IJITKM 0973-4414 Volume-9, Number-1 Jun-Dec 2015 pp. 100-102 Impact Factor 2.8 available online at www.csjournalss.com

Garbage Collection Time due to Major Collection

Shubhnandan S Jamwal, Nitan Singh
Department of Computer Science & IT, University of Jammu, Jammu, India
Jamwalsnj@gmail.com

Abstract: Garbage collection (GC) plays an important role in managing the memory automatically. But it is also a fact that for interactive and real time applications long pauses are not desirable. In the current research paper we have executed all the benchmarks of SPECjvm2008 with all the four garbage collectors available in java. After executing the benchmark we found the time during which the application is paused due to major collection.
Keywords: Benchmarks, garbage collector, major collection, pauses.

Introduction

GC is the process of automatic memory management or reclamation in which memory is reclaimed from the inactive objects. The memory reclaimed is added to the pool of free memory. Many languages including Java and C# have incorporated garbage collectors. The four garbage collectors available in JDK 1.7.0. are Serial, Parallel, Parallel Old and Concurrent Mark-Sweep collectors. The selection of a particular collector depends on the class of the machine. If the machine class is server then by default Parallel collector is selected. If the machine class is client the default collector is serial collector.

a. Serial Collector: The serial collector is suitable for small applications whose size is less than 100 mb. In this collector both young and old generations are collected serially. The application execution is paused when the collector is running.

b. Parallel Collector: This collector is suitable for machines having multiple processing units and has to operate on large data sets. In this collector minor collections are performed simultaneously while the major collections are performed serially.

c. Parallel Old Garbage Collector: This collector was introduced in J2SE 5.0 update 6. In this collector minor as well as major collections are performed parallel with the use of multiple CPU's. ParallelOld collector uses a new algorithm for old generation garbage collection.

d. Concurrent Mark-Sweep (CMS) Collector: In this collector minor collection are performed serially. But the major collection is performed simultaneously with the execution of the application. It is appropriate for applications that require shorter garbage collection pauses and can afford to share processors with the garbage collector thread when the application is running.

The garbage collector in the old generation is activated only when the old generation fills up with the inactive objects. When garbage collector is running to collect the garbage (dead objects) in the old generation, the application is paused. These pauses are long as compared to the pauses caused by minor collections.

Review of Literature

It was shown by Sunil Soman and Chandra Krintz [1] performance of the application in garbage collecting languages is dependent on the application behavior and available resources. They also proved that no garbage collector perform best across all programs and heap sizes. When the resources are abundant all the collectors behave in similar manner. But in case if the memory is limited the hybrid collector can improve the throughput of the application at least by 50%. It was shown by Clement R. Attanasio, David F. Bacon, Anthony Cocchi, and Stephen Smith [2] that parallel collector is best for online transactions processing applications. The concurrent collector was modified by Katherine Barabash, Yoav Ossia, and Erez Petrank [3]. They improved the throughput of the application, stack, and the behavior of cache of the collector without foiling the other good qualities such as short pauses and high scalability. Their proposed algorithm was implemented on the IBM production JVM and obtained a performance improvement by 26.7%, a reduction in the heap consumption by up to 13.4%, and no change in the pause time. Stephen M Blackburn, Perry Cheng, and Kathryn S McKinley [4] analyzed that the overall performance of generational collectors as a function of heap size for each benchmark is mainly dictated by collector time. Mark Sweep does better in small heaps and Semi Space is the best in large heaps. But the results are not satisfactory in small memory. Garbage collection algorithms still trade for space and time which needs to be better balanced for achieving the high performance computing. Kim, T., Chang, N., and Shin, H. [5] observed the memory management behavior of several Java programs from the SPECJVM98 benchmarks. The important observation is that the default heap configuration used in IBM JDK 1.1.6 results in frequent garbage collection and the inefficient execution of applications.

Stephen M Blackburn, Perry Cheng and Kathryn S McKinley[6], experimental design shows key algorithmic features and how they match program characteristics to explain the direct and indirect costs of garbage collection as a function of heap size on the SPEC JVM benchmarks. They find that the contiguous allocation of copying collectors attains significant locality benefits over freelist allocators. The reduced collection cost of the generational algorithms together with the locality benefit of contiguous allocation motivates a copying nursery for newly allocated objects. Jurgen Heymann [7] presented an analytical model that compares all known garbage collection algorithms. The overhead functions are easy to measure and tune parameters and account for all relevant sources of time and space overhead of the different algorithms.

Experimentation

Benchmarks

In the current research paper SPECjvm2008 benchmark suite is used. The eleven benchmarks available in SPECjvm2008 are studied in real JVM and no simulators are being used in the experimentation. The eleven benchmarks specified in the SPECjvm2008 are executed over a wide range of heap size varying from 20 mb to 400 mb with an increment of 20 mb size. Each of the benchmark is executed 10 times in a fixed heap size and the arithmetic mean is obtained. The performance of all garbage collectors is measured over different heap sizes. The Processor used in current research is Intel(R) Core(TM) Duo CPU T2250 @ 1.73GHz. 32 bit system with 2038 megabyte RAM. The frequency of the memory is 795MHz. The operating System used Microsoft Windows XP Professional Version 2002 Service Pack 2. Java used for performing the tests is jdk1.7.0.

Garbage Collection time for major collection

It is defined as the time spent in collecting the garbage due to major collection. When garbage collector is running in the old generation to collect the garbage, the application is paused during that time. The pause due to major collection is large as compare to pause caused by minor collection. The garbage collection time due to major collection should be as short as possible.

Conclusion/Future Work

It has been observed that, initially for small heap size the garbage collection time due to major collection is large. But as the heap size increases the garbage collection time decreases and after some time it becomes constant. From these experiments we conclude that heap size is an important factor in garbage collection. Large the heap size small is the garbage collection time. In future work we would find the effect of other metrics on garbage collection on all the benchmarks of SPECjvm2008.

References

[1]. S. Soman, C. Krintz,"Application-specific Garbage Collection", J. of Sys. and Software, Elsevier Science Inc. New York, NY, USA, Vol. 80, No. 7, pp. 1037-1056, July 2007.

[2]. C. R. Attanasio, D. F. Bacon, A. Cocchi, S. Smith, "A Comparative Evaluation of Parallel Garbage Collector and Implementations", LCPC'01 Proc. of the 14th Int. Conf. on Languages and Compilers for Parallel Computing, Springer- Verlag Berlin, Heidelberg, LNCS 2624, pp. 177–192, 2003.

[3]. K. Barabash, Y. Ossia, E. Petrank,"Mostly Concurrent Garbage Collection Revisited", OOPSLA '03 Proc. of the 18th Annual ACM SIGPLAN Conf. on Object-Oriented Prog., Systems, Languages, and App., pp. 255-268, ACM New York, NY, USA , 2003.

[4]. O. Agesen, D. L. Detlefs,"Finding References in Java Stacks", Submitted to OOPSLA'97 Workshop on Garbage Collection and Memory Manag., Atlanta, GA, October 1997.

[5]. Kim, T., Chang, N., Shin, H.,"Bounding Worst Case Garbage Collection Time for Embedded Realtime Systems", RTAS '00 Proc. of the Sixth IEEE Real Time Tech. and Appl. Symp.pp. 46, IEEE Compu. Society Washington, DC, USA, 2000.

[6]. S. M. Blackburn, P. Cheng and K. S. McKinley, "Myths and Realities: The Performance Impact of Garbage Collection", Proc. of the Joint Int. Conf. on Measurement and Modeling of Compu. Sys., June 12–16, ACM Press, New York, NY, USA, 2004.

[7]. J. Heymann,"A Comprehensive Analytical Model for Garbage Collection Algorithms", ACM SIGPLAN Notices, Vol. 26, No. 8, August 1991.

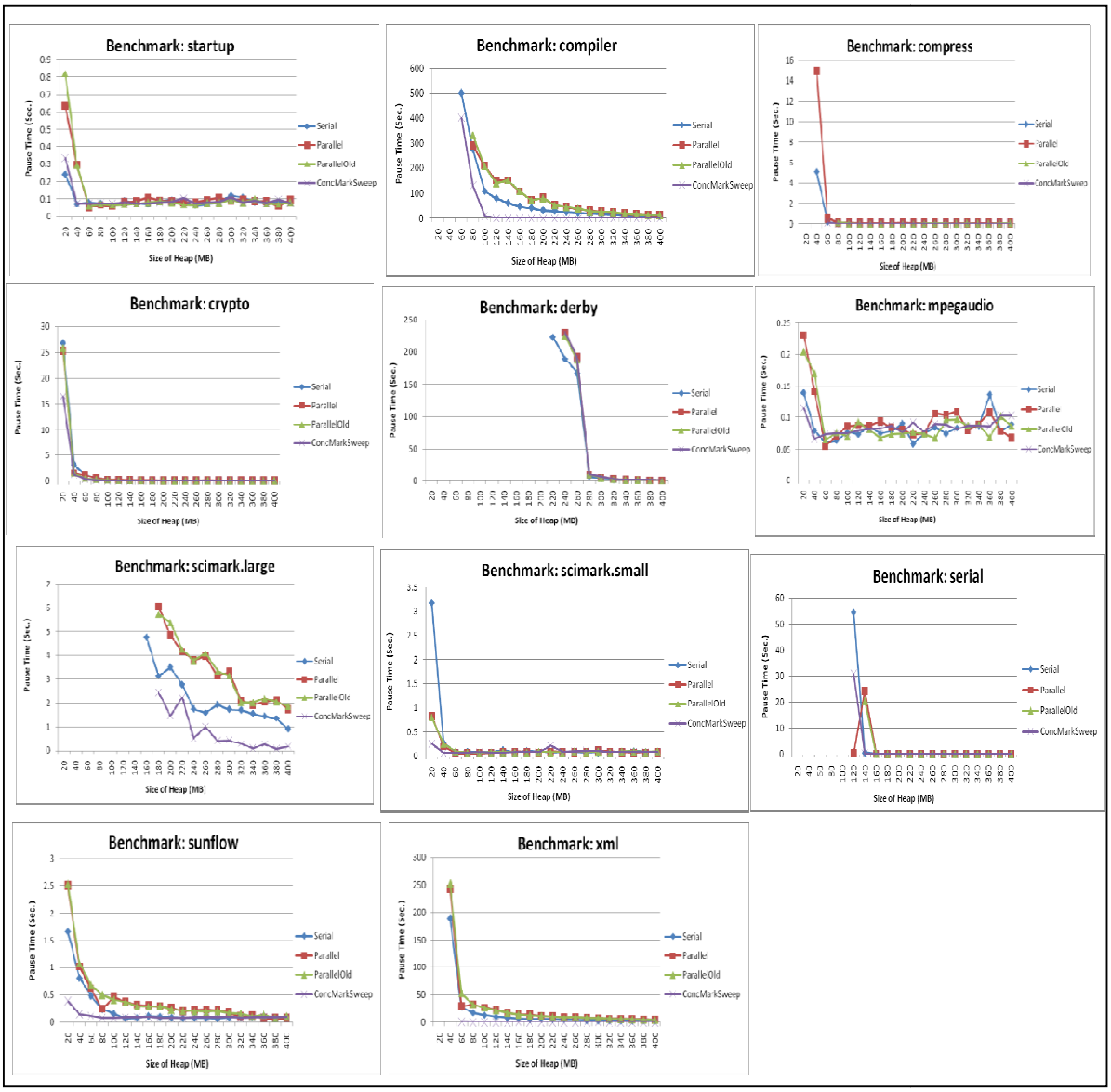

Figure 1. Time spent by garbage collectors for all the benchmarks of SPECjvm2008 due to major collection .