursery size is 10 MB per CPU; for example, the default for a 4-CPU system would be 40 MB.

`-client` mode: the default nursery size is 2 MB.

Additionally, the default nursery will never exceed 25% of maximum heap size, unless you use `-Xns` to explicitly set it to something larger.

Setting the Thread Stack Size

`-Xss<size>[k|K][m|M]` sets the thread stack size.

Minimum Thread Stack Size

Minimum thread stack size is 16KB. If `-Xss` is set below the minimum value, thread stack size will default to the minimum value automatically.

Default

If the thread stack size has not been set the default value depends on the platform on which BEA JRockit is running. Table 2-1 shows these defaults:

Table 2-1 Default Thread Stack Sizes

O/S	32-bit Default	64-bit Default
Windows	64 KB	320KB
Linux	128 KB	1 MB

Basic Tuning Tips and Techniques

When you install BEA JRockit JVM, it includes a host of default start-up options that ensure a satisfactory out-of-the-box experience; however, sometimes, these options might not provide your application with the optimal performance. Therefore, BEA JRockit JVM comes with numerous alternative options and algorithms to suit different applications. This section describes some of these options and some basic tuning techniques you can use at startup. You find information on the following subjects:

- Determine What You Want to Tune For
- Set the Heap Size
- Tune the JVM
- Analyze the Performance by Using the JRA

Determine What You Want to Tune For

Before you start BEA JRockit JVM, you need to determine these two factors:

- How much of your system memory do you prefer that the BEA JRockit JVM uses?
- What do you want from BEA JRockit JVM, the highest possible responsiveness or the highest possible performance?

Once you've answered these questions, use the information provided below to tune BEA JRockit JVM to achieve those goals.

Set the Heap Size

Generally, you want to set the maximum heap size as high as possible, but not so high that it causes page faults for the application or for some other application on the same computer. Heap sizing is accomplished by using the `-Xms` (minimum heap size) and `-Xmx` (maximum heap size) options. For details on these options and guidelines for sizing the heap, please refer to Setting the Heap Size.

Tune the JVM

As mentioned above, you need to consider how you want JRockit to perform: for the highest possible responsiveness or the highest possible performance? This section describes how to tune for either type of performance.

Tuning for High Responsiveness

If you want the highest responsiveness from your application and guarantee minimal pause times, do the following:

- Select a garbage collector that suits your application the best:
  - Use the default dynamic garbage collector (`-Xgcprio`) and set a pause time (`-Xgcprio:pausetime`).

    When you use the option `-Xgcprio:pausetime`, you can also set a target time for the pauses (`-Xpausetarget`). That way JRockit tries to adapt the pause times to the length that you specify. If you do not set the option to a specific time, the default value is used (500ms). This is an example of how you can set the pause target:

    ```
    -Xgcprio:pausetime -Xpausetarget=400ms
    ```

  OR

  - Use a fixed garbage collector, select the Generational Concurrent garbage collector (`-Xgc:gencon`).
- Set the initial (`-Xms`) and maximum (`-Xmx`) heap sizes, as described in Setting the Heap Size. If you're using a fixed, generational concurrent garbage collector, a larger heap reduces the frequency of garbage collection. This will prevent longer pauses.
- Set the size of the nursery (`-Xns`). If the application is creating a lot of temporary objects, you should use a large nursery, to reduce old collection frequency. Larger nurseries usually result in slightly longer pauses, so, while you should try to make the nursery as large as possible, don't make it so large that pause times become unacceptable.

You can see the nursery pause times in BEA JRockit JVM by starting the JVM with `-Xgcpause`.

Tuning for High Performance

If you want the highest possible performance BEA JRockit can provide, you will want to optimize for application thoughput. Set these tuning options at startup:

- Select a garbage collector that suits your application best:
  - Use the default dynamic garbage collector with the throughput priority specified (`-Xgcprio:throughput`)

  OR

  - Select the parallel (`-Xgc:parallel`) garbage collector. The parallel garbage collector doesn't use a nursery, so you don't need to set `-Xns`.
- Set the largest initial (`-Xms`) and maximum (`-Xmx`) heap sizes that your system can tolerate, as described in Set the Heap Size (not valid for the parallel garbage collector).

Analyze the Performance by Using the JRA

The JRockit Runtime Analyzer (JRA) is a great way to look at the performance of JRockit. The JRA records what happens in your system in runtime and then saves the findings in a file that can be analyzed through a separate JRA tool. The recording contains information about, for example, memory usage, Java heap content, and hot methods. For information on how to use the JRA, see:

Using the JRockit Runtime Analyzer

Index