International Journal of Technology and Engineering Studies
volume 11 pp. 29-33 doi: https://dx.doi.org/10.20469/ijtes.11.40005

Efficiency Analysis of Garbage Collection Method in Java Virtual Machine-Based Memory Management

Jarudin*
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Radja Satrio Seftiano
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Muhamad Damar Bagas
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Muhammad Abdul Rozak
Institute of Technology and Business Bina Sarana Global,Tangerang, Indonesia.
Ivan Darma Saputra
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Mison Wenda
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Sumarli
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Fajri Khaerullah
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.
Dimas Prasetyo
Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia.

Abstract:Efficient memory management is one of the crucial aspects of Java Virtual Machine (JVM)-based software development. Garbage Collection (GC) is an automated method used by JVMs to manage memory, allowing developers to focus on application logic without worrying about manually allocating and deallocating memory. This study aims to analyze the efficiency of various GC methods available in JVM, such as Serial GC, Parallel GC, Concurrent Mark-Sweep (CMS) GC, and G1 GC. The analysis is conducted based on performance parameters, including execution time, CPU usage, and memory utilization across various application workloads. The results show that the selection of the right GC method is highly dependent on the characteristics of the application and the desired optimization goals. The study provides practical guidance for developers in choosing the appropriate GC method for their specific needs.

Keywords: *Garbage collection, java virtual machine, memory management, efficiency, serial GC, parallel GC, CMS GC, G1 GC.*

Received: 03 June 2025; ***Accepted:*** 02 August 2025; ***Published:*** 15 September 2025

I. INTRODUCTION

Memory management is a fundamental aspect of software development, especially in virtualized environments such as the Java Virtual Machine (JVM). JVMs provide an automated memory management system that allows developers to focus on application logic without worrying about manually allocating and deal locating memory [1]. Memory in a JVM is divided into several main areas,

*Correspondence concerning this article should be addressed to Jarudin, Department of 9 Institute of Technology and Business Bina Sarana Global, Tangerang, Indonesia. E-mail: jarudin@global.ac.id

© 2025 The Author(s). Published by KKG Publications. This is an Open Access article distributed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

including heaps and stacks. The heap is the area where Java objects are dynamically allocated, while the stack is used to store transient data such as local variables and method calls. One of the key features in the JVM is the use of Garbage Collection (GC), which is responsible for detecting and freeing up memory that is no longer used by applications.

This process reduces the risk of memory leakage and improves application stability by automatically managing memory allocation [1, 2]. Different GC methods, such as Serial GC, Parallel GC, CMS (Concurrent Mark-Sweep) GC, and G1 GC, each have characteristics that affect memory management performance in the JVM. Choosing the right GC method depends on a variety of factors, including application type, workload, and performance requirements. While GC provides convenience, this automated memory management process still presents challenges, especially in applications that require low latency or high performance. Some GC methods can result in high CPU usage or decreased performance during the garbage collection cycle.

Therefore, the selection of the optimal GC method is critical to ensure that the application runs efficiently, especially on systems with limited resources [3]. In this context, a deep understanding of how GC works and JVM-based memory management is essential for software developers. Further research on the efficiency of the GC method and its effect on application performance is highly relevant in improving the quality and efficiency of JVM-based systems. How efficient are the various Garbage Collection methods in memory management on the Java Virtual Machine (JVM)?

Analyze the efficiency of the Garbage Collection method in the JVM to provide recommendations for the selection of methods that suit the needs of the application.

II. LITERATURE REVIEW

A. *Types of Garbage Collection Methods*

There are various GC methods used in JVM, and each has its advantages and disadvantages. The Serial Garbage Collector is the simplest method suitable for applications with little memory or applications running on a single processor. Although more efficient in memory usage, this method is less optimal for applications that require fast response times or multi-threaded applications, as garbage collection is done in series on a single thread [3]. Parallel Garbage Collector, on the other hand, uses multiple threads to collect garbage, which increases efficiency in multi-threaded applications.

However, this method can increase CPU usage, especially in applications with short object lifecycles. Concurrent Mark-Sweep (CMS) GC is designed to reduce latency time by doing most of the garbage collection work in the background, while the application stays running. However, CMS often requires more memory to complete its tasks, and can add complexity in the heap setup [4]. G1 Garbage Collector, introduced in Java 7, is designed to better replace CMS in handling large heaps and reduce latency time, making it more suitable for large applications and regional systems and performing garbage collection in a more structured and efficient manner [3].

B. *Garbage Collection's Impact on Performance*

One of the main challenges in JVM-based memory management is ensuring that the GC does not interfere with application performance. The garbage collection process can lead to increased latency, especially in real-time applications or applications that require high performance. In many cases, uncontrolled GC cycles can lead to significant performance degradation, such as decreased throughput or increased CPU usage [5]. Various techniques have been introduced to address this problem, such as heap size tuning, selection of the right GC algorithm, and the application of heap compression techniques to reduce memory overhead [6].

C. *Effect of Workload on GC Efficiency*

The selection of the right GC method is highly dependent on the characteristics of the application's workload. Applications that require efficient real-time memory management, such as server applications or web-based applications, are often better suited to use G1 GC or GC CMS, which reduces latency time. In contrast, applications with light workloads and limited resources may be better suited to use Serial GC to avoid the overhead associated with managing multiple threads. Research conducted by Kolesnikov et al. [7] shows that the selection of an appropriate GC method can improve memory efficiency and significantly reduce latency time, especially in applications that require fast responses.

III. METHODOLOGY

A. *Research Design*

This research uses a quantitative approach to analyze the efficiency of various Garbage Collection (GC) methods in the Java Virtual Machine (JVM). The research design used was an experiment conducted by testing various GC methods on JVM-based applications with various workload conditions. In this experiment, researchers will compare the execution time, CPU usage, and memory utilization of various GC methods, such as Serial GC, Parallel GC, Concurrent Mark-Sweep (CMS) GC, and G1

GC. This experimental method was chosen because it can provide a clear picture of the performance differences between different GC methods and how each method affects the performance of JVM-based applications.

In addition, the experiment will also involve various JVM setting parameters to obtain optimal and reliable results. In this study, the applications used for the experiment were Java-based applications with varied workloads, ranging from applications that had few living objects to applications that managed a large number of dynamic objects. Testing will be conducted on multiple platforms with different hardware specifications to see how the GC method performs in different environments.

B. Data Collection Techniques

The data used in this study was obtained through direct testing on JVM-based applications run with various GC methods. Data retrieval is carried out in two stages: Initial Testing: The application will be run without GC to obtain baseline data regarding CPU and memory usage under normal conditions without garbage collection. Testing with GC: After that, the application will run with various GC methods, and the data captured includes execution time, CPU usage, and memory utilization during the GC cycle. Data retrieval is performed using performance monitoring tools such as JVisualVM and Java Flight Recorder to monitor memory usage, the number of objects collected, and the impact of GC on application performance. The data collected will include statistics such as the time it takes to run the GC cycle, CPU usage during the GC cycle, and the size of the heap after garbage collection. Each test will be repeated three times to ensure consistency of results and reduce possible measurement errors.

C. Data Analysis

The stage of data obtained from the experiment will be analyzed using descriptive and inferential statistical techniques. Descriptive analysis is used to describe data characteristics such as execution time, CPU usage, and memory usage on each GC method. In addition, inferential analysis will be used to compare the performance differences between GC methods by using t-tests to measure the significance of differences between the tested methods. The results of the analysis will show which GC method is the most efficient in terms of resource usage and execution time. Based on the results of the analysis, recommendations will be given to developers to choose the GC method that best suits their application needs, taking into account factors such as latency, CPU usage, and memory utilization. The study will also explore the influence of heap size and workload on GC performance, as well as provide practical guidance for tuning JVMs in different application contexts.

IV. RESULTS

In this section, the results of tests conducted using various Garbage Collection (GC) methods on Java Virtual Machine (JVM)-based applications will be discussed. Testing is performed on applications that have different levels of workloads, from applications with few live objects to applications with large numbers of dynamic objects. Testing is performed on platforms with different hardware specifications to analyze how each GC method performs in diverse environments. The data obtained during testing includes three main parameters: Execution Time: The time it takes for the application to complete the entire execution cycle with GC active. CPU usage: The percentage of CPU usage during the GC cycle and garbage collection. Memory Usage: Memory usage after the GC cycle, as well as changes in heap size. The following table 1 shows the runtime test results for various GC methods in applications with light and heavy workloads.

TABLE 1
Results of the Test

GC Method	Execution Time (sec)	CPU Usage (%)	Memory Usage (MB)
Serial GC	3.25	10	1024
Parallel GC	2.85	40	1020
CMS GC	3.10	30	1030
G1 GC	2.60	25	1015

V. DISCUSSION

From the test results obtained, it can be seen that each GC method has different characteristics in terms of execution time, CPU usage, and memory usage. Here are some key findings from the test results: Execution Time: The G1 GC method shows the shortest execution

time compared to other methods, with an average time of 2.60 seconds. This shows that G1 GC is more efficient at managing memory and reducing waste collection overhead, especially in applications with heavy workloads [8]. On the other hand, Serial GC requires a longer execution time, which is 3.25 seconds, which can be caused by garbage collection being done serially on a single thread.

Parallel GC and CMS GC show pretty good results with execution times of 2.85 and 3.10 seconds, respectively, but it's still a bit longer compared to G1 GC. CPU usage: Parallel GC requires the highest CPU usage, with average usage reaching 40%. This is due to the characteristics of Parallel GC, which uses multiple threads for garbage collection, which increases the load on the CPU [2]. The G1 GC, while providing faster execution times, shows lower CPU usage, only about 25%. The GC CMS also shows moderate CPU usage, which is around 30%, as most GC cycles are performed in parallel in the background [9]. Memory Usage: Memory usage after a GC cycle tends to be lower on G1 GC and Parallel GC, with an average of 1015 MB and 1020 MB, respectively. Meanwhile, the GC Series shows the highest memory usage, which is 1024 MB, which may be related to the simpler way GC works. Although the GC CMS uses more memory (1030 MB), this memory usage is still within reasonable limits for larger, more complex applications.

A. *Comparison and Implications*

Overall, the test results show that G1 GC is the most efficient method in terms of execution time and CPU usage, especially in applications with heavy workloads. G1 GC excels because it is able to efficiently manage large heaps, reduce latency, and maintain application performance even on systems that require complex memory management. However, G1 GC also requires more precise tuning to optimize performance, especially in applications with large heap sizes. Meanwhile, Parallel GC can be a good choice for applications that require fast memory processing with higher CPU usage. This method is particularly effective in applications running in multi-core environments, where the speed of garbage collection can be accelerated through the use of multiple threads. On the other hand, Serial GC may still be an option for lighter applications or those running on devices with limited resources, albeit with limitations in execution time and less efficient memory management. GC CMS is a good choice for applications that require low latency time, but may not always be efficient in memory usage. Overall, the selection of the right GC method should consider the characteristics of the application, the size of the heap, and the hardware specifications on which the application is run [5].

VI. CONCLUSION

This study aims to analyze the efficiency of the Garbage Collection (GC) method in memory management based on the Java Virtual Machine (JVM). Based on the results of experiments conducted with various GC methods (Serial GC, Parallel GC, CMS GC, and G1 GC) on Java-based applications with various workload levels, it can be concluded that G1 GC performs best in terms of execution time and CPU usage, especially in applications with heavy workloads. By using a more advanced generation of waste collection algorithms, G1 GC can reduce latency time and maximize the use of existing hardware resources.

Therefore, G1 GC is the optimal choice for applications that require efficient memory management with large heaps. Parallel GC has proven to be efficient in terms of execution time, although it requires higher CPU usage. This method is suitable for applications running on multi-core platforms, where garbage collection capabilities can be accelerated by the use of parallel threads. The GC series is simpler and suitable for lighter applications or running on devices with limited resources. Although it has a longer execution time compared to other methods, Serial GC can be an option for applications with less large memory requirements. CMS GC can still be used for applications that require low latency, although the results are not as efficient as G1 GC in terms of memory management. GC CMS tends to be better on applications that have critical response times, although it's worth noting that this method can consume more memory.

Overall, the selection of the right GC method depends largely on the characteristics of the application and the hardware on which the application is run. G1 GC is recommended for large applications that require efficient memory management, while Parallel GC is better suited for applications with parallel processing needs. Meanwhile, Serial GC and CMS GC are still relevant for applications with limited resources or with low latency. This research provides useful insights for software developers in selecting and configuring the right GC method according to their application needs, which can ultimately improve application performance and resource usage efficiency.

REFERENCES

[1] P. Pufek, H. Grgic, and B. Mihaljević, "Analysis of garbage collection algorithms and memory management in java," in *2019 42nd International Convention on Information and Communication Tech-*

nology, Electronics and Microelectronics (MIPRO), 2019. doi: 10.23919/MIPRO.2019.8756844 pp. 1677–1682.

[2] J. Velasco, D. A. Alonso, and K. Olcoz, "Memory power optimization of java-based embedded systems exploiting garbage collection information," *Journal of Systems Architecture*, vol. 58, no. 1, pp. 61–72, 2012. doi: 10.1016/j.sysarc.2011.11.002

[3] H. Lee, Q. Chen, H. Yeom, and Y. Son, "An efficient garbage collection in java virtual machine via swap i/o optimization," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020. doi: 10.1145/3341105.3373982

[4] J. R. Jalaman and J. Teleron, "Optimizing operating system performance through advanced memory management techniques: A comprehensive study and implementation," *Engineering and Technology Journal*, vol. 9, no. 05, 2024. doi: 10.47191/etj/v9i05.33

[5] G. Vijayakumar and R. K. Bharathi, "Predicting jvm parameters for performance tuning using different regression algorithms," in *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, 2022. doi: 10.1109/ICERECT56837.2022.10060788 pp. 1–8.

[6] S. Iqbal, M. Khan, and I. Memon, "A comparative study of garbage collection techniques in java virtual machines," *Sindh University Research Journal*, vol. 44, 2012.

[7] K. LU, X. ZHOU, W. CHI *et al.*, "A high performance system level garbage collection algorithm," *Computer Engineering & Science*, vol. 32, no. 11, p. 119, 2010.

[8] A. Chourmouziadis and G. Pavlou, "Efficient information retrieval in network management using web services." [Online]. Available: http://www.ee.surrey.ac.uk/CCSR/

[9] D. Rong-Hui, "Performance tuning of java garbage collection mechanism," *Computer Engineering and Design*, 2006.