Garbage Collection in Multi-Threaded Java Programs

Gerhard W Dueck
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada

Abstract

Java relies on the garbage collector to free the memory of objects that are no longer accessible. Many garbage collectors are executed in two phases: mark and sweep. In the mark phase all reachable objects are marked. In the sweep phase the unmarked objects are freed. During the mark phase the execution of the program must be halted, this is known a stop-the-world. It is clear that stop-the-world phases are particularly undesirable in a multi-core multi-threaded environment, because all threads must be stopped. It would be advantageous if the garbage could be collected for a single thread. However, this is not simple since the many threads share the same address space. An object that is accessed by more than one thread is said to be escaped. The key to handle allocation and deallocation of objects by individual threads is an accurate prediction of likelihood that a particular object will escape. If a significant percentage of object allocations can be handled by the thread, then the number of stop-the-world phases could be reduced. From the analysis of benchmark programs, we show that a significant number of objects do not escape. Furthermore, we show that for some objects it is possible to predict their likelihood of escaping with a great deal of accuracy. The runtime costs of such predictions are analyzed.