Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 134 (2018) 334–339

Procedia Computer Science

www.elsevier.com/locate/procedia

The 2nd International Workshop on Big Data and Networks Technologies (BDNT'2018)

# Analysis of the allocation of classes, threads and CPU used in embedded systems for Java applications

Hicham MEDROMI[a],Laila MOUSSAID[b],Laila FAL[a,b,*]

[a]*Systems Architecture Hassan II University, ENSEM Casablanca, Morocco*
[b]*Systems Architecture Hassan II University, ENSEM Casablanca, Morocco*
[a,b]*Systems Architecture Hassan II University, ENSEM Casablanca, Morocco*

[a]*hmedromi@yahoo.fr* , [b]*moussaid@univcasa.ma* , [a,b] *lailafal.ensem@gmail.com*

**Abstract**

In order to analyze the volume of classes and threads allocated by Java applications, as well as the CPU used in embedded systems, we performed a comparative study to follow the behavior of several simple applications of different categories and in two VMs. different. After collecting data through monitoring tools, we found that with the VMZ more classes and threads are created, and more CPU is consumed, so more collections are made to free up memory space. This is due to the fact that VMZ is based on Oracle's Open JDK, it takes a little more memory and CPU to load all the libraries used.

*Keywords:* Virtual Machine Zulu Azul (VMZ), Virtual Machine Oracle(VMO), embedded systems, Garbage Collector(GC), eclipse, memory management , java micro edition,

## 1. Introduction

This article presents in a first part how is managed the memory in java via the GC contrary to other programming languages. The second part is devoted to a detailed comparative study of the allocation of classes and threads as well as the CPU used during the execution of the application. The comparison is done on both VMO and VMZ for micro edition systems. Finally, a conclusive synthesis following this comparison with perspectives will conclude the article.

* Corresponding author. Tel.: +212-604-246-908
*E-mail address:* lailafal.ensem@gmail.com

## 2. Notions

### *2.1. Memory management*

Since the memory is automatically managed by the runtime environment, the programmer is freed from this task, which is the source of many errors that are difficult to find. Manual memory management is one of the most common sources of error [11].

Three main types of errors can occur:

a. Access to an unallocated area, or that has been released,
b. The liberation of an already liberated zone,
c. The non-release of unused memory (memory leaks)

If appropriate tools and methodology can reduce its impact, the use of a garbage collector eliminates them almost completely (memory leaks remain possible, especially during collision exceptions, but rarer). This simplification of the programming work pays a few disadvantages, such as sudden slowing down when the operation is triggered, which makes this technique inappropriate for real time [11].
Since manual memory management can be difficult, other languages such as Java and Ruby have chosen to use a garbage collector (GC).

### *2.2. Garbage Collection*

A GC is responsible for allocating memory, ensuring that all referenced objects remain in memory, and recovering the memory used by objects that are no longer accessible from the references during code execution. The referenced objects are called alive, those that are no longer referenced are considered dead and are called Garbage. The process of finding and releasing the space used by these objects is Garbage collection.

Garbage Collection solves many memory allocation issues, but not all. For example, you could create objects indefinitely and continue to reference them until there is no more available memory. Collection is also a complex task taking time and resources. The precise algorithm used to organize memory and allocate and free space is managed by the GC and hidden programmer. Space is usually allocated from a large pool of memory referred to as the heap.

The time of collection belongs to the garbage collector. As a general rule, the entire pile or part of it is collected either when it is filling up or when it reaches a threshold of occupation.
The task of completing an allocation request, which involves finding an unused block of memory of a certain size in the heap, is difficult. The main problem for most dynamic memory allocation algorithms is to avoid fragmentation while maintaining efficient allocation and release.

### *2.3. Java Micro Edition [9] :*

The Java Micro Edition1 specification was born from Sun's desire to adapt Java technology to embedded devices that do not have enough resources to offer Java Standard compatibility. To best match the different families of equipment, the specification distinguishes different configurations, themselves cut into profiles. The two configurations used in practice are CDC (Connected Device Configuration), which is aimed at fixed equipment, such as digital television set-top boxes, and CLDC (for Connected Limited Device Configuration), which targets mobile equipment such as mobile phones. Each of these configurations corresponds to a certain virtual machine as well as some standard library. CDC requires a complete virtual machine, and offers a fairly rich API, in which the different profiles essentially correspond to the presence or absence of a graphical interface. CLDC, on the other hand, is intended for more constrained equipment, and offers only a restricted version of the language. To simplify the implementation of the virtual machine, reflexivity, finalization, floating point numbers, and custom class loaders are eliminated. A large part of the standard APIs are also deleted, in favor of profiles specific to each type of application.

For example, the most used profile is the MIDP (Mobile Information Device Profile) which gives access to the screen of the mobile phone, as well as some network connectivity (HTTP requests).

## 3. Realization

To make this study, we used two virtual java machines VMO and VMZ for micro systems. The tests were carried out on several simple applications, each application is launched on both VMs and with the same conditions.

The first step is to collect the information based on supervision tools (VisualVM and others), the second step will be to summarize these data and convert them into graphs in order to come out with a more explicit synthesis.

### *3.1. Application overview*

We have used several applications [12] mobile java of different categories:

Table 1. Java applications.

| Category | Application | Description |
|---|---|---|
| Medecine | First Aid App 1.0 | This app provides access to first aid information, it provides the way to deal with some emergency situations |
| Games | vPoker | This is an open source video poker game for compatible mobile phones MIDP1.0 J2ME. Designed for a minimum screen size of 128x128 pixels. It was updated to v1.2: The account is saved when you close the game and restore it when you start again. The statistics have been added. Many other improvements. |
| System utility | Zip Utility | This software allows you to compress and extract zip files on your mobile phone itself. Created using LWUIT, you have a better visual experience with animations. You can increase the speed by turning off transitions and changing the theme to low resolution. |
| Communication | Skype (officiel) 1 | Official Skype for mobile, communication software. This version is supported by most Nokia, Sony Ericsson, Motorola, Samsung and LG phones |
| Multimedia | KM Player 1.0 | Convenient audio and video player from VCD, Ogg, DVD, AVI, MKV, 3GP, WMV, MPEG-1/2/4, QuickTime and RealMedia formats - KM Player |
| Entertainment | X Ray Scanner Best 1.0 | This app uses your built-in camera to generate infrared X-rays |
| Navigation | Mobile GMaps 1.0 | Mobile GMaps is a free application that displays maps from various sources such as Yahoo! Maps, Windows Live Local, Ask.com and Open Street Map on Symbian and Java J2ME-enabled mobile phones, PDAs and other devices |
| Science and education | Calculatrice scientifique | Scientific calculator that calculates the value of complex expressions directly from your mobile phone. It comes with all the functions you could want from a mobile calculator app - trigonometric functions, degrees / radians, exponential, logarithm etc. |

## 4. Experimental results

The figures below were taken on a run time of 30seconds.

Table 2. Résultats of VMO.

| Application | Classes | Number of Threads | CPU (%) |
|---|---|---|---|
| First Aid App 1.0 | 3158 | 37 | 1,9 |
| vPoker | 3082 | 37 | 0,3 |
| Zip Utility | 3141 | 37 | 0,3 |
| Skype (officiel) 1 | 3168 | 35 | 1,5 |
| KM Player 1.0 | 3605 | 44 | 1,1 |
| X Ray Scanner Best 1.0 | 3095 | 28 | 0,3 |
| Mobile GMaps 1.0 | 3181 | 44 | 0,3 |
| Calculatrice scientifique | 3079 | 35 | 0,0 |

Table 3. Résultats of VMZ.

| Application | Classes | Number of Threads | CPU (%) |
|---|---|---|---|
| First Aid App 1.0 | 3766 | 39 | 2,7 |
| vPoker | 3803 | 46 | 0,7 |
| Zip Utility | 3655 | 40 | 0,4 |
| Skype (officiel) 1 | 3894 | 38 | 1,5 |
| KM Player 1.0 | 3604 | 54 | 2,7 |
| X Ray Scanner Best 1.0 | 3821 | 48 | 0,7 |
| Mobile GMaps 1.0 | 3906 | 46 | 0,3 |
| Calculatrice scientifique | 3806 | 38 | 0,7 |

By combining the two results obtained:

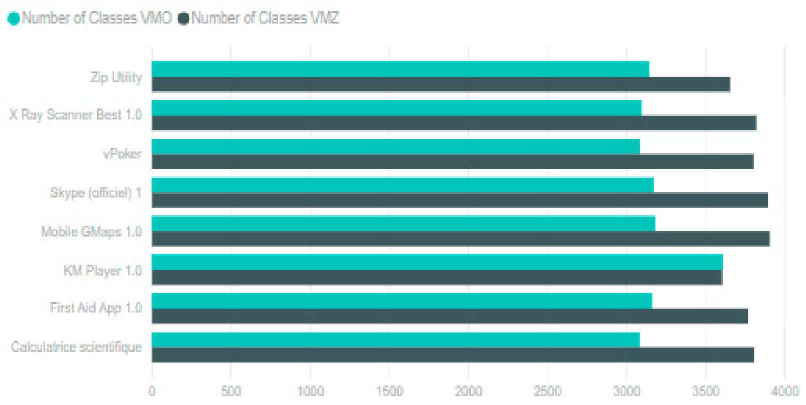


Fig. 1: Number of classes allocated with VMO and VMZ

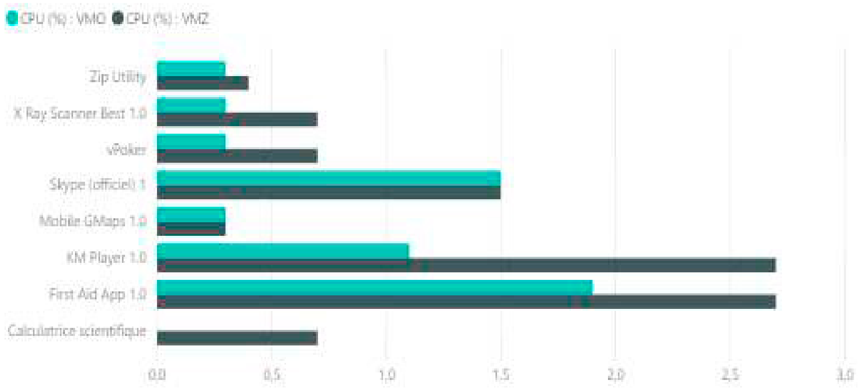


Fig. 2: CPU (%) with VMO and VMZ

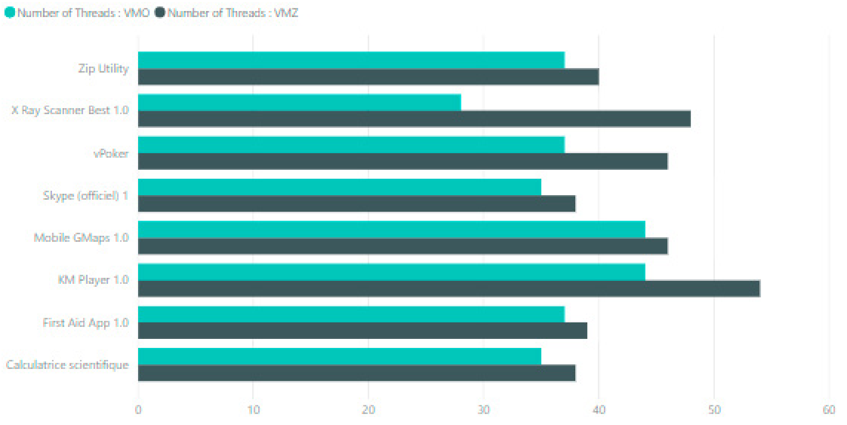
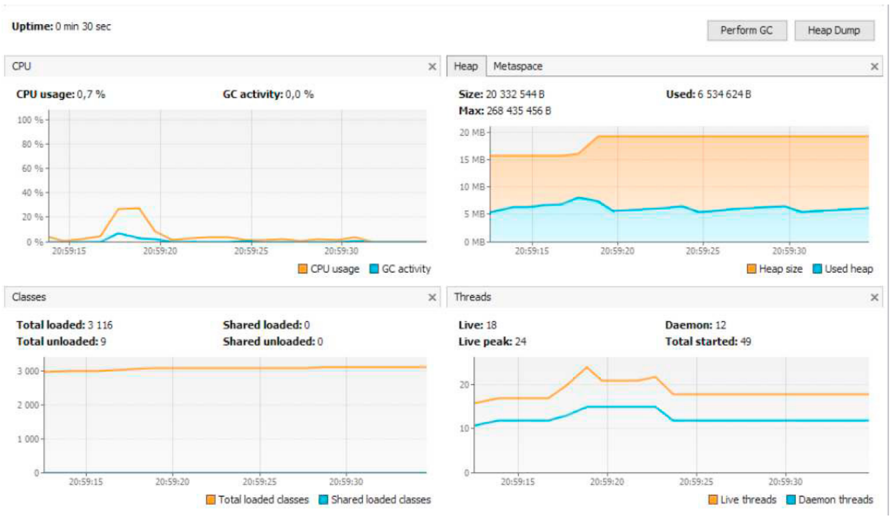


Fig. 3: Number of Threads with VMO and VMZ

## Example of monitoring :

Below are two screenshots taken after 30 seconds of running the application :



**Uptime:** 0 min 30 sec

**CPU usage:** 0,7 % **GC activity:** 0,0 %

**Size:** 20 332 544 B **Used:** 6 534 624 B
**Max:** 268 435 456 B

**Total loaded:** 3 116 **Shared loaded:** 0
**Total unloaded:** 9 **Shared unloaded:** 0

**Live:** 18 **Daemon:** 12
**Live peak:** 24 **Total started:** 49

Fig. 4: Monitoring the KM Player 1.0 application with VMO

## 5. Discussion :

Looking at the summarization tables and graphs, we will see that with the VMZ more classes and threads are created, and more CPU is consumed, so more collections are made to free up memory space.
This is due to the fact that VMZ is 100% based on Oracle Open JDK, so it needs to allocate more CPU memory to load Oracle libraries in addition to its own libraries. Most VMs are based on the oracle JDK, so before choosing a VM, you must take into account the performance of the micro system in question, the need and the java application of the user.
To free up memory, several GCs are available, the type of GC used can also be modified via commands depending on the application.

There are no ideal GCs, the performance of the garbage collector is closely linked to the size of the memory it has to manage. Thus, if the size of the memory is small, the processing time of the Garbage collector will be short but it will intervene more frequently. If the size of the memory is large, the execution frequency will be less but the processing time will be long. The memory size setting affects the performance of the Garbage collector and is an important factor depending on the needs of each application.

Finally, if you notice that your application consumes a lot of unwarranted memory, you can use a monitoring tool and visualize the activity of the application or activate the garbage collector logs [13].
For more details on available garbage collectors see [16] and [17].

## 6. Conclusion et perspectives

In this article we performed a comparative study to monitor the behavior of the allocation of classes, threads and CPU activity on two different VMs.
We have seen that changing VMs, class allocation, the number of threads change depending on the application running, the CPU consumption also differ and depends on the application launched and the GC used.
At the end of this work, another study seems very interesting to me, it is to make the same comparison on two VMs and on two different operating systems. A priori, Linux will be more efficient since it is not graphical and does not run several processes in the background.

## References

[1] https://docs.microsoft.com/en-us/java/azure/eclipse/azure-toolkit-for-eclipse-installation

[2] http://www.wideskills.com/j2me/install-eclipse

[3] http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf

[4] Nicolas BERTHIER : Analyse de la démographie des objets dans les systèmes Java temps-réel Laboratoire VERIMAG
http://nberth.space/file:magistere/06/report.pdf

[5] https://www.jmdoudoux.fr/java/dej/chap-gestion_memoire.htm#gestion_memoire-4
Paragraphe 57.3.2

[6] https://blog.octo.com/supervisez-votre-jvm/

[7] http://koor.fr/Java/Tutorial/TuningTools.wp

[8] https://ellipse.developpez.com/tutoriels/javaweb/comment-detecter-fuites-memoire-dans-application-ellipse/

[9] Guillaume Salagnac : Synthèse de gestionnaires mémoire pour applications Java temps-réel embarquées

[10] https://www.azul.com/files/Zulu_Embedded_DataSheet.pdf

[11] https://fr.wikipedia.org/wiki/Ramasse-miettes_(informatique)

[12] http://www.java-mobiles.com/tag/business-profession

[13] https://confluence.atlassian.com/doc/garbage-collector-performance-issues-200707997.html

[14] https://stackify.com/what-is-java-garbage-collection/

[15] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

[16] http://www.baeldung.com/jvm-garbage-collectors

[17] https://jelastic.com/blog/tuning-garbage-collector-java-memory-usage-optimization/