#ASLI ENGINEERING

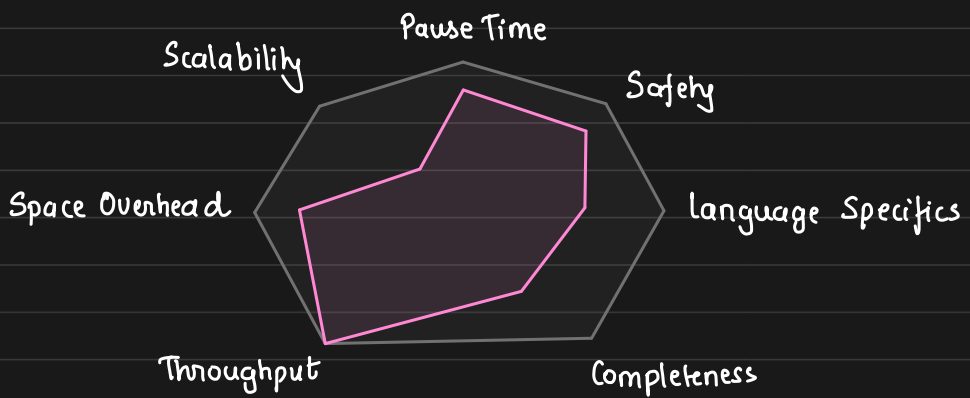
How to pick a Garbage Collector

SWIPE

BY

ARPIT BHAYANI

Metrics of a Garbage Collector



There is no 'best' garbage collector

From a study done on Garbage Collectors in 2000s ↗

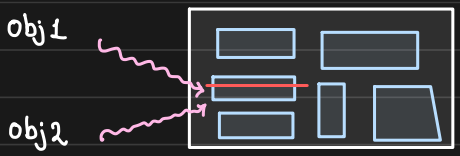
There are many Garbage Collectors and each one is superior than others in atleast one use-case by atleast 15%.

* Most GCs give you a lot of knobs to tune the performance for the load you are handling

- -XX: UseSerialGC
- -XX: UseParallelGC
- -XX: MaxGCPauseMillis
- -XX: GCTimeRatio
- -XX: MinHeapFreeRatio

After a certain scale you will be spending more time tuning these params than coding.

1. Safety collector must never reclaim the storage of live objects.



obj1 and obj2 are pointing to the same location in memory. GC should NOT be deleting & reclaiming the space

* No Dangling Pointers

2. Throughput The time spent in garbage collection should be as low as possible.

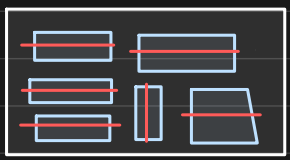
as possible.

User wants "program" + "garbage collection" to execute in as little time

Most GC trades its performance and execution time in favour of higher program exec. throughput

eg: Once a while GC will run an expensive defragmentation phase so as to improve on program's memory allocation performance.

3. Completeness eventually, all garbage in the heap should be reclaimed



Complek cleanup in one shot is not desirable nor always possible
Hence the word Eventual

One cycle v/s many cycles

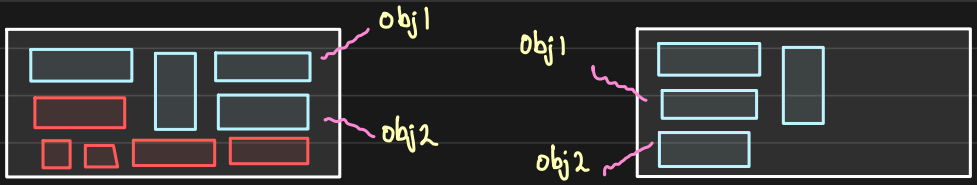
4. Pause Time — Many garbage collectors pause the program execution during cleanup; this pause should be as low as possible

* One of the most important and impactful metric

Most execution of GC do not "stop the world" but some times to maintain the correctness of references it needs to stop the world

Defragmentation

Object Shuffling



Garbage collectors do their best to reduce the Pause Time but it comes with its own set of challenges & complications

5. Space Overhead — GC may require auxiliary data structures to track objects and decide efficiently

but it puts additional load on memory and consumption

Bitmap Tables → To keep a check on objects already considered

Graphs → To manage & maintain object dependency

6. Language Specific Optimizations — A GC may provide language specific optimizations to gain that extra ounce of performance

eg: some languages are pure functional

some languages have only heap allocation

some languages may have explicit de-allocation

some languages have only persistent data structures

A GC exploits its understanding of the language and its constructs to optimize its execution.

* Some GC runs in constant time because of how objects are laid out by the memory manager

7. Scalability — GC needs to leverage the modern hardware capabilities to make its execution faster.

Servers are growing ... 10s and 100s GB of heap and hence GC will have a lot of work to do going through this massive heap

This would increase the GC time and hence GC that always stop the world become inefficient ↖ Favoured

Some GCs have evolved & become Pause free