CS263: Runtime Systems

Distributed Garbage Collection Report

Team Members: Fuheng Zhao, Mengjia Zhao

Problem and Objective

With the rise of big data in companies such as Google, Amazon, and Facebook, managing terabytes to petabyte of data and serving millions to billions of user requests has become a necessity for day to day operations. To enhance the scalability of the system and to minimize the user queries latency, companies have shifted from central service storage architectures to distributed storage architectures and placed the data closer to the users. A fundamental problem in distributed storage architectures such as distributed object systems is the necessity of the distributed garbage collection algorithm. Garbage not only consumes storage spaces but also degrades performance. If a huge amount of garbage is not collected, then it decreases the data storage locality and may poison the storage cache.

An object is live if the object is reachable from at least one server's root and it is dead if it is not reachable from any server's root. The distributed garbage collection algorithm needs to ensure safety and liveness, meaning no live objects should be claimed and eventually all dead objects should be removed. A popular approach in garbage collection is the reference counting. However, reference counting does not handle cyclic dependencies and in distributed object systems cycles are frequent [1]. As a result, the distributed garbage collection algorithm leverages both reference counting and tracing strategies. Periodically, a node can trigger the cycle detection process and if a cycle is found and all nodes in the cycle are unreachable, then the process can safely claim all these dead nodes. This report is structured as follows: we first discuss our contributions. We then introduce, implement, and analyze two different algorithms, in which one is designed by us and another is designed by Viega and Ferreira. At the end, we summarize and conclude the work.

Contributions

We first came up with a simple DFS solution for cycle detection. This algorithm is correct but relatively slow because it requires servers to wait for responses.

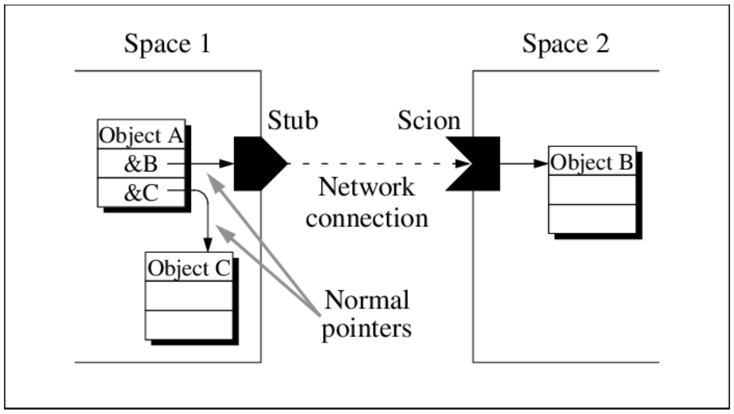
Then we read the paper *Asynchronous Complete Distributed Garbage Collection* written by Lu´ıs Veiga and Paulo Ferreira. They proposed a new distributed garbage collection algorithm, which has a lighter overhead compared to our simple solution, and it doesn't require servers to wait for a response.

Solution 1: Simple Cycle Detection

We first propose a simple cycle detection algorithm. The algorithm takes a sender S, a node X, a process P, and a set of visited nodes. Without loss of generality, we assume each node is uniquely defined by (node, process) tuple. If the node X is reachable from the root then tell S no cycle and abort. If X in Visited then tell S cycle found. Else, add self to set Visited and ask all outside connections (node X', process P') to also start cycle detection. Wait for responses and if any outside party says cycle not found, then tell S that cycle is not found. Otherwise, tell S cycle found. Eventually, the node initiated the cycle detection would know if it's inside a cycle

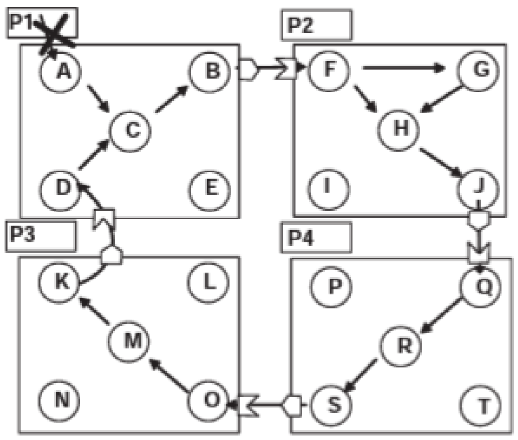
Solution 2: Viega and Ferreira's Algorithm [2]

They described a Distributed Garbage Collection (DGC) algorithm capable of reclaiming distributed cycles of garbage asynchronously and efficiently, which does not require any particular coordination between processes and it tolerates message loss.



For the objects that are referenced by other servers, they define that the *stub* represents the exiting reference from Space 1 to object B of Space 2. The *scion* is the entering reference from object A of Space 1 to Space 2, as shown in figure. The *StubsFrom* list for a scion is defined by the list of stubs, in the same process, transitively reachable from the scion. The *ScionsTo* list for a stub is the list of scions in the same process that transitively lead to the stub.

Their algorithm defines that each Cycle Detection Messages(CDM) passed among servers will contain two sets: a *sourceSet* that holds dependencies for the detection path, and a *targetSet* that holds target objects that the message has been forwarded to.



On arrival of a CDM at a target object, the target object is added to the *sourceSet*. Then the *StubsFrom* list for the target object is calculated and added to the *targetSet*. These objects are marked as the next targets. Then we find the *ScionsTo* list for the next targets, and add them to the *sourceSet*. Then the CDM is forwarded to the next target.

Cycle is found when the *targetSet* and the *sourceSet* are equal.

Implementations

We implemented both algorithms in Python. The advantage of using this scripting language is that it is easy to read, write, and debug the code in the team and enhance our productivity.

We used sockets for communication between servers. All servers run on 127.0.0.1 and their ports are predefined. Each server has an independent thread that runs forever to receive messages from other servers. We used the pickle library to pack the message sent between servers.

We allowed users to create nodes in servers and connect them together. However, it is difficult to implement actual ‘referencing’ of objects, so instead we added an extra field in each node to indicate whether it is referenced by another server.

Finding and Conclusion

By comparing these two algorithms, we found that our proposed algorithm doubles the latencies as it requires waiting for other nodes’ cycle detection responses. The advantage of our proposed algorithm is that it decreases the communication cost by half since only one set of visited nodes need to be kept, whereas in Viega and Ferreira’s Algorithm two sets of nodes need to be maintained. To summarize, we studied garbage collection in distributed environments; we propose a simple solution for cycle detection but require servers to wait for response; we read and understand a distributed garbage collection paper proposed by Veiga et al; we implemented both algorithms in python.

References

[1] N. Richer and M. Shapiro. The memory behavior of the WWW, or the WWW considered as a persistent store. In POS 2000, pages 161– 176, 2000.

[2] Veiga, Luis, and Paulo Ferreira. "Asynchronous complete distributed garbage collection." 19th IEEE International Parallel and Distributed Processing Symposium. IEEE, 2005.