VM Internal Object Pools

Konstantin Nasartschuk / Aleksandar Micic / Kenneth B. Kent
University of New Brunswick, IBM Canada
Faculty of Computer Science
kons.na@gmail.com / aleksandar_micic@ca.ibm.com / ken@unb.ca

Motivation

Virtual machines are used by many programming languages in order to provide system independence, automatic memory management and other functions. Compared to languages that do not require a virtual machine nor utilize automatic memory management, performance is still a key disadvantage.

Research in virtual machines often aims to close this gap by either improving allocation, memory management, heap structure or parallel execution.

The approach presented aims to improve execution time by creating a pool of allocated objects of a certain class. Those instances are provided to the user application and reclaimed again before they are collected by the garbage collector.

Background

The object pool is a frequently used software engineering pattern, which is most useful when the creation and deletion of an object is more costly than keeping it alive until it can be used again. Most known examples are the thread pool and database connection pool. Both these resourses are very costly to create. Whenever a pool object is needed, the developer requests an instance and has to return it once its work is finished.

The cost for using the pattern adds additional complexity, which is usually avoided by using automated memory management. Developers have to request objects and return them to the pattern. If the developer fails to do so, the object might be lost and the application could run into a deadlock.

Problem

In order to create objects pools based on classes within the VM, three parts have to be changed: vm start-up. the allocator and the garbage collector.

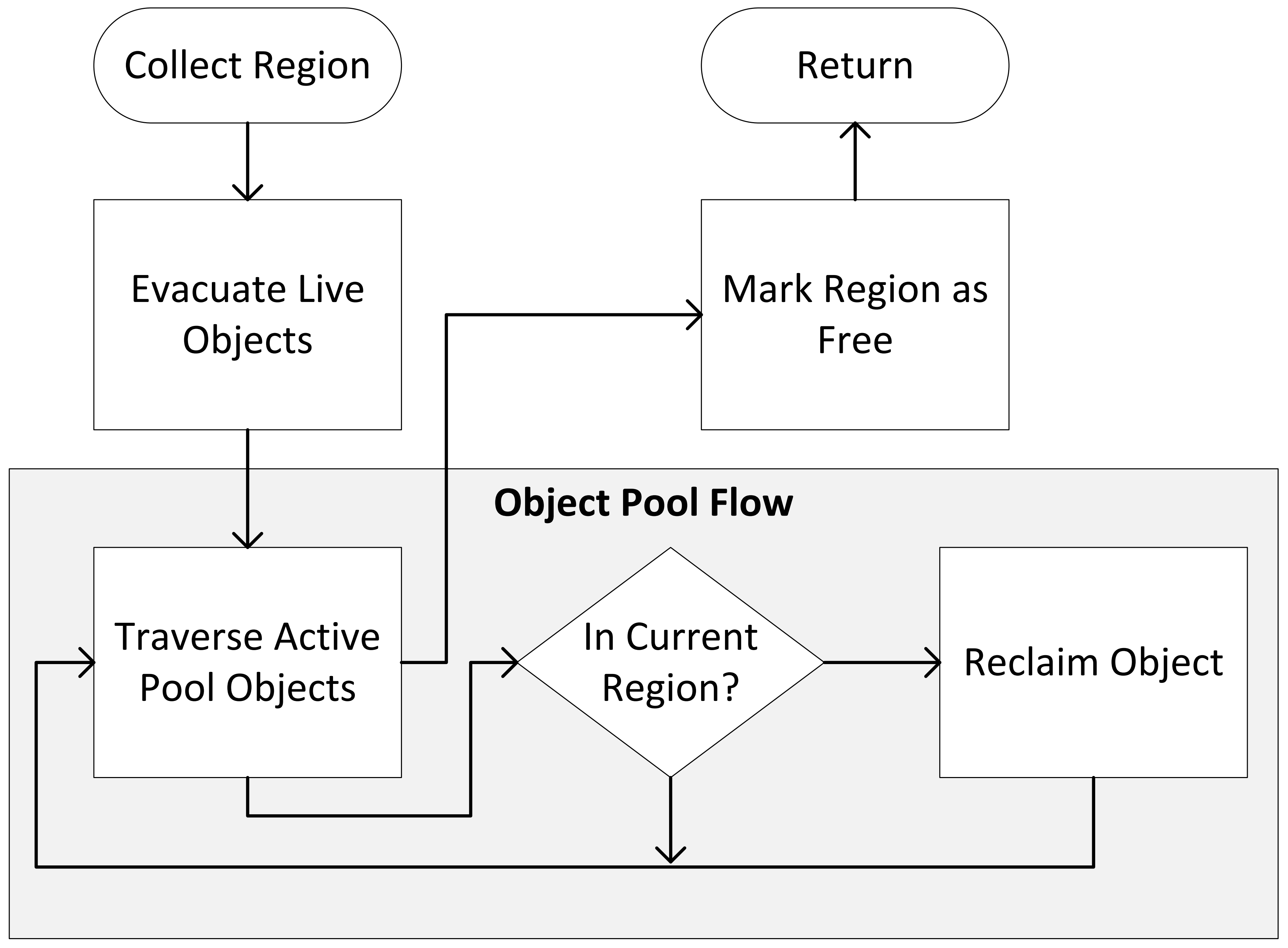
The startup has to be changed to include an initialization phase for the object pools. The data structures have to be created and prepared to be able to handle pool operations during execution time. The allocation process has to detect allocations of pooled classes and redirect the requests to the object pool data structures.

The main problem of the project is located in the garbage collector. As memory is reclaimed, garbage collectors traverse live objects and evacuate them to another region before declaring the collected region as free. Unreferenced objects are never touched. In order to reclaim objects that have to be returned to the pool, an additional stage has to be introduced.

Proposed Solution

The solution under development includes a dynamically growing pool, which is populated during run time. The pool pattern is to be completely integrated into the VM and to be activated by the user using the command line. The parameters of the object pool is a list of classes and the pool size.

During startup, a table of potential objects is created, which is capable of holding the maximum number of object instances. In order to save time, objects are not allocated during start up. Instead, the pool is filled with objects as the user application requests more and more instances. Pooled objects used by the user application are kept in a separate data structure, which is used to determine when objects can be reclaimed before regions are marked as free. The changes to the GC flow are shown below.


Collect Region
Return
Evacuate Live Objects
Mark Region as Free
Object Pool Flow
Traverse Active Pool Objects
In Current Region?
Reclaim Object