Energy consumption of Garbage Collectors

Auteur :
Anis TELLO

18th January 2016

Table of contents

Introduction

"Every program is guilty until proven innocent".

During the development phase, developers spend lots of time writing tests for code they have written. But these tests can be very so tedious and repetitive that developers sometimes botch or simply skip this very important part of software development.

But what if developers didn't have to spend hours writing tests to find bugs and have a good code coverage of their program? What if there was a magic stick that can generate a bunch of unit tests?

Since Java is one of the most used programming languages nowadays, we implemented a solution that would generate automatically Java unit tests. This solution would save time for developers, and give the time and the energy to focus on working on the business layer.

This project is an extended version of an application developed by Valentin Lefils and Quentin Marrecau [**?**] [**?**] . The first version of the application treated the same problems we are facing, but only on small examples of Java programs.

In this project, we present our tool: JUnitMe2.0. Our tool can generate automatically Java unit tests for any Java open source application. From a description of specification, our tool generate all instances that covere the data specifications, then generate the unit tests corresponding to these instances.

Chapter 1

Technical work

1.1 Goal

The goal of this study is to estimate the energetic consumption of different Garbage collection(GC)[1] mechanism in Java Virtual machine (JVM)[2].

1.2 Garbage Collector

Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

In a programming language like C, allocating and deallocating memory is a manual process. In Java, process of deallocating memory is handled automatically by the garbage collector.[3] There are 4 different types of garbage collectors available for Java

1.2.1 Serial Garbage Collector

The serial collector is the default for client style machines in Java SE 5 and 6. With the serial collector, both minor and major garbage collections are done serially (using a single virtual CPU). In addition, it uses a mark-compact collection method. This method moves older memory to the beginning of the heap so that new memory allocations are made into a single continuous chunk of memory at the end of the heap. This compacting of memory makes it faster to allocate new chunks of memory to the heap.[3]

Usage Cases The Serial GC is the garbage collector of choice for most applications that do not have low pause time requirements and run on client-style machines. It takes advantage of only a single virtual processor for garbage collection work (therefore, its name). Still, on today's hardware, the Serial GC can efficiently manage a lot of non-trivial applications with a few hundred MBs of Java heap, with relatively short worst-case pauses (around a couple of seconds for full garbage collections).

Another popular use for the Serial GC is in environments where a high number of JVMs are run on the same machine (in some cases, more JVMs than available processors!). In such environments when a JVM does a garbage collection it is better to use only one processor to minimize the interference on the remaining JVMs, even if the garbage collection might last longer. And the Serial GC fits this trade-off nicely.

Finally, with the proliferation of embedded hardware with minimal memory and few cores, the Serial GC could make a comeback.[3]

Command Line Switch To enable the Serial Collector we can use: *-XX:+UseSerialGC*

1.2.2 Parallel Garbage Collector

The parallel garbage collector uses multiple threads to perform the young genertion garbage collection. By default on a host with N CPUs, the parallel garbage collector uses N garbage collector threads in the collection. The number of garbage collector threads can be controlled with command-line options: *-XX:ParallelGCThreads=¡desired number¿*

On a host with a single CPU the default garbage collector is used even if the parallel garbage collector has been requested. On a host with two CPUs the parallel garbage collector generally performs as well as the default garbage collector and a reduction in the young generationgarbage collector pause times can be expected on hosts with more than two CPUs. The Parallel GC comes in two flavors.[3]

Usage Cases The Parallel collector is also called a throughput collector. Since it can use multilple CPUs to speed up application throughput. This collector should be used when a lot of work need to be done and long pauses are acceptable. For example, batch processing like printing reports or bills or performing a large number of database queries.[3]

The Parallel collector is also called a throughput collector. Since it can use multilple CPUs to speed up application throughput. This collector should be used when a lot of work need to be done and long pauses are acceptable. For example, batch processing like printing

reports or bills or performing a large number of database queries.[3] To enable the Parallel collector we can use: *-XX:+UseParallelGC*

Parallel Old Collector To enable this garbage collector we use *-XX:+UseParallelOldGC*. With this option, the GC is both a multithreaded young generation collector and multithreaded old generation collector. It is also a multithreaded compacting collector. HotSpot does compaction only in the old generation. Young generation in HotSpot is considered a copy collector; therefore, there is no need for compaction.

Compacting describes the act of moving objects in a way that there are no holes between objects. After a garbage collection sweep, there may be holes left between live objects. Compacting moves objects so that there are no remaining holes. It is possible that a garbage collector be a non-compacting collector. Therefore, the difference between a parallel collector and a parallel compacting collector could be the latter compacts the space after a garbage collection sweep. The former would not.[3]

1.2.3 Concurrent Mark Sweep (CMS) Collector

The Concurrent Mark Sweep (CMS) collector (also referred to as the concurrent low pause collector) collects the tenured generation. It attempts to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads. Normally the concurrent low pause collector does not copy or compact the live objects. A garbage collection is done without moving the live objects. If fragmentation becomes a problem, allocate a larger heap.

Note: CMS collector on young generation uses the same algorithm as that of the parallel collector.[3]

Usage Cases The CMS collector should be used for applications that require low pause times and can share resources with the garbage collector. Examples include desktop UI application that respond to events, a webserver responding to a request or a database responding to queries.[3]

Command Line Switch To enable the Concurrent Mark Sweep (CMS) collector we can use: *-XX:+UseConcMarkSweepGC*
And to set the number of threads use: *-XX:ParallelCMSThreads=¡n¿*

1.2.4 G1 Garbage Collector

The Garbage First or G1 garbage collector is available in Java 7 and is designed to be the long term replacement for the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector that has quite a different layout from the other garbage collectors described previously.[3]

Command Line Switch To enable the Concurrent Mark Sweep (CMS) collector we can use: *-XX:+UseG1GC*

1.3 Implementation

Since we know that objects that are not referenced will be removed from memory by the garbage collector. And in order to realise this study I have modified an open source application[4] which test garbage collection behavior so the application would generate a larg number of objects and then remove the reference pointing to these objects.
Each minute a separate thread calls System.gc() to encourage JVM to run the garbage collector. Using PowerAPI[5]: "wattmeter software based on a model of actors, en uses this model to estimate software energetic consumption".[6] I was able to measure the consumption energetic of this application running each time a different mechanism of garbage collection.

Chapter 2

Evaluation

2.1 Performance

2.2 Validation

2.3 Limitations

Conclusion

Using Spoon Java library to analyze and trasform source code, Alloy a language and tool for relational models, Alloy Analyzer a solver that takes the constraints of a model and finds structures that satisfy them and CodeModel a Java library for code generators we have succeeded in creating a tool capable of generating Java unit tests a given Java program. Our tool can verify that there no actual error exists for an application. This tool can be extended in the future to be able to treat a bigger variety of Java programs. Today, our tool has been tested on couple of Java open source project and it is able to generate Java unit tests that can achieve up to 88% of code coverage.

Bibliography

[1] Wikipedia. Garbage collection. https://en.wikipedia.org/wiki/Garbage_collection_(computer_science).

[2] Wikipedia. Java virtual machine. https://en.wikipedia.org/wiki/Java_virtual_machine.

[3] Oracle. Java garbage collection. http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html.

[4] herongyang.com. Testing garbage collection behavior. http://www.herongyang.com/JVM/GC-Garbage-Collection-Test-Program.html.

[5] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. Prspoonocess-level power estimation in vm-based systems. https://hal.inria.fr/hal-01130030/file/paper.pdf.

[6] Maxime Colmant, Romain Rouvoy, and Lionel Seinturier. Prspoonocess-level power estimation in vm-based systems. https://hal.inria.fr/hal-01171696/document.