Leveraging a user-land page table to implement a concurrent garbage collector

- Supervisor: Gaël Thomas

1 Context

A garbage collector [3–5, 8] is a system component that automatically and transparently reclaims the unused memory of an application. Using a garbage collector increases productivity. A garbage collector avoids first many bugs caused by explicit memory management (e.g., double free or use after free). Using a garbage collector also avoids the use of direct pointers in the code, which allows the language runtime to enforce the type safety at runtime.

Today, many language runtimes use a garbage collector. This is for example the case of Java, Go or python. These languages are commonly used to perform large data analysis. For these applications, the garbage collector has to manage very large heaps of hundreds of gigabytes. Unfortunately, garbage collectors do not scale well with such large heap sizes.

The problem is that a garbage collector has to inspect the memory in order to find the dead objects while the application modifies the memory. Running the garbage collector in parallel of the application leads thus to concurrent read/write accesses to the heap from both the collector and the application. These concurrent read/write accesses lead to inconsistencies. In order to avoid these inconsistencies, we have to finely synchronize the garbage collector and the application. Unfortunately, the current techniques used to synchronize the garbage collector and the application leads to performance degradation, which becomes high when the heap is large [1, 2, 6, 7].

2 Subject

As part of the DiVA project, the Benagil team is implementing a large scale garbage collector. The implementation consists in two components implemented by two PhD students. The first one is a new GC algorithm implemented inside the Hotspot Java virtual machine. The second one is an infrastructure that allows a process to leverage a userland page table. In order to make the GC algorithm efficient, we propose to use the page table in user land in order to implement the Java heap as a cache for a large memory located on different machines. By managing the Java heap as a cache, we eliminate the risk of inconcistencies when the application and the collector executes concurrently. For that, we have to ensure that, during a collection, the Java virtual machine do not evict data from its cache. By preventing eviction, the memory is only modified by the collector during a collection, which avoids concurrent read/write accesses.

The master student will join the team in order to help porting the GC algorithm on the infrastructure that exposes a userland page table. This implementation will be carried on by the two PhD students, and the master student will come in support to the team. He will help identify bugs, ensure the maintainability of the code, implement regression tests and implement simple features.

3 Expected skills

The candidate must have a good background in system programming, C and Java.

References

[1] Maria Carpen-Amarie, Patrick Marlier, Pascal Felber, and Gaël Thomas. A performance study of java garbage collectors on multicore architectures. In *Proceedings of the International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM'15*, page 10, San Francisco Bay Area, USA, 2015. ACM.

[2] Edsger W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On-the-fly garbage collection: An exercise in cooperation. *Communications of the ACM*, 21(11):966–975, nov 1978.

[3] Lokesh Gidra, Gaël Thomas, Julien Sopena, and Marc Shapiro. A study of the scalability of stop-the-world garbage collectors on multicores. In *Proceedings of the conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'13*, pages 229–240, Houston, Texas, USA, 2013. ACM.

[4] Lokesh Gidra, Gaël Thomas, Julien Sopena, Marc Shapiro, and Nhan Nguyen. NumaGiC: a garbage collector for big data on big NUMA machines. In *Proceedings of the conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'15*, page 14, Istanbul, Turkey, 2015. ACM.

[5] Richard Jones, Antony Hosking, and Eliot Moss. *The Garbage Collection Handbook: The Art of Automatic Memory Management.* Chapman & Hall/CRC, 1st edition, 2011.

[6] Anatole Lefort, Yohan Pipereau, Kwabena Amponsem, Pierre Sutra, and Gaël Thomas. J-NVM: Off-heap persistent objects in java. In *Proceedings of the Symposium on Operating Systems Principles, SOSP'21*, page 16, online, 2021. ACM.

[7] Gil Tene, Balaji Iyengar, and Michael Wolf. C4: The continuously concurrent compacting collector. In *Proceedings of the International Symposium on Memory Management, ISMM'11*, page 79–88. ACM, 2011.

[8] Gaël Thomas. *Improving the design and the performance of managed runtime environments.* PhD thesis, UPMC Sorbonne Université, Paris, France, 2012.