International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 01 | Jan 2020 www.irjet.net p-ISSN: 2395-0072

# Performance Comparison of Real-Time Garbage Collection in the Sun Java Real-Time Systems with Different Garbage Collection Techniques

**Amandeep kaur[1], Balpreet kaur[2], Rupinder kaur[3]**

[1,2,3]*Assistant Professor, Dept. of CSE, BBSBEC, Fatehgarh Sahib*

---

**Abstract -** *Garbage collection performances vary when we use reference counting technique. Generational garbage collection worst case allocation characteristics are different from reference counting. In this review paper, a performance comparison has been made using the default and the proposed RTGC(Real time garbage collection) approach.*

***Key Words***: Real time garbage collection, Reference counting

## 1. INTRODUCTION

Sun Java Real-Time System (Java RTS) is Sun's commercial implementation of the Real-Time Specification for Java (RTSJ), or JSR 1.

Starting with Sun Java RTS 2.0, a new real-time garbage collector (RTGC), based on Henriksson's algorithm, is available. The garbage collector runs as one or more real-time threads (RTTs). These run at a priority that is lower than all instances of NoHeapRealtimeThread (NHRT) and that may be lower than some RTTs as well, so that critical threads may preempt the collector. In this way, critical threads are shielded from the effects of GC.

By default, the garbage collector runs at its initial priority, which is below that of the noncritical real-time threads. But as memory grows short, the VM will boost or raise the priority of the collector to the maximum configured priority.

### 1.1 Thread distinction in java

The important point about the RTGC provided with Java RTS is that it is fully concurrent, and it can thus be preempted at any time. There is no need to run the RTGC at the highest priority, and there is no stop-the-world phase, during which all the application's threads are suspended during GC.

RTGC starts executing at its initial priority and is preempted by a high-priority thread. When that thread stops, the RTGC runs again but is again preempted. Finally, if the running threads are allocating memory and the remaining memory becomes low enough, the RTGC is boosted to its maximum priority, where it is preempted only by critical threads.

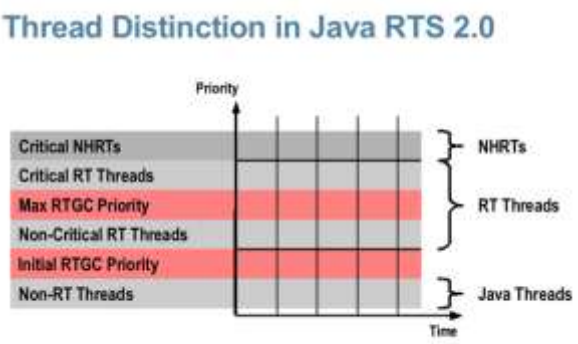


**Fig -1**: thread distinction in java

### 1.2 Default RTGC using one and two CPUs

On a multiprocessor, one CPU can be doing some GC work while an application thread is making progress on another CPU. In Figure 6, the critical NHRTs are running on a separate CPU and therefore do not pre-empt the RTGC. The RTGC runs to completion without its priority having to be boosted.

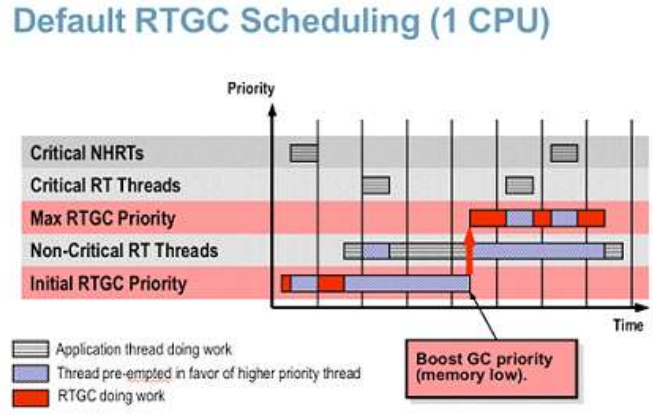


**Fig -2**: default real time garbage collection in one CPU

The RTGC considers that the criticality of an application's threads is based on the threads' respective priorities and ensures hard-real-time behavior only for real-time threads at the critical level, while trying to offer soft-real-time behavior for real-time threads below the critical level.

This reduces the total overhead of the RTGC and ensures that the determinism is not affected by the addition of new low-priority application threads. This makes the configuration easier because there is no need to study the allocation

behavior of an application in its entirety in order to configure the RTGC.

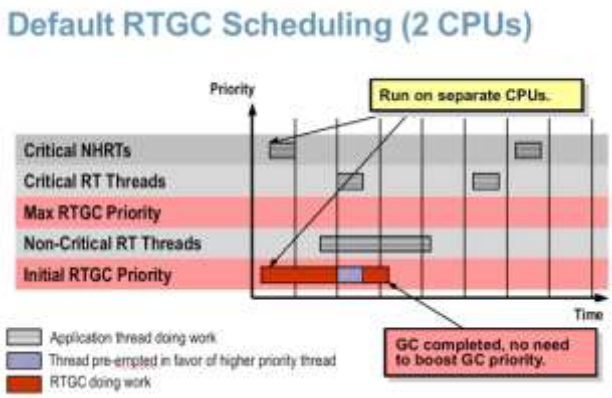


**Fig -3: default real time garbage collection in two cpus**

## 2. The proposed approach

Java possesses many advantages for embedded system development, including fast product deployment, portability, security, and a small memory footprint. As Java makes inroads into the market for embedded systems, much effort is being invested in designing real-time garbage collectors. The proposed garbage-collected memory module, a bitmap-based processor with standard DRAM cells is introduced to improve the performance and predictability of dynamic memory management functions that include allocation, reference counting, and garbage collection. As a result, memory allocation can be done in constant time and sweeping can be performed in parallel by multiple modules. Thus, constant time sweeping is also achieved regardless of heap size. This is a major departure from the software counterparts where sweeping time depends largely on the size of the heap. In addition, the proposed design also supports limited-field reference counting, which has the advantage of distributing the processing cost throughout the execution.

## 3. Results: Performance comparison

By doing reference counting operation in a coprocessor, the processing is done outside of the main processor. Moreover, the hardware cost of the proposed design is very modest (about 8,000 gates). Our study has shown that 3-bit reference counting can eliminate the need to invoke the garbage collector in all tested applications. Moreover, it also reduces the amount of memory usage by 77 percent.

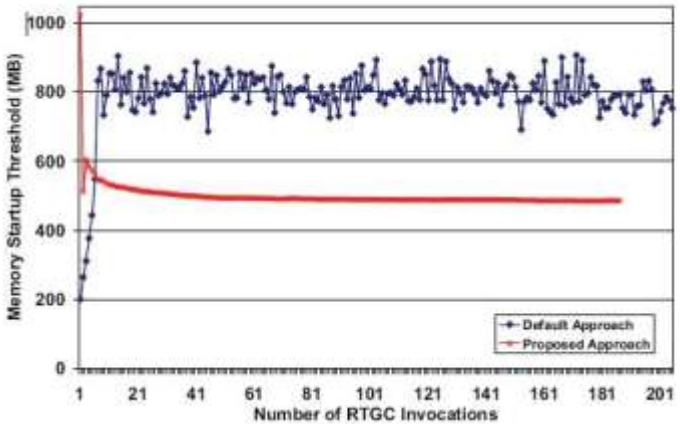


**Fig -4: default real time garbage collection in two cpus**

## REFERENCES

1. D.F. Bacon, P. Cheng, and V.T. Rajan, "A Real-Time Garbage Collector with Low Overhead and Consistent Utilization," In Proceedings of the ACM Principles of Programming Languages (POPL 03), pp. 285-298, 2003.

2. Roger Henriksson. "Scheduling garbage collection in embedded systems." Phd thesis, Lund Institute of Technology, 1998.

3. Scott Nettles and James O'Toole."Real-Time replication garbage collection. Pages 217-226. ACM Press, 1993.

4. Fridtjof Siebert." Real –Time garbage collection in multi-threaded systems on a single processor".

5. Fridtjof Siebert. Hard real-time garbage collection. Phd thesis, Universitat Karsruhe, 2002.