A Study on GC Performance of ART

Shintaro Hamanaka, Saneyasu Yamaguchi
Electrical Engineering and Electronics, Kogakuin University Graduate School
cm15021@ns.kogakuin.ac.jp, sane@cc.kogakuin.ac.jp

I. Introduction

The latest Android OS adopts ART as the Android application runtime environment. ART has several GC algorithms. Naturally, it is expected that comparing their performance is important for choosing the suitable GC [1] according to application behavior. In this paper, we explore performance of GCs in ART.

II. GC Performance Evaluation

A. Mutator performance and STW time

We performed our benchmark applications using CMS (Concurrent Mark and Sweep) GC and SS (Semi Space) GC in ART. In the initialization phase, the benchmark application creates array of *Link* objects with length 100,000. A *Link* object allocates an array of `int` type variables with length *m* and creates a pointer to another *Link* instance. *m* is determined on creating an instance. *m* is randomly set using exponential distribution with average m_{avg}. In the measuring phase, the benchmark repeats the flowing three things. 1) create a *Link* instance. 2) overwrite a randomly selected instance in the *Link* array with the new *Link* instance. 3) change the pointer of the randomly selected *n* instances to randomly selected instances. Random selections in 2) and 3) obey uniform distribution. As a result of 2), the overwritten instance loses a pointer from the array. If the lost pointer is the last link to the instance, it becomes a garbage object. 3) is modification to an instance. Thus, this causes a re-mask process, which is STW (stop the world), in CMS GC. In this paper, we call *n* "link change frequency". Experimental times are three minutes.

Fig. 1 and Fig. 2 show mutator performance and STW time. Performance indicates object creation throughput, which is the number of repeated operation of 1), 2), and 3) per second. Fig. 1 shows the relation between m_{avg} (average length of array of `int` in *Link*) and results (performance and STW time). Fig. 2 shows the relation between *n* (link change frequency) and results. From these figures, we can say that CMS GC is better than SS GC with all the cases from the aspects of performance and STW time. Focusing on the case with large m_{avg}, we can see that the mutator stopped almost all time.

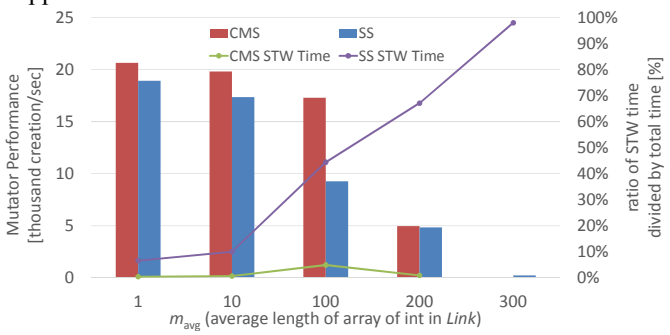

Fig. 1 Performance and STW time (vs m_{avg})

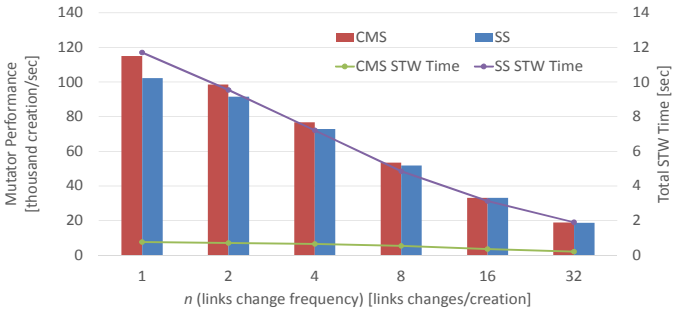

Fig. 2 Performance and STW time (vs link change frequency)

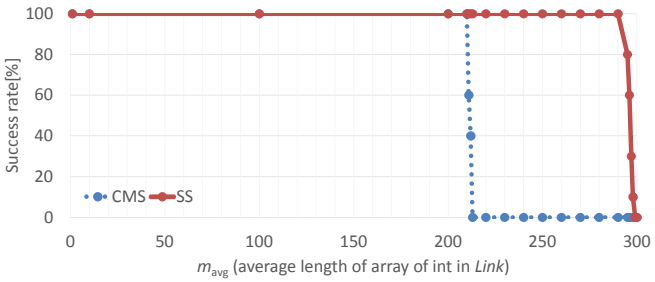

Fig. 3 Memory availability

B. Memory Availability

We repeated 100,000 creations with large m_{avg} and checked memory availability. The experimental results are shown in Fig. 3. "Success rate" in the figure shows the ratio of the benchmark finished successfully. If a construction of a new instance fails, the benchmark application forcefully terminated. This is failure. If all the constructions, 100,000 times creations, are performed successfully, the benchmark finishes successfully. The figure implies that SS GC is better than CMS GC in the aspect of memory allocation.

From our experiments, we can conclude that CMS GC is better than SS GC in usual applications, which does not severely consume memory. On the contrary, SS GC is suitable for memory-consuming applications.

III. Conclusion

In this paper, we investigate GC performance of ART. Our experimental results shows that CMS GC is suitable for usual applications, non-memory-consuming application. Contrary, SS GC should be chosen in cases of applications which heavily allocate memory.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 24300034, 25280022, 26730040, 15H02696.

[1] Sunil Soman, Chandra Krintz, David F. Bacon, "Dynamic selection of application-specific garbage collectors," In Proceedings of the 4th international symposium on Memory management (ISMM '04). ACM, New York, NY, USA, 49-60., 2004