Garbage Collection Guide for Jikes RVM

Contributors: Julia Chapple, Dallas Treder, Richard Thai
Last Updated: 11 May 2013
Tested for: Ubuntu 12.10 32-bit

Garbage Collector Readme

Resources

Mailing List

We highly recommend subscribing to the Jikes RVM community (https://lists.sourceforge.net/lists/listinfo/jikesrvm-researchers) to have an idea of how open and responsive the community is to helping out beginners. Be sure to follow good email etiquette and researching your problem a bit before mailing them your issues.

Jikes RVM API

http://jikesrvm.sourceforge.net/apidocs/latest/

The API will speed up your understanding of the different and classes and dependencies (versus looking through the code and tracing by hand).

Care and Feeding

http://jikesrvm.org/Care+and+Feeding

This page guides the installation and configuration of Jikes RVM. Our Installing Jikes RVM section supplements these guides.

Jikes Architecture

http://jikesrvm.org/Architecture

The Architecture guide describes the different components of Jikes and their general implementations. Of particular interest is the Memory Manager Toolkit (MMTk) which controls garbage collection.

MMTk Tutorial

http://jikesrvm.org/MMTk+Tutorial

The above is a tutorial on creating a new collector in Jikes RVM. It is not how to create a collector from scratch (this approach is highly discouraged) but rather a guide on copying an existing collector and making some modifications. The tutorial is somewhat out of data and has many known issues. As you follow it please see the Troubleshooting section for update and fixes.

The Anatomy of a Garbage Collector

http://docs.codehaus.org/display/RVM/Anatomy+of+a+Garbage+Collector

This document provides a good high level discussion of the structure of garbage collectors in Jikes. This document is discussed and expanded upon in the Exploring Jikes RVM section.

Mark Sweep implementation in Jikes RVM

http://cs.anu.edu.au/~Robin.Garner/pf-ismm-2007.pdf

The white paper provided explores optimizations made to the mark sweep collector implemented in Jikes RVM. In particular, section 4 of the white paper explores the bitmap marking mechanism that is in use.

Troubleshooting the RVM

The following troubleshooting steps are resolutions to problems we encountered during our use of Jikes RVM. They are intended as possible solutions to some problems that may arise during installation and use of Jikes, and while we do suggest looking through here first in case we ran into your problem and have done the needed research to fix it, be prepared to google and ask the mailing list (https://lists.sourceforge.net/lists/listinfo/jikesrvm-researchers) for help for problems you run into.

Running a Program Issues

1. If you get an error saying how java cannot be located and that it's located somewhere in "usr/bin/java/bin/java", then go to your .bashrc or .bash_profile file and add the line "export JAVA_HOME=/usr/" at the end and delete any other export command.
2. If something says that a java thing, "java blah", wasn't found, and you're pretty sure that you have java, then there's some config file somewhere pointing to the wrong location. If you want to know the location of your java thing, here's a good way to find where it may be located:
   $ whereis java
   though you may need something more specific depending on what is missing. Once you figure out where the correct location is, run this command from your current directory to figure out where this incorrect directory for "java blah" is hardcoded:
   $ find . -exec grep -l "java blah" {} \;
   You'll get mostly directories, but you should find a file or a couple that defined this (most likely in an obscure properties file). Go to it, and change that line to the proper java location.
1. You must be able to **run and compile** a Java program. Since you'll likely need to switch java versions (*update-alternative*), then your compile + run for a test program should go along the lines of
   $ javac foo.java
   $ java foo
   of which you may receive an error similar to
   Exception in thread "main" java.lang.UnsupportedClassVersionError: Main : Unsupported major.minor version 51.0.
   The key thing to notice is the error*: java.lang.UnsupportedClassVersionError.*
   What this means is that there's a higher JDK during compile time and lower JDK during runtime. *update-alternative* affected the runtime JDK, but javac is still working with a newer version of java. To remedy this, you must specify source and target versions of java when compiling like so (if Java 6 is being used):
   $ javac -source 1.6 -target 1.6 foo.java

Tutorial Erratum

Fortunately, most of the tutorial is still pretty useful, but there do exist sections that are inaccurate and out of date. Most of these issues are in the main part of the tutorial, Building the Mark-Sweep Collector.

Free-List Allocation

1. In the first step: in TutorialConstraints the imports "import org.mmtk.policy.SegregatedFreeListSpace;" and "import org.mmtk.policy.MarkSweepSpace;" must be added. If there are still other import errors, copy over more imports from the marksweep collector (plan/marksweep)
2. In the second step, DEFAULT_POLL_FREQUENCY is never defined and should be omitted from the constructor.

Mark-Sweep Collection

1. In the second step, there is no need to add the getPagesRequired method.

Hybrid Collector

1. There is no need to add the precopyObject method.

Miscellaneous Issues

1. Assuming that you're connected to a VM via ssh, Eclipse has a bug when trying to use a remote connection to use the rvm. To run Eclipse over a remote connection, run the following command (see https://bugs.eclipse.org/bugs/show_bug.cgi?id=386955 for details):

$ eclipse -vmargs -Dorg.eclipse.swt.internal.gtk.cairoGraphics=false

1. Renaming or moving your Jikes directory can cause problems when building Jikes. If you cannot build after doing this try running "ant cleanest" to reset ant so it refers to the right files.

Installing Jikes RVM

This guide is written under the assumption that you are connecting (via ssh) to a VM with Ubuntu 12.10 32-bit. Do not use 64-bit; JikesRVM currently does not work properly with 64-bit. If you're connecting from Windows, we recommend using PuTTY (www.putty.org) to connect via ssh and Xming (http://sourceforge.net/projects/xming) as your X Server for windows (this will enable GUI's from your VM to appear on your screen). When using PuTTY, be sure to enable X11 forwarding by checking the box in ***Connection->SSH->X11->Enable X11 forwarding***.

The first step to running the garbage collector is installing the required dependencies. If you run into any issues during this install process, or later on during the use of it, refer to the troubleshooting section at the end of this guide. To **install the dependencies**, run the following command:
$ sudo apt-get install mercurial ant gcc g++ gcc-multilib g++-multilib openjdk-6-jdk bison -y

After the dependencies are installed, make sure that you are using the **correct version of Java** (Jikes RVM requires Java 6). To do so, run the following command to bring up a menu with options for versions of java installed on your machine:
$ sudo update-alternatives --config java

Once all of the dependencies have been satisfied, you can **download the virtual machine** itself. Follow the guide on Jikes' website: http://jikesrvm.org/Get+The+Source. Next, go to the folder where you downloaded the virtual machine and create a file named .ant.properties, and add a line in it referencing the type of system you are building on; for example, in our system, the line was:

1. host.name=ia32-linux

You can find the other options for host name and many more things to specify from this guide: http://jikesrvm.org/Building+the+RVM

Finally, you should build and test the rvm. To build the rvm, run the following command from Jikes' root directory:
$ ./bin/buildit localhost -t gctest BaseBaseMarkSweep
This should build and test a simple mark sweep collector. To build other collectors, simply change the "BaseBaseMarkSweep" to a different plan (to be explained later). To run a different test, just change "gctest". To build without testing, remove the -t flag and the test name.

If the build and tests run successfully, you should have a newly built virtual machine, contained in a directory under the dist directory. Simply use the rvm binary in that folder in the same way you would use Java to run programs (i.e., ./dist/{plan_name}_{os_name}/rvm foo, assuming foo is a compiled Java program in Jikes' root directory).

Exploring Jikes RVM

Introduction

Jikes RVM is a complete virtual machine with its own runtime service, compilers and memory managers. Garbage collection is only a small part of the RVM. Still the MMTk component of Jikes is very large and complex. A good understanding of it's structure is needed before it can be used to create new garbage collectors.

The first thing to read to get a good introduction to the MMTk (Memory Management Toolkit) is the Anatomy of a Garbage Collector page created by JikesRVM (currently at http://docs.codehaus.org/display/RVM/Anatomy+of+a+Garbage+Collector). This provides a high-level overview of the different parts of the MMTk, and what each does to contribute to garbage collection.

A quick explanation of the information on that page: a garbage collector is split into two main components, a plan and a policy. All garbage collectors, whether simple or complex, have support for both global and thread-local collection, and thus multi-thread support. This means that the plan and the policy are both split into several classes, some containing methods and data for the entire area of the heap being managed, and others that refer to only certain sections and are local to one thread only. This means that each mutator thread has its own resources and its own allocator and collector threads. Almost all of the relevant code for the garbage collectors is in the org.mmtk package, and so any references to code should be assumed to be in that package.

The policy describes how an area of memory is managed, and the plan contains data structures and configuration information for the specific garbage collector. A policy has both a global class (a "space"), that eventually inherits from *Policy.Space*, and a thread-local class that inherits from *utility.alloc.Allocator*, and is designed to hold a buffer for allocation, and get another buffer from the global class when that runs out.

The plan is used to contain data about the garbage collector and to organize the actual collection of the heap. The plan is split into five classes, each of which starts a given prefix for the collection algorithm:

1. A subclass of *plan.Plan*. This holds the global data structures.
2. A subclass of *plan.MutatorContext*. This holds data local to a mutator thread (like an instance of the thread-local policy class, for allocation).
3. A subclass of *plan.CollectorContext*. This holds data local to a collector thread.
4. A subclass of *plan.PlanConstraints*. This holds configuration information for the VM.
5. A subclass of *plan.TraceLocal*. This hold data for a specific way to trace the heap. There may be more than one of these if there is more than one way to trace the heap in the collector (e.g., generational collector).

In general, the collector is executed in phases. There are many phases (defined in plan.Simple), which can be called on either the global or thread-local classes, and are executed in the collectionPhase methods of the appropriate classes.

Example Collector

Introduction

This section takes the theoretical Mark-Sweep garbage collector from the Garbage Collection Handbook, and explains how the different sections map to the different parts of the implementation in MMTk. One thing to note is that there is quite a bit of additional other things going on at the same time as any collector; all collectors implemented in the MMTk have, in addition to the spaces required for the collector itself, an immortal space for objects that will not be collected, a large object space for objects large enough that they might need to be dealt with not through the usual collector, and possibly others.

Marking

The first part of marking takes place in the STACK_ROOTS and ROOTS phases, where plan.SimpleCollector has the VM get the roots of each thread and add them to a buffer (this is implemented in plan.SimpleCollector and not one of the Mark-Sweep specific classes because getting the roots of the mutators is a generic action. Once the roots have been added to a buffer, the CLOSURE phase has plan.marksweep.MSCollector call completeTrace on plan.marksweep.MSTraceLocal, which runs through all of the roots added, and does a transitive closure on them to mark the entire heap

Sweeping

Once the thread-local tracing has been completed, the entire space can be swept. This is done in the RELEASE phase. plan.marksweep.MS calls the space's release method, which sweeps through the blocks of different sizes in the space, looking for objects that have not been marked, so that it can free that space.

Optimizations

The collectors in Jikes RVM are very thoroughly implemented and often have many optimization present. Mark-Sweep in particular has bitmap marking and lazy sweeping available as options. However, the implementations of these optimizations are not easy to find and don't look like their high level descriptions.

Bitmap marking is controlled with the HEADER_MARK_BITS flag. It is used in MarkSweepSpace. It is used in advanceToBlock to control whether an accessed memory block is markedin the bitmap. It is also used in prepare where marks are used or cleared.

Although lazy sweeping is unique to Mark Sweep Collection the LAZY_SWEEP flag is defined at a very high level in SegregatedFreeListSpace. Lazy sweeping affects how sweeping proceeds in the sweepBlock method. In the standard case the entire heap is processing in sweeping. The mark sweep flag limits the sweeping so that it only sweeps until enough free space is found. The flag also used in the advanceToBlock method in MarkSweepSpace where it limits the sweeping step so it is not all done at once.