Technical Disclosure Commons

Defensive Publications Series

February 29, 2016

Back-traced Garbage Collection

Liam Appelbe

Back-traced Garbage Collection

Authors: Liam Appelbe

ABSTRACT

This disclosure describes back-traced garbage collection in a computer. A back-traced garbage collector searches backwards from an object in an object graph, until a root node is encountered, or until there are no further objects to search. If a root node is not encountered, the searched objects are unreachable and are deleted. The garbage collector can run incrementally, process portions of the object graph, and determine reachability of individual objects without examining the entire object graph. The garbage collector has low latency. The garbage collector is tunable, for example, in response to program characteristics and performance requirements.

KEYWORDS

- Garbage collection
- Mark-sweep
- Memory management
- Back tracing

BACKGROUND

Garbage collection techniques reclaim garbage i.e., memory that is occupied by objects that are no longer in use by a computer program. Most current garbage collectors involve variations of mark and sweep techniques.

Garbage collectors based on mark and sweep scan the entire heap (or some partition of the heap) before determining that an object is unreachable garbage. This places limits on incremental operation of such garbage collectors. In mark and sweep, the garbage collector searches from root pointers (i.e., all pointers in the call stack or in global variables, etc.) to find and mark every reachable object. Once the search is complete, a mark-and-sweep garbage collector scans all objects again and deletes objects that were not marked. These techniques have two main drawbacks. First, the program (e.g., an application program that allocated the objects in the heap) must be halted during garbage collection, since it cannot

cope with the object graph being mutated during the search. Second, every reachable object must be examined by the garbage collector before any objects are deleted.

Tri-color marking is a variation of mark and sweep that addresses the first drawback by introducing a three state marking system. In operation, garbage collectors mark objects as white (not yet reached), gray (reached, but not yet processed), or black (reached and processed). Tri-color marking starts with all objects marked as white, except for the roots which are marked gray. In each iteration, the garbage collector picks a gray object, marks it black, and marks all the objects it points to as gray. This preserves the invariant that no black object ever points to a white one. Therefore, once there are no gray objects left, the garbage collector concludes that all the white objects are garbage.

Tri-color marking can be run incrementally. However, such garbage collectors can encounter significant pauses. For example, the entire reachable set has to be processed by such garbage collectors before any objects can be deleted. Further, the requirement of reasonable bounds on the heap size limits how incremental the search can be.

Some modern garbage collectors combat these issues with more complicated optimizations such as the use of multiple threads to perform the search, or running the search concurrently with the rest of the program. Another current technique is generational collection. Under generational collection, the garbage collector divides the heap into young and old generations, and runs different policies on each. These optimizations add complexity, and do not necessarily solve the underlying problem. Garbage collectors are usually optimized for specific use cases, balancing latency, throughput, and heap size.

DESCRIPTION

This disclosure describes techniques for back-traced garbage collection in a computer. A computer that implements the techniques includes one or more processors and memory. The one or more processors execute instructions stored as software. In different examples, the garbage collection techniques of this disclosure can be implemented as part of an operating system, part of a run-time environment, part of a sandbox, or other computer programs. The garbage collection techniques receive as input an object graph. The object graph includes one or more objects stored in the memory of the computer. In some implementations, the techniques identify and mark objects that can be deleted without affecting operation of the computer. In some implementations, garbage collection includes deleting the identified

objects e.g., by marking memory addresses corresponding to the object as unused, by overwriting memory addresses corresponding to the object, etc.

The techniques eliminate the requirement to search the entire object graph before deletion of any objects. A garbage collector that implements these techniques can examine parts of the object graph and determine the examined objects to be reachable or unreachable, without scanning the entire object graph. The garbage collector can run the search fully incrementally.

Back-tracing example

Fig. 1 shows an example heap with details of back-traced garbage collection. Fig. 1a shows an initial heap. The heap has two root objects A and B, and other objects C-P. Back-traced garbage collection can begin from any object in the heap. In some examples, the starting object may be picked randomly. In some examples, the starting object may be picked based on an age of the object.

In the example shown in Fig. 1b, garbage collection begins from object I. Back-tracing from object I towards object D eventually finds root object A. Therefore, the subgraph involving objects D and I is determined to be reachable. In the example shown in Fig. 1c, garbage collection begins from object P. Back-tracing from P towards object N finds N and terminates at object J. Since no root object is found, it is determined that the subgraph that includes P, N, and J is unreachable from a root object, and can be deleted. Fig. 1d shows the heap after the sub-graph is deleted.

Overview of back-tracing technique

During execution, each object in the heap maintains a list of every pointer that points to it. The garbage collector keeps a list of every object it manages. These lists introduce memory overhead, but are maintainable in O(1). The technique performs an incremental depth first search from each selected object back along the list of incoming pointers for that object. Each iteration of the search processes a single object as follows:

1. If the search stack is empty, the garbage collector picks an object from the global object list, marks the object as visited, pushes it to the search stack, and adds it to the list of visited objects.
2. The garbage collector pops an object from the search stack.

3. The garbage collector determines, for each pointer pointing to the object, if the pointer is from a root object.
    a. If the pointer is from a root object, the garbage collector determines that the sub-graph is reachable. If the sub-graph is reachable, the garbage collector clears the "visited" flag of each object in the list of visited objects. Further, the garbage collector empties the list of visited objects and the search stack, and ends the iteration.
    b. If the pointer is from an object that is not visited, the garbage collector marks it as visited. Further, the garbage collector pushes the object to the search stack and adds it to the list of visited objects.
4. If the search stack is empty, the garbage collector determines that the search completed without finding a root object. Therefore, the garbage collector concludes that the subgraph is garbage, deletes all the objects in the list of visited objects and empties the list.

Each back-tracing iteration processes a single object. Therefore, a search that successfully finds and deletes a subgraph of N unreachable objects takes N iterations to complete. On average, the number of deleted objects per iteration is one.

Garbage collector in operation

In some implementations, whenever a new object is allocated, one or more iterations of search are performed. Performing such iterations can ensure that the ratio of unreachable objects to reachable objects (i.e., the garbage ratio) remains with a range around a steady state. With a greater number of iterations performed per allocation the steady state garbage ratio is lower.

In one example, two iterations of the search are performed each time an object is allocated. If there is no garbage, the garbage collector does not delete any objects and the heap grows at a rate of one object per allocation. However, if the heap is mostly garbage, a majority of iterations of the garbage collector result in deletions. Therefore, as one object is allocated, an average of two objects are deleted. In this case, the heap shrinks at a rate of one object per allocation. Thus, in this example, the garbage collector acts as a feedback controller, with a steady state where the heap includes approximately equal numbers of reachable and garbage objects i.e., a 1:1 garbage ratio.

Back-traced garbage collection state machine

Fig. 2 shows an example state transition diagram for a garbage collector that implements back-traced garbage collection techniques of this disclosure. A garbage collector that implements steps described in the overview described above may encounter pauses. For example, pauses may be encountered when the number of objects searched is large. For example, pauses may be encountered at step 3a described above, where the garbage collector determines that the sub-graph is reachable. In another example, pauses may be encountered at step 4 described above, where the garbage collector determines that the search completed without finding a root object. In some cases, pauses may also be encountered in the loop in step 3 described above, where the garbage collector determines if an incoming pointer is from a root object, when a searched object has a large number of incoming pointers.

In a garbage collector that implements the state machine illustrated in Fig. 2, such pauses are eliminated. As shown in Fig. 2, the garbage collector operates in five different modes. Each iteration of the garbage collector begins in one of the modes, processes one object or reference, e.g., a pointer, and transitions to a particular mode for the next iteration.

1. **Initialize mode:** In the initialize mode, the garbage collector chooses an object to search. For example, if the search stack is empty, the garbage collector picks an object to begin the search from. If the search stack is not empty, the garbage collector pops an object from the search stack. After selection of the object, the garbage collector sets the current reference to be processed to the first incoming reference to the selected object. The garbage collector then transitions to search mode.
2. **Search mode:** In the search mode, the garbage collector processes a single reference to the current object (e.g., the object selected in the initialize mode). If the current reference is a root, the garbage collector transitions to clear mode. If the object that the reference comes from is not yet visited, the garbage collector marks the object as visited, pushes it to the search stack, and adds it to the list of visited objects. If there are more references to the current object, the garbage collector sets the current reference to the next one in the list of incoming references and remains in search mode. Otherwise the garbage collector determines that the search for the current object is completed. The garbage collector then determines if the search stack is empty. If the search stack is empty, the garbage collector transitions to the finalize

mode. If the search stack is not-empty, the garbage collector transitions to initialize mode.

3. **Clear mode:** In the clear mode, the garbage collector clears the visitation flag on a single object from the list of visited objects. If the processed object is determined to be the last object, the garbage collector empties the list of visited objects and transitions to initialize mode.
4. **Finalize mode:** In the finalize mode, the garbage collector calls the destructor of a single object from the list of visited objects and removes it from the global list of objects. If the processed object is determined to be the last object, the garbage collector transitions to destroy mode.
5. **Destroy mode:** In the destroy mode, the garbage collector deletes a single object from the list of visited objects. If the processed object is determined to be the last object, the garbage collector empties the list of visited objects and transitions to initialize mode.

Each iteration of the garbage collector is O(1). However, more iterations may be needed to maintain the garbage ratio at 1:1. The garbage collector completes a reachable search of N objects with an average of r incoming references per object in N(r + 2) steps- N initialize steps, Nr search steps, and N clear steps). The garbage collector completes an unreachable search in N(r + 3) steps- N initialize steps, Nr search steps, N finalize steps, and N destroy steps. In this example, maintaining the garbage ratio at 1:1 requires 2r + 5 iterations per allocation, where r is the ratio of total references to total objects.

Advantages of back-traced garbage collection

The back-traced garbage collection techniques of this disclosure can maintain consistent performance in different situations, in terms of mean latency, mean throughput, and mean waste. The search of the object graph can be run completely incrementally, with only a few steps performed at a time. Also, only a small portion of the object graph need to be examined to make a determination of whether the examined objects are reachable from root nodes i.e., to determine whether the examined objects are garbage or not. The techniques also have clear and predictable relationship between throughput, waste, and the number of iterations per object allocation.

In some implementations, the number of iterations per object allocation can be selected based on the application program. For example, the number of steps can be selected

on the fly, as the garbage collector is in operation, based on the memory usage of the program. Less memory usage may be achieved, for example, by executing more iterations of back-traced garbage collection per object allocation. In another example, the garbage collector steps can be performed in larger chunks e.g., during a downtime of the program.

Back-traced garbage collection techniques of this disclosure permit garbage collection to performed incrementally, by scanning small sections of an object graph. These techniques can determine reachability of objects without examining the entire graph. Garbage collectors that implement these techniques can be run in small increments and reduce pauses due to garbage collection. Further, such garbage collectors can maintain predictable heap size. Also, the pause time is independent of the size of the heap. Garbage collectors that implement these techniques can achieve low latency. Such techniques may be effective, for example, in situations where latency minimization and consistency of garbage collector performance are priorities.

FIGURES

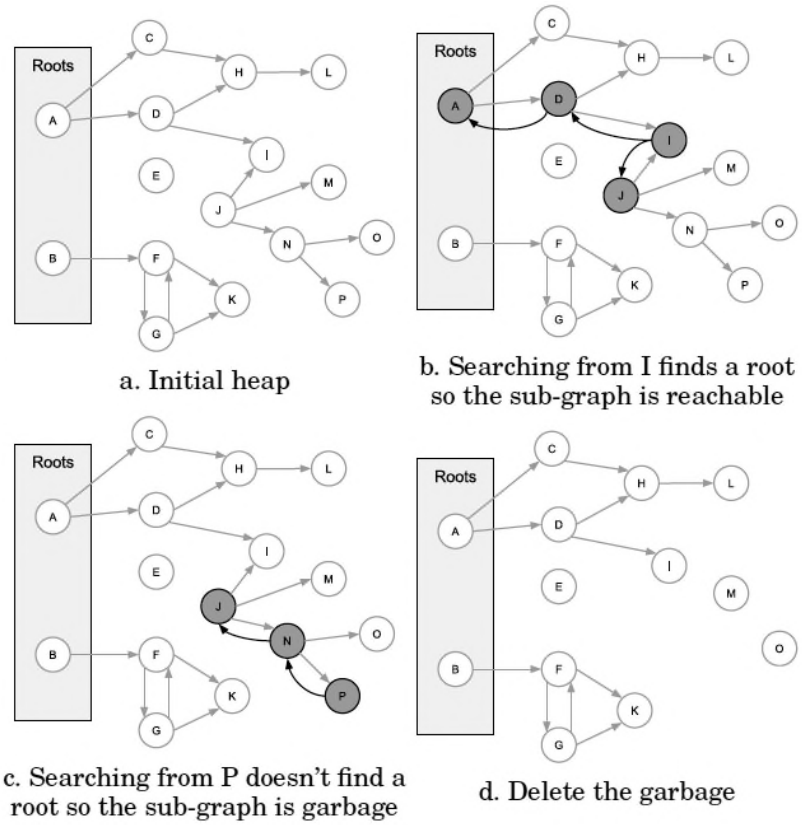

Fig. 1

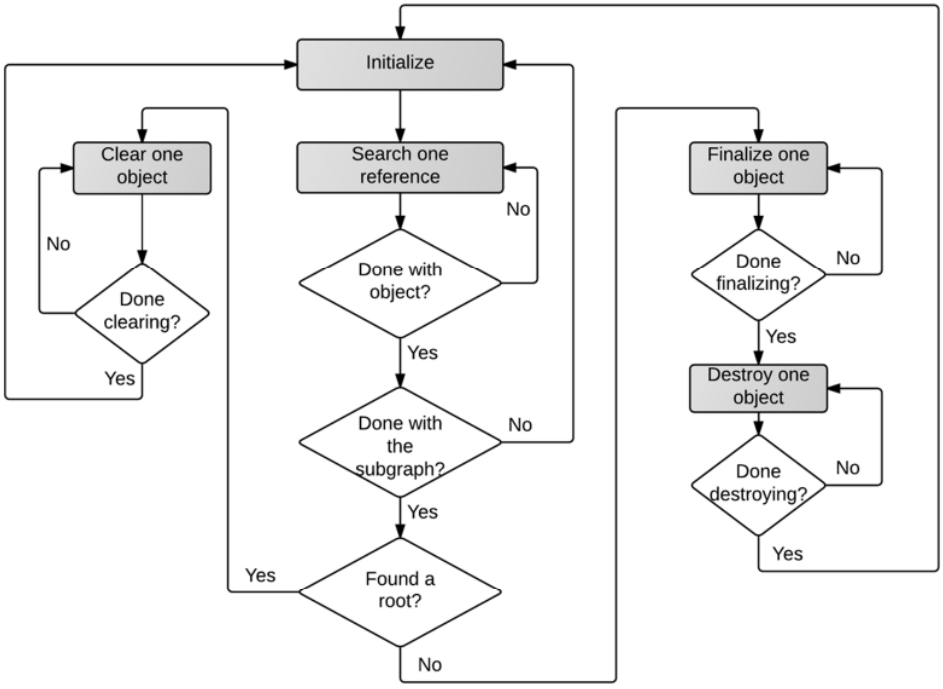
Initialize
Clear one object
No
Done clearing?
Yes
Search one reference
No
Done with object?
Yes
Done with the subgraph?
No
Yes
Found a root?
Yes
No
Finalize one object
Done finalizing?
No
Yes
Destroy one object
Done destroying?
No
Yes

Fig. 2