Energy consumption of Garbage Collectors

Author :
Anis TELLO
https://github.com/AnisTello/GreenComputing

18th JANUARY 2016

Table of contents

Introduction

Our computers and smartphones might seem clean, but the digital economy uses a tenth of the world's electricity — and that share will only increase, with serious consequences for the economy and the environment.[1]
As concerns about global energy consumption increase, power consumption of Hardwares, softwares and even the Internet is a matter of importance.

Since Java is all always on the top of IEEE Spectrum ranking list as the most popular programming language[2]. And since freeing unused memory is an essential part of any application, I have decided to focus in this Study on measuring the energy consumption of Garabge Collector[3].

Chapter 1

Technical work

1.1 Goal

The goal of this study is to estimate the energy consumption of different Garbage collection(GC) mechanism in Java Virtual machine (JVM)[4].

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.[5] 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.[5]

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.[5]

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.[5]

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.[5]

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.[5] 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.[5]

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.[5]

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.[5]

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.[5]

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[6] which test garbage collection behavior so the application now 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.
To measure the energy consumption of this Java application I used **PowerAPI**[7] : Wattmeter software based on a model of actors. we use it to estimate software energetic consumption.[8]
Each time I run the application I use a different mechanism of garbage collection. So powerAPI tells for each execution the energy consumption of the application using a specific mechanism of garbage collection.

Chapter 2

Evaluation

2.1 Validation

In order to have a valid result, we had to execute the application for each Garbage collection mechanism at least for 5 minutes.
PowerAPI generates an output file with the informations related to the execution (Process ID, duration, energy consumption. . .)
An automatic script treats an output file was executed for each output file to generate a modified file with only the extraction of the value of energy consumption.
these modified files were passed to Gnuplot[9] : a command-line program that can generate two -and three-dimensional plots of functions, data, and data fits.[10]

2.1.1 Results

Using Gnuplot I was able to generate a graph representing the energy consumption for each Garbage collection in time:

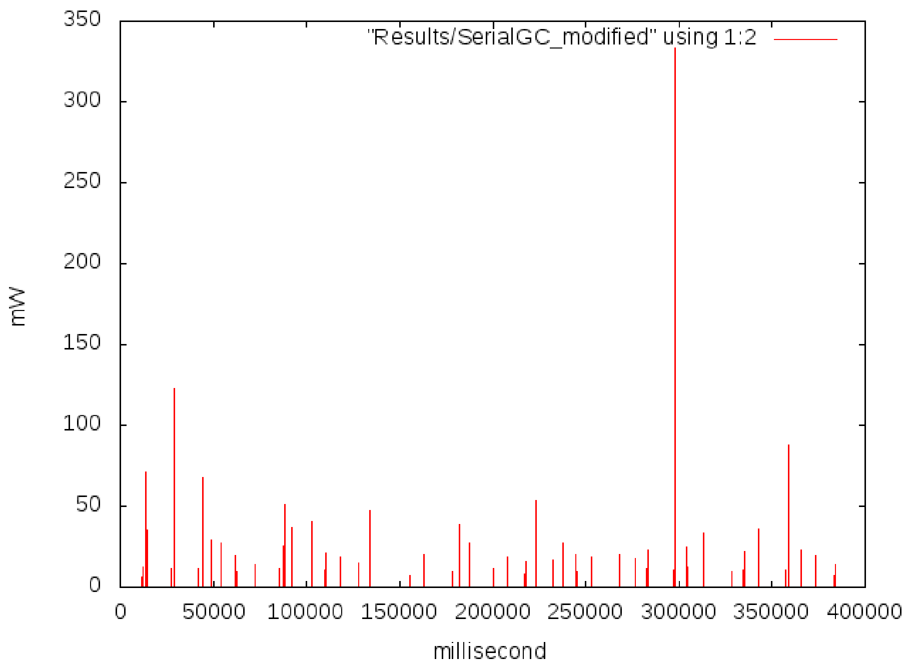

Figure 2.1: Serial Garbage collector

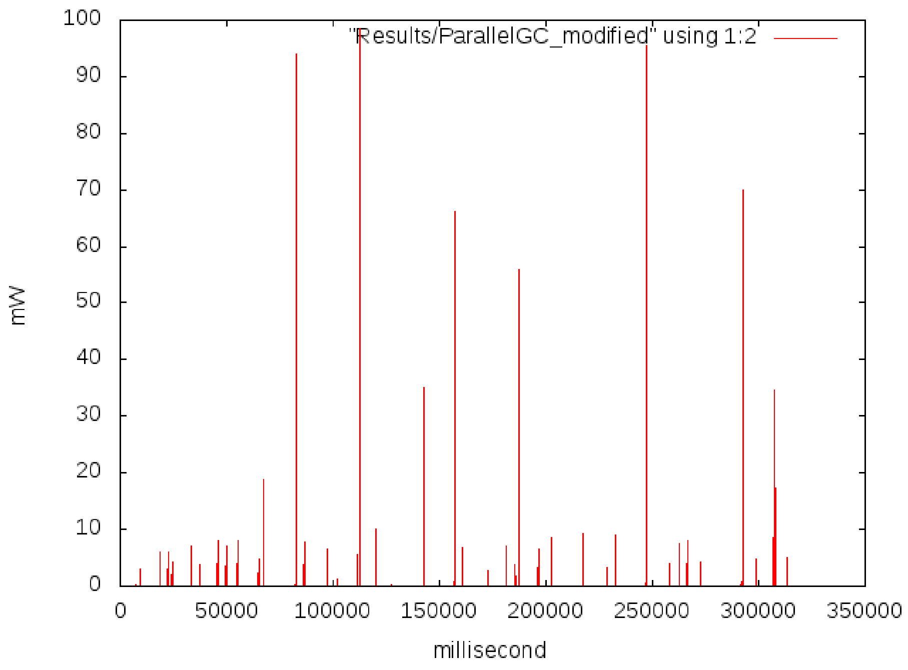

Figure 2.2: Parallel Garbage collector

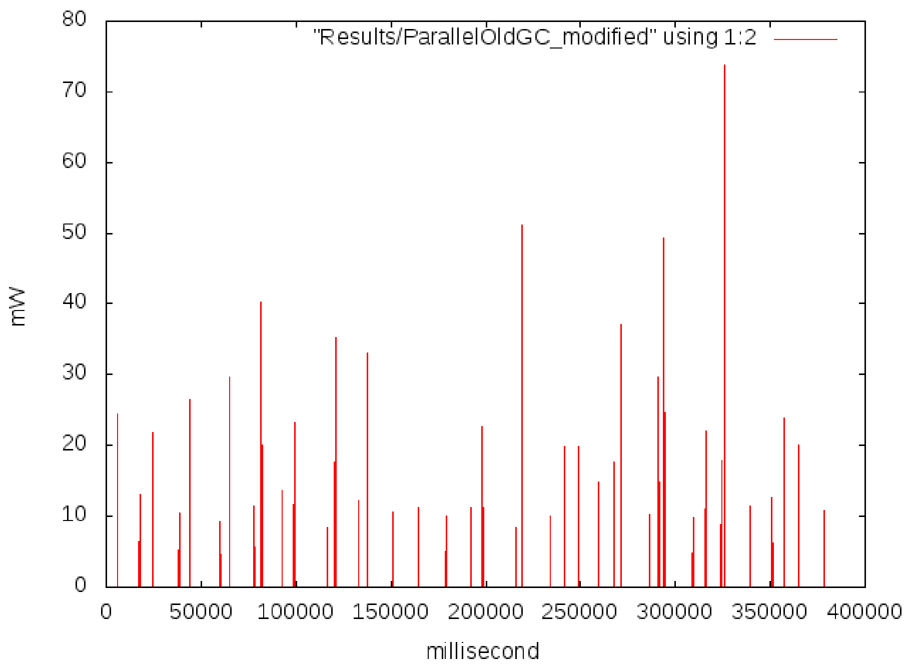

Figure 2.3: ParallelOld Garbage collector

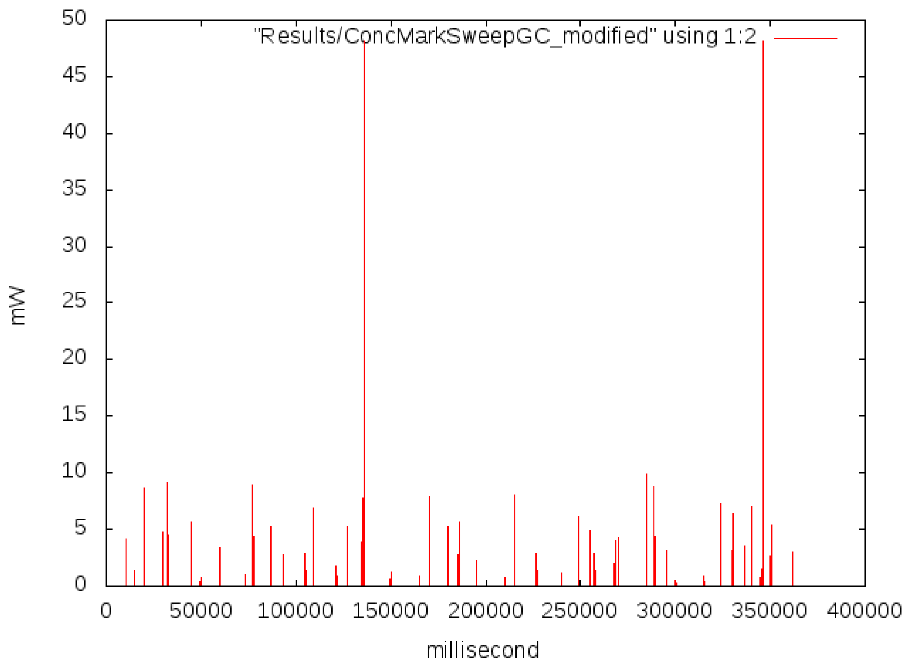

Figure 2.4: ConcMarkSweep Garbage collector

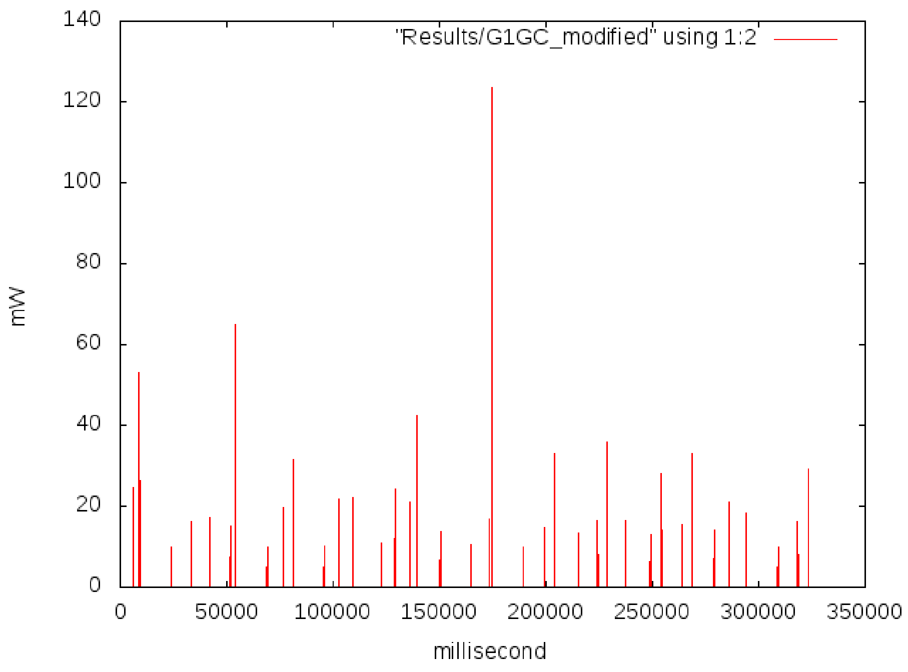

Figure 2.5: G1 Garbage collector

As we can see from the graph presented above, the average consumption for Serial Garbage collector is around 34.7mW while the average consumption for Parallel is around 16.2mW and for ParallelOld is around 21.2mW.
For ConcMarkSweep Garbage collector the consumption is around 6.3mW while for G1 Garbage collector it is arround 24.1mW.
We Can conclude that Serial Garbage collector is the most consuming Garbage collector and ConcMarkSweep is the least consuming Garbage collector.
As we can see, none of the four Garbage collector consumes a lot of energy, so we can safely say that the choice of the garbage collector for a Java application won't play a significant role in determining the energy consumption of an application.

2.2 Limitations

This study has been done on a small scale program. The results obtained may not be the same on a real world application. A further analyzes can be done in real world cases where data can be measured during days not only several minutes.

Conclusion

In this study I succeeded in showing the energy consumption of different Garbage Collection mechanism in Java Virtual machine using PowerAPI a tool for measuring the energy consumption of softwares running on a machine.
This study shows that Serial Garbage collector is the most consuming Garbage collectors while ConcMarkSweep is the least. This study can be extended in the future to be cover real world programs on large period of times.

Bibliography

[1] TIME. The surprisingly large energy footprint of the digital economy. http://science.time.com/2013/08/14/power-drain-the-digital-cloud-is-using-more-energy-than-you-think/.

[2] IEEE SPECTRUM. The 2015 top ten programming languages. http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages.

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

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

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

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

[7] 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.

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

[9] Gnuplot. Gnuplot official website. http://www.gnuplot.info/.

[10] Wikipedia. Gnuplot. https://en.wikipedia.org/wiki/Gnuplot.