Abstract

Different from traditional garbage collection (GC), in actor system GC identifying whether the object is garbage or not depends on not only its references but also its processing state. It makes garbage collection more complicated. In previous work tracing algorithms such as mark-sweep are utilized which lead to GC too complicated to be suitable for large-scale distributed system. We present in this paper a hybrid distributed garbage collection which is comprised of local GC and global GC. A reverse reference is added for each reference which is from an actor that can be activated (include remotely referenced actors). Then we can utilize tracing algorithm to collect local garbage. We use timestamp algorithm and backward inquiry to identify and collect global garbage. This GC scheme can collect all garbage correctly. Furthermore, it is fault tolerant and suitable for large-scale distributed system.

1. Introduction

In distributed object-oriented run-time environment, the active entities may be heavyweight processes or lightweight threads encapsulated within an object, that is active objects (actors). The distributed system includes actors is called actor system. Since programmer-controlled memory management is error-prone, automatic memory management that is called garbage collector is used to detect and reclaim unneeded objects without the programmer’s intervention.

However, different from traditional GCs that manage static objects, management of live objects makes garbage collection more complicated. Two criteria often used in garbage collection such as traversal criterion and reference counting [1,2] do not properly detect garbage in actor system. Because the topology of the system alone is not sufficient to discriminate between garbage and nongarbage actors. The processing state (active, blocked) of an actor is also critical. Some GC algorithms [3,4] are proposed and they can correctly collect garbage actors. However, tracing algorithms used lead to their GCs too complicated to be suitable for large-scale distributed system. [3] is not fault tolerant.

We propose a hybrid distributed GC scheme, which uses different algorithms for local GC and global GC to manage live objects. For each nodes, local GCs can work independently and concurrently to reclaim local garbage. We add a virtual inverse reference for each reference from an actor that can be activated. Thus, tracing algorithm can be used to reclaim local garbage actors correctly. To be suitable for large-scale distributed system, we use timestamp algorithm for global GC and Ryu’s backward inquiry approach can be utilized for confirmation of distributed garbage. This algorithm can reclaim all cyclic garbage and it is fault-tolerant.

The paper is organized as follows: Section 2 describes our system model. Section 3 gives the detail description of our algorithm. Section 4 discusses the main properties of our distributed GC. Finally, conclusions are given in section 5.

2. Distributed garbage collection in actor system

2.1. System model
In actor system, an object (actor) is an entity composed of data and of one or several threads of control that operate on that data. If actor A knows the name of B, we call A has B as an acquaintance and A is the inverse acquaintance of B. The state of an actor is either active or blocked. Active actors can send messages to acquaintances and blocked actor can be activated by messages. If mail queue is empty, the actor is blocked. Figure 1 shows the actor system model as used in [3] for comparison with other algorithm.

**Figure 1. An actor system**

A root actor is an actor that is never garbage. A garbage actor is an actor that is not a root actor and in all reachable states, cannot communicate with a root actor. Different from an object in system model of traditional GC, an actor has processing state that should be taken into consideration when GC works.

For example, in view of traditional GC, I is garbage because no root can reach it. However, I is not garbage in actor system as it is active and can send a message to root H, then H can reach I. Similarly, actor G is not garbage in actor system, consider the following sequence of actions:

- A sends its own address to B;
- E sends the address of D to G;
- G sends a message, MSG to D;
- D sends MSG to C;
- C sends MSG to B;
- B sends MSG to A;

As shown in figure 1, O and N are garbage obviously since they are not roots and are isolate from other actors. K and J are also garbage (cyclic garbage) as no root can reach them. M and N are garbage though they have name addresses of roots. Since they are blocked and cannot be activated by other actors, they cannot communicate with roots. Other actors are all nongarbage as mentioned above. Figure 1 shows that GC for actor system should check processing states of actors when it identifies and reclaims garbage.

### 2.2. Related work

Many distributed garbage collection algorithms are proposed while most of them process static objects rather than live objects presented in this paper. Objects that are not accessible by roots are identified as garbage to be reclaimed by GC.

Rudalics [5] presented a distributed GC based on a copying local GC. It operates on the whole space and it is not fault-tolerant. The global GC cannot be completed until all local GCs are finished. If there are lots of nodes in system, response time of global GC will be too long. So it is not suitable for large-scale distributed system. Marking-tree collector [6,7] has the same problem as Rudalics’s copying distributed GC. Lang et al. [8] proposed partitioned GC based on the concept of a group of processors. Network is divided into groups. Lang’s algorithm can collect cyclic garbage and is fault-tolerant. However, in order to see if all nodes finish the local marking, a distributed termination detection algorithm is used and the overhead is extremely heavy. Ryu [9] proposed a distributed GC by timeouts and backward inquiry. His scheme can collect distributed cyclic garbage safely and completely without global synchronization or backward references. We also utilize Ryu’s backward inquiry approach in our paper.

Puaut [10], Halstead [11] and Kafura [3] et al, proposed distributed GC models of active objects. In Puaut’s algorithm, local collectors are independently and global synchronization is avoided while the global collector is centralized and, therefore, susceptible to congestion at the global collector’s node. Halstead’s garbage collector for distributed actors uses the concept of an actor reference tree. However, the algorithm cannot detect cyclic garbage. Kafura’s approach can collect all garbage actors including cyclic garbage. The mutators, the local collectors and the global collector run concurrently without global synchronization. His scheme utilized marking-sweep algorithm to detect global garbage. Thus, it is not suitable for large-scale distributed system. Furthermore, his approach is not fault tolerant.

### 3. Algorithm description

#### 3.1. Hybrid distributed garbage collection

In actor system, active actor can send its name address to referenced actors and activate blocked actors. For example in figure 1, though root H cannot reach
actor I, I can sent its name address to H since I is active. Therefore, H can reach I and I is not garbage. Similarly, though actor B, C and D are blocked and not accessible by root A, they can be activated by root A or active actor E. Then they can send their name addresses to A which lead to B, C, D, E and G are all nongarbage. We can conclude that if actor X has a reference to Y (X has an acquaintance as Y) and X is active, it implies that Y has a reference to X. Thus, in the topology of the system, we can regard all actors that are accessible by active actors as active actors and they have virtual inverse reverse references to referencing actors. In figure 1, if we add inverse references orderly from active actor E as D->E, G->E, C->D and B->C, we can use tracing algorithm from root A to identify that B, C, D, E, G are all accessible by root A. Remotely referenced actors are regarded as active actors because local garbage collector cannot know whether they are referenced by active actors or not. Consequently, all these actors and reachable actors are remained by local GC and to be checked and reclaimed by global GC.

We add a virtual inverse reference for each reference from active actors and the referenced actors are regarded as active actors and are processed similarly. Then we obtain figure 2.

![Figure 2. Add inverse references in actor system](image)

In figure 2, if we use tracing algorithm to traverse all actors from roots, actors B, C, D, E, G and I will be identified as nongarbage correctly as mentioned in section 2.1. Actor F, L, M, N and O are still not reachable from roots and therefore are identified as garbage to be collected by local GC. Cyclic garbage K and J will be retained to be collected by global GC. In addition, as tracing algorithm can check and reclaim cyclic garbage, local GC using tracing algorithms can correctly collect all local garbage.

As previous work, our GC collector is comprised of independent local collectors, one per node, and a distributed global collector. The mutator, the local collectors and the global collector run concurrently. Differently, most of GC models for actor system such as [3,4] used tracing algorithms (such as mark-sweep). We use tracing algorithm in local GC while timestamp algorithm in global GC. Though tracing algorithms can collect all garbage include cyclic garbage correctly, they do not scale well to distributed systems as a distributed termination detection algorithm will be used and the overhead is extremely heavy.

We use timestamp algorithm in global GC. When local GC works, a new timestamp is created which is equal to current time and is propagated along chains of remote pointers. To be noticed, here `remote pointers' includes virtual inverse references added. Consider figure 2, when local GC in node 0 executes, root A creates a new timestamp and propagates it to B, C and D. After this GC cycle, node 0 sends message to node 1 and the timestamp included is propagated to E and G in node1. A Lampert clock is used to simulate global time at each participating space. In addition, if more than two references with different timestamp to an actor, maximum timestamp is selected to be propagated. Thus, when local GC is completed, timestamps of all reachable actors can be refreshed while that of garbage actors cannot increase as they cannot communicate with roots. We can set a threshold that all actors whose timestamp is smaller than the threshold should be regarded as garbage to be reclaimed by global GC. Considering that each system may select a different threshold, we can utilize Ryu’s backward inquiry approach [9] to confirm whether suspected actors are garbage actors or not.

To describe our GC scheme clearly, we choose a copying collector [12] for discussion and give detail description of algorithms of local GC and global GC in the following two sections because copying algorithm is easy for analysis. Moreover, it can compact memory space when it works and, therefore, to avoid impact of memory fragment.

### 3.2. Local garbage collection

Since our garbage collector works in a distributed actor system, we will make some change of traditional copying GC [12]. As mentioned in section 3.1, for local GC all remotely referenced actors should be regarded as active actors. To avoid reclaim nongarbage actors incorrectly using tracing algorithms, a virtual inverse reference should be added for each reference from an actor that can be activated. In actual algorithm, to avoid duplicate work, local GC first scans roots and remotely referenced actors and moves reachable actors from them to tospace. Then all remaining actors will be processed by the method described above. The model is shown in figure 3 and detail algorithm is in the following.
Figure 3. Our copying GC model

1. scan root table and remotely referenced actors table and move reachable actors from fromspace to tospace; meanwhile, a new timestamp is created for roots and timestamps are propagated when actors are scanned;
2. scan remaining actors in fromspace; mark reachable actors from active actors as ‘can be activated’ in a bit; add a inverse reference for each reference from a ‘can be activated’ actor; continue these steps till all reachable actors from active actors are marked;
3. if a reference from tospace to fromspace or a remote reference is added, it will be traversed to move all reachable actors from this reference to tospace;
4. no more actors will be evacuated to tospace and flip happens;
5. messages including information of referenced actors and timestamp are sent to referenced nodes (including those nodes referenced by inverse virtual references added);

3.3. Global GC

The purpose of global GC is to collect distributed garbage. As described in section 3.2, local GC can reclaim local garbage while it cannot detect distributed garbage. Thus, remotely referenced actors are all regarded as active actors to be retained. For example in node 0 of figure 2, if the reference A->B is deleted, actor B, C and D will become garbage. However, local GC cannot collect them because local GC will regard actor D as active as D is remotely referenced. This work is left to global GC. After reference A->B is deleted, no roots can reach B, C and D. Each time local GC executes, the timestamp propagated among them cannot be refreshed and finally stabilized to be a fixed value just as that of cyclic garbage actor K and J. Therefore, a threshold can be set that if an actor’s timestamp is smaller than the threshold, the actor can be marked as garbage to be collected. Considering that selecting a well-suited threshold may be difficult, we can utilize Ryu’s backward inquiry approach to confirm whether the suspected actors are garbage or not. Ryu gave detail description of the algorithm in his paper.

4. Performance discussion

An available distributed GC should have basic functions such as collecting all unused actors includes cyclic garbage. To be suitable for large-scale distributed systems, GC should discard centralized control and global stop-the-world synchronization. Furthermore, our distributed GC has some other desirable properties compared with other algorithms such as [3]:

1. resources consumption

Previous work such as Kafura’s approach did not consider resources consumption. Especially in his paper, in the worst case on the order of $K^2$ total messages must be sent to garbage collect a distributed actor system of $K$ actors. However, in our algorithm messages for remote references and verification can be bundled together to be sent after each GC cycle. Even the incremental overhead of backward inquiry is low because messages necessary for the inquiry are bundled with the messages used for refreshing. Furthermore, memory usage and computation time is not high compared with other algorithms.

2. fault tolerance

In Kafura’s scheme, mark-sweep algorithm that is not fault-tolerant is used in global GC. Thus, if any node is unavailable, global collection is prevented from occurring. Our global GC uses timestamp algorithm. Unavailability of a node only has impact on those actors referenced by actors in this node while roots in other nodes continue to refresh their reachable actors as usual. Thus, global GC will not be prevented and distributed garbage that is not relevant to the unavailable node will be collected normally. It demonstrates our garbage collector is fault-tolerant.

Furthermore, since remote references are recorded, each node knows not only referenced actors but also referencing actors. Therefore, it is easy for application administrator to find out the nodes that are referenced by the failed node and to take appropriate action for recovery.

3. suitable for large-scale distributed system

Our algorithm is well suited for very large nets of nodes. On the one hand, we abandon traditional tracing algorithm such as mark-sweep for global GC and use timestamp algorithm. When Kafura used mark-sweep in his global GC, a distributed termination detection
algorithm was used and the overhead was extremely heavy if there were many nodes in distributed system. Timestamp algorithm does not require GC traverse all nodes all at once. On the other hand, our algorithm is fault tolerant as mentioned above. Even though some nodes cannot work in distributed system, our garbage collection will continue to work as normal. Therefore, our algorithm is suitable for large-scale distributed system.

5. Conclusion

We have described an efficient distributed garbage collection mechanism for actor system. The mutator, the local collectors and global collector run concurrently. Local collector adds inverse references and uses tracing algorithm to reclaim local garbage. Global collector uses timestamp propagation and backward inquiry, which can collect distributed garbage safely and completely without global synchronization. Our garbage collector is fault tolerant. Some nodes failure will not stop global GC’s work. Moreover, messages necessary are bundled together and are sent to referenced nodes after local GC completes. Our algorithm does not need traverse nodes all at once. Consequently, our garbage collector is suitable for large-scale distributed actor systems.

Though we propose hybrid distributed garbage collection of active objects, scheduling analysis and memory analysis need to be made more precise. Future work will focus on it and implementation of our garbage collector model.

References