Modeling Minor Garbage Collection in JVM

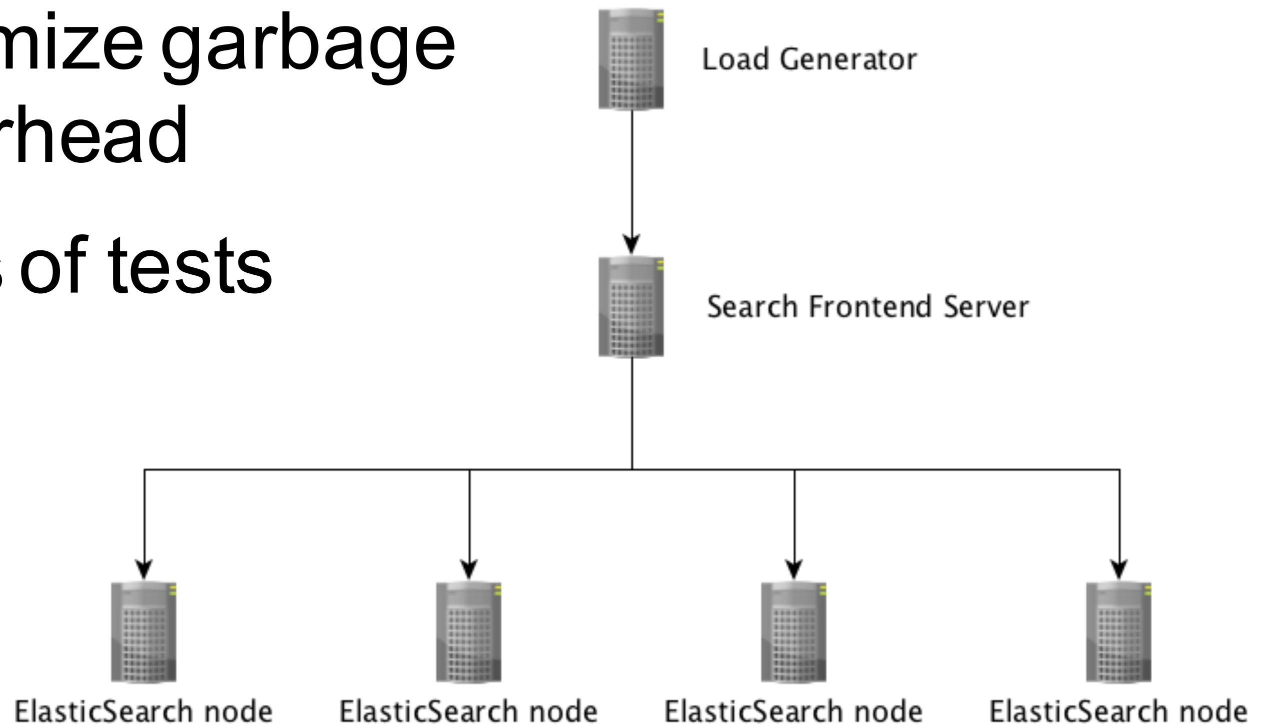
Xiaosong Lou
Steven Xu

Agenda

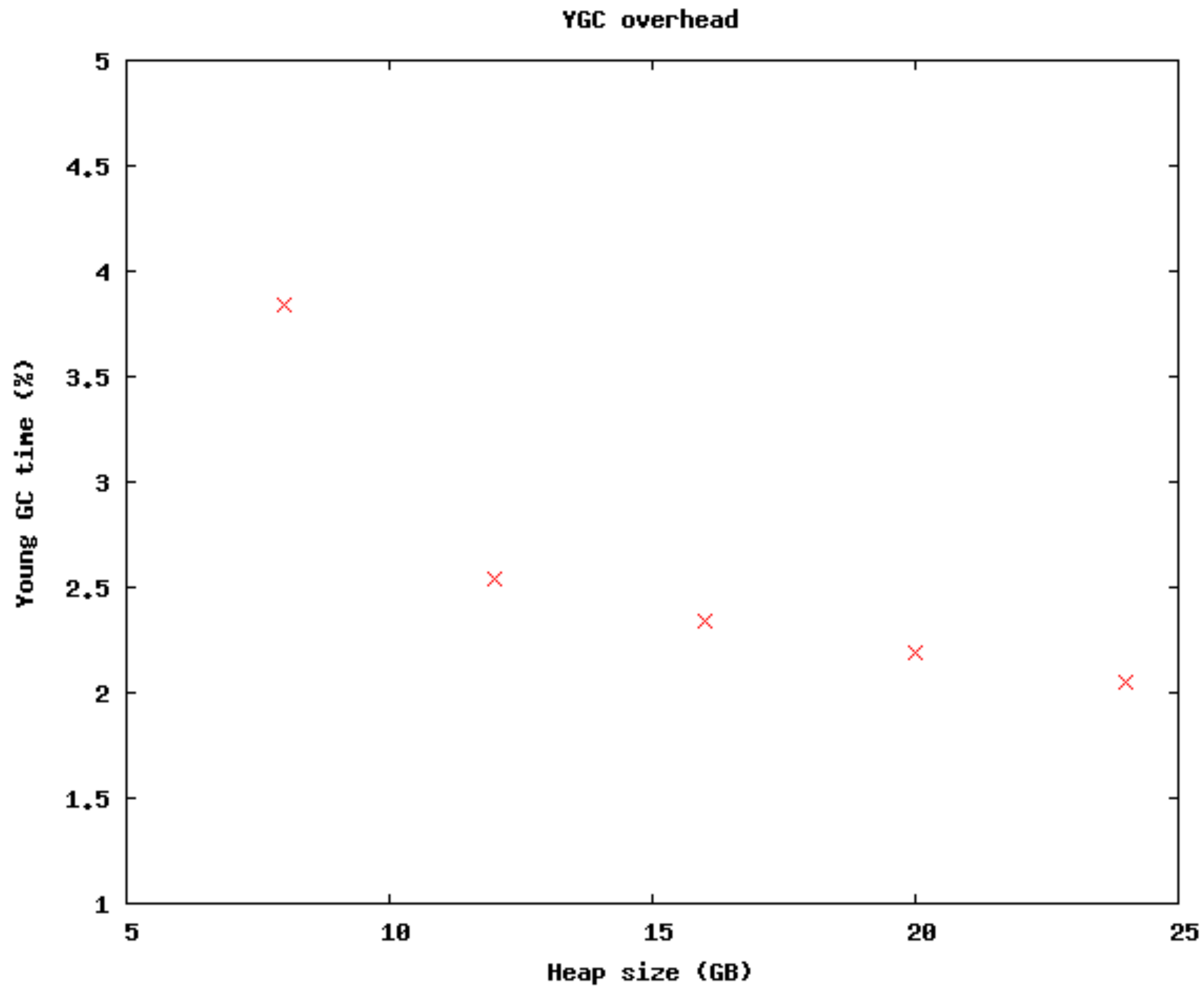
- Our Original Assignment
  - To suggest a set of optimal JVM parameters for an application stack
- Interesting Results
- Modeling GC process
  - The Law of Minor GC Overhead
  - Lower Bound Corollary
- Experiments and findings

The Original Project

- Elastic Search
  - 4 hosts, 40 cores/32GB each
  - 32GB of data
- Trying to minimize garbage collection overhead
- Ran hundreds of tests



First Result



Observations and Questions

- Inversely proportional
- Goes to infinite when heap size is small
- Does not go to 0 when heap size is infinite
    - Why?

From Java Document :

"*The Young Generation is where all new objects are allocated and aged. When the young generation fills up, this causes a minor garbage collection.*"

Garbage Collection Model

- y : young generation size
- s : the amount of memory (in bytes) that survives each YGC.
- r : the speed at which the application consumes memory (object creation speed)
- T : the average interval between two consecutive YGC events

$$T = \frac{y-s}{r}$$

Garbage Collection Time

- *t* : average YGC time.
  - From Java Documentation

    "After a minor GC, when aged objects reach a certain age threshold they are promoted from young generation to old generation."

  - t equals the time to delete objects plus the time to promote objects.

    $$t = t_{release} + t_{promote}$$

    $t_{release} = (y - s)/g_1$ and $t_{promote} = \frac{s}{g_2}$

    g_1 : object release rate

    g_2 : object promotion rate.

Law of Minor GC Overhead

$$O_{minor} = \frac{t}{T} = \frac{(y-s)/g_1 + s/g_2}{\frac{y-s}{r}} = \frac{rs}{g_2}\frac{1}{y-s} + \frac{r}{g_1}$$

$$object\ promotion\ factor\ P = \frac{rs}{g_2}$$

$$object\ release\ factor\ C = \frac{r}{g_1}$$

$$O_{minor} = \frac{P}{y-s} + C$$

Lower Bound Corollary

Minor GC overhead must be larger than the ratio of object creation speed divided by object release speed:

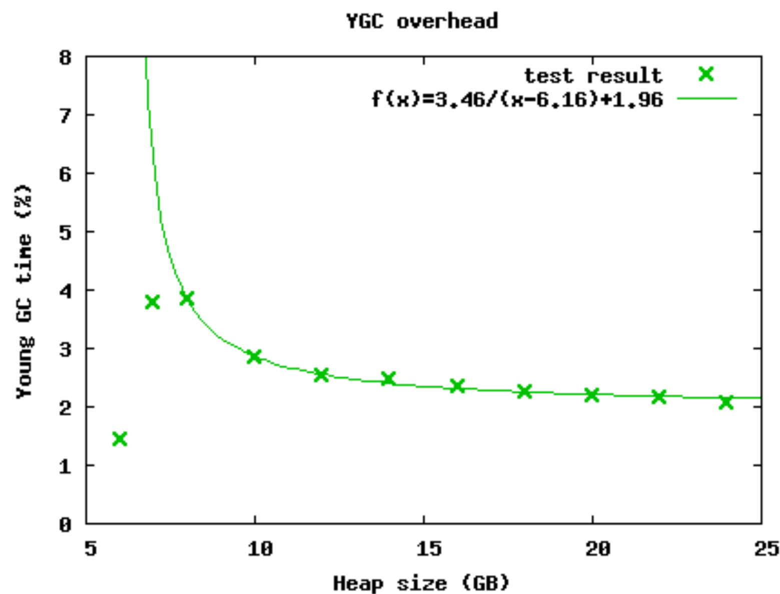
$$O_{minor} > C = \frac{r}{g_1}$$

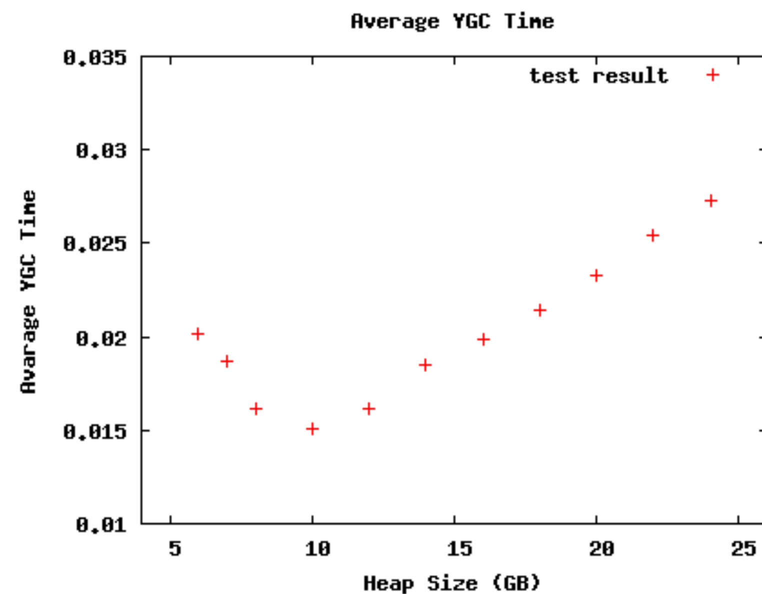
There is *always* at least that much GC overhead een with very large heap size

YP Proprietary Information (Internal Use

More Tests @ 300qps

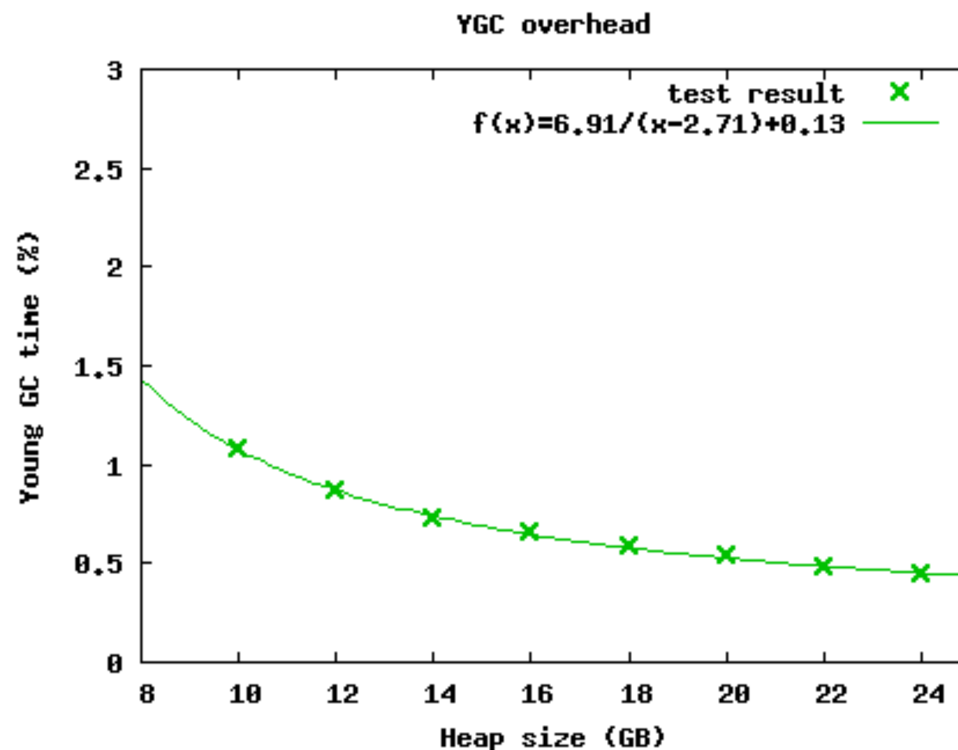
- When heap is barely enough, promotion time increases a lot more than release time decreases





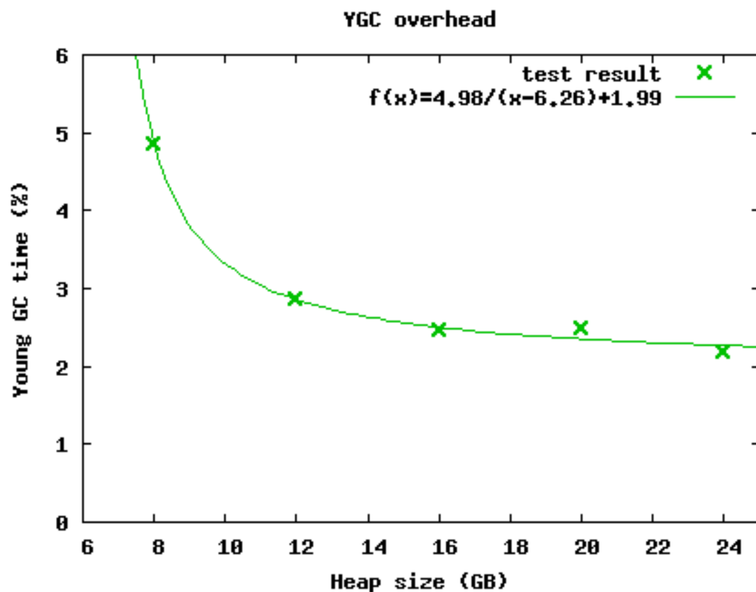
Light Load Test @ 50qps

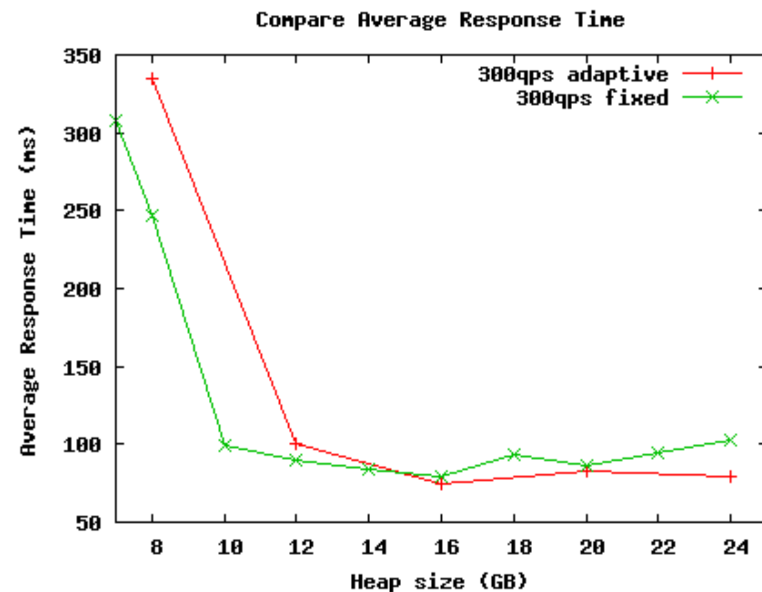
- Under lighter load, both release factor and relocation factor decrease



Adaptive JVM Settings @ 300qps

- MaxGCRation : the ratio of non GC time divided by GC time does not exceed MaxGCRatio
  - Set MaxGCRatio = 99
  - GC ratio is $\frac{1-c}{c}$





Summary of Findings

- Under steady workload, YGC overhead is inversely proportional to the young generation size
- YGC overhead is lower bounded by the value of object creation speed divided by object release speed
- Reducing workload reduces YGC overhead, its lower bound, and its minimum survival space size
- There is performance penalty if heap size is ‘barely enough’
- Runtime option MaxGCRatio is not always achievable

Assumptions and Limits

- Assumptions
  - Constant *Object Release Factor* and *Object Promotion Factor*
  - Generational GC mechanism as specified in JAVA documents
  - Steady workload
- Limits
  - The results come to us as an surprise side effect. We have limited resources to conduct other senarios.