Some important points about GC  
From Evernote:<http://evernote.com/>  
  
Symptoms of Issue with GC  
  
  
 1.  OOME because of fragmented heap available. Unable to allocate large object in spite of total memory being available is more than the allocation request.  
 2.  GC kicking very often  
 3.  Long GCs resulting on longer pause  times.  
 4.  Concurrent mode failure resulting in Full GC happening very often.  
  
Some of the popular tools  
  
  
 1.  HP JMeter  
 2.  garbagecat (google)  
 3.  GCViewer  
 4.  JHicupp  
 5.  Censum  
  
What to look for in GC Log  
  
 1.  For leakages -  
 2.  
    \*   The log file will show many Full GCs,  
    \*   Look at memory freed after Full GC  
    \*   You need a tool to analyse the exact amount of memory leaked.  
 3.  Premature Promotion -  
 4.  
    \*   Default threshold is 4; in theory objects getting promoted with age lesser than 4 is premature promotion  
    \*   In logs -  
    \*  
       \*   Age 1 means objects got promoted from Eden to Tenuered  
       \*   Age 2 means objects got promoted from S0  
       \*   Age 3 means objects got promoted from S1  
       \*   Age 4 means objects got promoted from S0  
 5.  Pause Times -  
 6.  
    \*   High percentage of time spent means heap is undersized which is causing frequent Full GCs  
    \*   Long pause times heap is oversized  
 7.  GC Throughput  
 8.  
    \*   Calculate the total time spent not doing GC (% is calculated as (Total time spent not doing GC / Total JVM Time) \* 100)  
    \*   Timestamps and time spent in GC is available in the log  
 9.  Heap Shape after GC & Before GC  
 10. Allocation Rate - Rate at which objects are created  
 11. Heap recovered after each GC  
  
  
  
 \*   -XX:+UseAdaptiveSizePolicy  
 \*   Pause Goal: -XX:MaxGCPauseMillis  
 \*   Throughput Goal: -XX:GCTimeRatio  
  
  
  
Ideas -  
  
 1.  Simulate with the help of code the consumption of memory (Be careful of not beating in the dark)  
 2.  
    \*   Small object getting allocated in eden and then moved to tenured  
    \*   Large objects directly in tenured  
    \*   The number of GCs in Young  
    \*   The number of GCs in Tenured  
    \*   The time it takes  
 3.  Mandatory parameters to be set : -verbose:gc -Xloggc:log.gc -XX:+PrintGCDetails -XX:+PrintTenuringDistribution  
 4.  You want most objects to die young in young generation.  
 5.  
    \*   So that fewer objects get promoted  
    \*   So that fewer tenured collections happens  
 6.  AdaptiveSizing  
 7.  Goals (Throughput and Pause Times)  
  
  
Understanding the GC Log  
  
  
  
  
  
{Time since VM started, in seconds}: [GC {Time since VM started}: [{GC type}  
Desired survivor size {size of one survivor space} bytes, new threshold {y} (max {x}) <- how many collections objects can stay in the young gen. More on this later  
- age 1: {a} bytes, {a} total <- How many objects have survived one collection. Next collection, any surviving objects will appear in age 2  
- age 2: {b} bytes, {a+b} total  
: {pre-collection young gen[5] usage}K->{post-collection young gen usage}K({Total young gen size}K), {young gen time} secs]  
{pre-collection heap size}K->{post-collection heap size}K({total heap size}K), {total collection time} secs]  
  
Log Rotation  
  
  
  -verbose:gc -Xloggc:gc.log -XX:+PrintGCDetails -XX:+PrintTenuringDistribution -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=30K  
  
  
  
 \*   -XX:+UseGCLogFileRotation  
must be used with -Xloggc:<filename>;  
 \*   -XX:NumberOfGCLogFiles=<number of files>  
must be >=1, default is one;  
 \*   -XX:GCLogFileSize=<number>M (or K)  
  
  
  
  
  
introduced three flags:  
1) -XX:+UseGCLogRotation                       must be used with -Xloggc:<filename>  
2) -XX:NumberOfGClogFiles=<number of files>    must be >=1, default is one  
3) -XX:GCLogFileSize=<number>M (or K)          default will be set to 512K  
  
if UseGCLogRotation set and -Xloggc gives file name, do Log rotation, depends on other flag settings.  
if NumberOfGClogFiles = 1, using same file, rotate log when reaches file capicity (set by GCLogFileSize) in <filename>.0  
if NumberOfGClogFiles > 1, rotate between files <filename>.0, <filename>.1, ..., <filename>.n-1  
  
GC output through outputStream, we have multiple threads, vmThread, CMS threads, GC worker threads. Check if need rotation at every safepoint. When stop world (at safepoint), changing file handle is safe, else need grab lock to do the change. i.e. tty\_lock.