COMPSCI 590S Systems for Data Science **Fall 2017**

Lecture 11: Garbage Collection and File Systems

Lecturer: Emery Berger *Scribe(s): Benjamin Cheung, Rohit Narasimhan*

11.1 Garbage Collection

In the previous class, we discussed the concept of Garbage Collection and two major techniques for GC.

1. We discussed Reference Counting which is an incomplete algorithm due to the presence of cycles.
2. Alternatively, we can use Mark and Sweep to fill in holes. It marks from the roots to the extent that can be reached and then sweeps away the nodes that are unreachable.

11.1.1 Semi-Space

Semi-Space garbage collection (also called Copying or Relocating GC) is an algorithm which works on the concept of dividing all memory into two chunks - a 'from-space' and a 'to-space'.

- Place all active memory into the from-space.
- Use bump pointing to know the area of memory that you insert into.
- When the pointer reaches the end i.e. there is insufficient space to fulfill an allocation, the garbage collection starts.
- We define a 'live' memory object as one that has a pointer to it from some other memory object.
- Iterate through the current from-space and move all live memory objects to the to-space.
- All items left in the from-space are not alive and can be removed. The to-space and the from-space then switch roles.

Thus, a collection consists of swapping the roles of the regions, and copying the live objects from from-space to to-space, leaving a block of free space (corresponding to the memory used by all unreachable objects) at the end of the to-space. Since objects are moved during a collection, the addresses of all references must be updated. This is done by storing a 'forwarding-address' for an object when it is copied out of from-space.

Problems

- The actual memory space is halved. You would need 2X the memory to do the same things. In Mark Sweep, we don't lose the amount of available memory.
- Copying your memory over to the new space takes time and costs CPU cycles.
- After copying, you get Cache Thrashing i.e. everything currently in cache space is a miss to all the new copied memory.

Benefits

- Implicitly compacting the heap. Fragmentation is avoided.
- Allocation costs are extremely low as there is no need to maintain and search lists of free memory
- Locality is high.

11.1.2 Stop-the-world

- Some GC algorithms employ a mechanism called 'Stop-the-world'.
- Stop-the-world garbage collectors completely halt the execution of the program when GC is running and this lasts till the collection is completed.
- This is a reasonably elegant algorithm with simple allocation.
- The major disadvantage is that the program can perform no useful work while a collection cycle is running. It also leads to a slowdown due to the associated cache warm up.

11.1.3 Mark-Sweep Compact

- One of the challenges of the Mark-and-Sweep algorithm was the need for manual compaction.
- On the other hand, in semi-space garbage collectors, the compaction was implicitly achieved by simply laying the objects out in a row.
- Mark-Sweep compaction walks through and compacts after performing GC.
- Thus explicit compaction was still required. It was also much more challenging to code and required complicated fix-up.

11.1.4 Generational Garbage Collection

The background for Generational GC comes from 'Self' - an object-oriented programming language created by David Ungar and Craig Chambers, in which everything is an object! This led to innovations in dynamic compilation through JIT as well as garbage collection. We would typically need to allocate a heap size which is 3-5 times what the program needs and this over-provisioning is necessary for performance. Generational GCs operate on the underlying concept that most objects die young, also called the generational hypothesis.

Working

- New objects are spawned, but placed in a separate buffer area called a nursery.
- Objects are expected to die young and not make it in the nursery. The survivors are copied out into a next generation buffer.
- There are immortal objects as well that live longer than what is ideal. These are never GC'd. Objects that are speculated to be immortal are 'pretenured '.
- The objects in the older generation that may reference objects in new space are kept in a 'remembered set'.

- The code that gets executed after pointer updates is called a a write barrier which ensures that generational invariants are maintained.

In Java, you can specify the number of generations. However, more number of generations can be bad because it has to copy more stuff out.
Generational GCs do not work with Big Data. As objects need to die young, it doesn't work with MapReduce. When you map, a lot of objects are produced after each parallel map phase. In the reduce phase, the objects would finally not be used and thus can be cleared by GC. This breaks the notion of objects dying young.

11.1.5 Concurrent and Incremental GC

- Incremental and Concurrent garbage collectors do not require a 'stop-the-world' when the collector is running.
- Incremental garbage collectors perform the garbage collection cycle in discrete phases, with program execution permitted between each phase (and sometimes during some phases).
- Concurrent garbage collectors are incremental collectors which perform collection in parallel with the program execution.

11.2 File Systems

11.2.1 Traditional File Systems

File Systems have stood the test of time. The traditional file system has a hierarchical structure and a directory tree as the file system layout. Apple's Mac OS was one of the first to have such a file system called the Hierarchical File System (HFS). Most traditional file systems have a POSIX abstraction layer which functins like an API layer. This API is quite narrow meaning you cannot do too much with a file system. Some of the functions allowed using this API layer include :

- create()
- open()
- close()
- read()
- write()

Some of the popular file systems are : ext3, ext4, HFS, XFS, NTFS, ReiserFS, FAT32, and ZFS

11.2.2 Network File System

A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server. Programs using local interfaces can transparently create, manage and access hierarchical directories and files in remote network-connected computers.

11.2.3 Google File System

- Scalable distributed file system for large distributed data-intensive applications.
- Provides fault-tolerance
- Presents unified view of the world to API.
- Consists of a single master and multiple chunkservers.
- Files broken into 64MB chunks
- For reliability, each chunk is replicated on multiple chunkservers.
- The large chunk size offers several advantages. Also, considering the huge size of the data, having small 4KB blocks doesn't make much sense.