GARBAGE COLLECTION PERFORMANCE TUNING USING REINFORCEMENT LEARNING

Prajapati Keyur p1, Akashbhai Dave2
Assistant Professor1, 2.
Department of Computer Engineering, MBICT, New Vallabh Vidhyanagar, Anand1, 2.
Aadave@mbict.ac.in1, kprajapati@mbict.ac.in2

ABSTRACT

Garbage collection is the primary key feature for an end-user developer for any kind of application development. It frees the developer from the burden of memory management. Explicitly allocation and de-allocation of memory management create a lot of burdens. Java GC is a magnificent feature in JVM. Tuning of JVM [2] GC is a hike in tech era development and we can do it by reinforcement mechanism.

Keywords: Garbage collection, Memory management, Reinforcement learning, Serial collector, Parallel collector, Garbage collector G1, muti-threaded hardware.

INTRODUCTION

Java is a vast platform for developers to build any kind of application. Except for Java language, it is a big issue for managing memory in the execution of application. Java allow to define a constructor but doesn't allow define explicitly destructor. And this is the reason why Java is robust and famous.

Java has its own mechanism to manage memory. It has to be handled by inside component in JVM called GC(Garbage collector). Different flavors of GC in JVM are available. The serial collector is applicable for small size application (approx not higher than 100MB in size). Parallel collector is to provide maximum throughput for the application. While CMS(Concurrent-Mark & Sweep) is known as latency collector.

Reinforcement learning [9] is a type of machine learning in which reward comes in the picture for the smooth functioning and proper selection of the GC algorithm. This is the technique in which action would be taken by the machine itself and based on it, reward [2]will be given against its action. Then finally it will be a log generated file and all the future action will be taken the base on this log file.

To my current best knowledge I am not aware of any other attempt to utilize reinforcement learning in a JVM [11]. Therefore, I am not able to provide references to similar approaches to that particular
problem. Many papers on garbage collection techniques include some sort of heuristics on when the technique should be applied, but they are usually quite simple. These methods are usually straightforward and based on general rules that do not take the specific characteristics of the application into account.

Garbage collection [3] is gaining more importance in next-generation languages like Java and C#; But the selection of a suitable garbage collector still depends on the type of application and underlying resources. Therefore, it is very important to invoke a suitable garbage collector for a particular type of application.

An object is considered garbage and its memory can be reused by the VM [2] when it can no longer be reached from any reference of any other live object in the running program. A theoretical, most straightforward garbage collection [1] algorithm iterates over every reachable object every time it runs. Any leftover objects are considered garbage. The time this approach takes is proportional to the number of live objects [6], which is prohibitive for large applications maintaining lots of live data.

The primary measures of garbage collection are throughput and latency.

- Throughput [4] is the percentage of total time not spent in garbage collection considered over long periods of time. Throughput includes time spent in allocation (but tuning for the speed of allocation generally isn't needed).
- Latency is the responsiveness of an application. Garbage collection pauses to affect the responsiveness of applications.

CMS is default in JDK 8.0 and is good for large sized application. In those two main cycles of GC 1,) Minor collection and 2) Major collection happened.

The minor collection finds restful objects and migrates them to survivor 0 and survivor 1 as per need. Major collection must need a pause for the application and in that time duration old generation wipe out their objects and free memory.

In JDK 9.0 by default, the collector is garbage collector G1; it has the same mechanism what we realized in CMS.

Instead of static allocation memory for Eden space and two different survivors, it will create memory blocks for the survivor, old and young and look up the current scenario and make it tunable as per need.

So now in a day's Garbage collector, G1 gives you low pauses with high throughput.

But still some time user expectation is not satisfied and for that still, we have to need more optimization for the same. And by using a reinforcement learning approach we can still make good tunable g1.
**RESEARCH CONCEPT**

By using reinforcement learning we can still optimize the performance of garbage collector g1.

The Garbage- First stack is partitioned into equivalent estimated pile areas, each a coterminous scope of virtual memory. Designation in a stack district comprises of augmenting a limit, top, among apportioned and unallocated space. One area is the present assignment district from which stockpiling is being dispensed. Since we are chiefly worried about multiprocessors, mutator strings distribute just string neighborhood designation bu_ers, or TLABs [14], specifically in this store area, utilizing a look at and-swap, or CAS, task. They then allocate objects privately within those bu_ers, to minimize allocation contention. When the current allocation region is allocated, a new allocation region is chosen. Empty regions are organized into a linked list to make region allocation a constant time operation.

Reinforcement learning techniques will give you best result and execution once all therapy is logged. And in future it has also the capability of its own learning will give an extraordinary hike for the long sized applications.

In the above figures, we realized the optimized work of GC.

In Garbage collector g1 [4] we can see the dynamic memory allocation units. By using reinforcement learning still, we have better look up for the concurrent mark-sweep and garbage collector g1. Incremental Collection
An incremental collector divides the heap into sections and collects one section at a time. One consequence of this is that only a small amount of the garbage – the garbage of one section of the heap [7] – is collected at a time and that it may not be enough to satisfy the allocation needs of the program. A resulting positive feature is that an incremental garbage collection does not cause such a large break in the running program as a complete garbage collection of the heap might do. Why this technique is seldom used is because it is very hard to implement.

Concurrent Collection [13] another effective, but also hard implemented garbage collector technique is the concurrent approach. A concurrent garbage collector works in a certain thread by itself, at the same time as the program. To work “at the same time as the program” means that the program and the collector take turns executing instructions. Both the incremental and the concurrent collectors [12] collect little garbage at a time. The difference between the two approaches is that incremental “little at a time”-approach means little garbage is collected at a time, where little refers to a small area of the heap. Concurrent “little at a time”-approach, on the other hand, means little garbage collection [5] at a time, that is, the garbage collection is divided into steps and only one step at a time is performed. In other words little, but not necessarily complete, garbage collection is performed at a time. Consequently concurrent collectors need to consider allocations made by the program in between the step executions of

The collector. Another important issue is to keep track of the changes made by the running program in order to be able to update all pointers [10] correctly. This technique is hard to implement, but is very effective according to total interruption time of the running program. The alternative is to stop the program and complete the garbage collection and then return to the program, which would cause a much more noticeable interruption Parallel Collection [8]

The parallel collection technique may be used when the system where the collector is being used has more than one processor. Only in this case would it be possible for several threads to really work at the same time, i.e. in parallel. Advantages with this technique are that the garbage collector may work concurrently and incrementally on each processor and thereby shorten the total time of the garbage collection, i.e. shorten the interruption time in the running program.

Important to consider when it comes to parallel garbage collection is the need of synchronization of the garbage collecting threads. It is also important to distribute the work to the separate processors in an efficient and fair way.

CONCLUSION

BY using reinforcement learning we can still optimize the performance of application in terms of short pause and high throughput.

By making hybrid combo of different GC give us a better choice for application performance.
REFERENCES


