Taking out the Trash!
The G1 Garbage Collector Overview for Everyone

@mattjrobson

Matthew Robson
Senior Technical Account Manager
May 3rd, 2017
Today’s Goals

✔ What’s your role?

✔ How does it work?

✔ Why is it making those decisions?

✔ Which logs are useful?

✔ Addressing the most common problems
G1 is...

A Java Garbage Collector

- Dynamic
- Generational
- Region Based
- Non-Contiguous
- Parallel
- Multi-Phased
- Incrementally Compacting
- Fully Evacuating
- Garbage First
Your Role
G1 Has Goals

How can I help?

- Keep it simple - Predictable Pause Times
  - Soft target defined by `MaxGCPauseMillis`
  - How many regions are collectible within my target

- Consistent Throughput
  - Maintain a predictable number of transactions per second

- Find the Balance - Understand Your Application!
  - **Low Latency / Time Sensitive** = Lower Max Pause Time
    - Absolutely cannot tolerate application disruption
  - **High Throughput / Lots of Data** = Higher Max Pause Time
    - Push as much data as fast as we can; longer pauses are not a problem

Do what’s necessary - In the time defined - Irrespective of the overall Heap Size
G1 Has Goals

How can I hinder?

- Unlike other collectors, G1 set out to simplify parameters and tuning options
  - The more you set, the less G1 is able to do dynamically
- Start out simple; do not carry over settings from other collectors
  - Enable G1
  - Set `Xms=Xmx`
  - Define a pause target
  - Turn on lots of GC logging
  - Test
  - Tune
  - Repeat

There is no definitive guide or magic set of options; you are responsible for **evaluating performance**, making **incremental changes** and **re-evaluating** until you reach your goals
The *How* and The *Why*
(with some sweet *logs*)
Regions

Understand me, before you change me

- 5 Region Types - \( (E) \)den, \( (S) \)urvivor, \( (O) \)ld, \( (H) \)umongous and \( (F) \)ree
- Breaks the heap into ~2048 Regions
- Power of \( 2 \) from 1 to 32MB

<table>
<thead>
<tr>
<th>12 GB Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regions</td>
</tr>
<tr>
<td>12288 / 2048 Regions</td>
</tr>
<tr>
<td>12288 / 8MB Region</td>
</tr>
<tr>
<td>12288 / 4MB Region</td>
</tr>
</tbody>
</table>

- Explicitly set through \texttt{G1HeapRegionSize}
  - Fewer Regions means less flexibility
  - Longer to scan, mark and copy
Why Regions?

And what are they?

A Region represents a block of allocated space that can hold objects of any generation without the need to maintain contiguity with other Regions of the same generation.

- Reduced synchronization
  - Regions are allocated through a Thread Local Allocation Buffer (TLAB)
  - Object allocation can happen within a TLAB without additional synchronization

- Reduced fragmentation
  - Guaranteed evacuation of Young Regions
  - Incremental and Concurrent compaction of Old Regions

- Dynamic
  - Number of Young Regions is proportional to what’s collectable within the pause target
  - Size is adjusted after each collection
Allocation, Evacuation and Promotion

Phase 1 - Young Collection Pause (YC)

- All new objects smaller than 50% of the Region size are allocated in Eden
- Number of Eden Regions defined by what can be collected within the pause target
Allocation, Evacuation and Promotion

Phase 1 - Young Collection Pause (YC)

- Younger objects are compacted into new Survivor Regions
- Tenured objects are promoted to new Old Regions
Young Log

The Most Common Collection

2016-12-1T04:18.811-0500: 29.959: [GC pause (G1 Evacuation Pause) (young), 0.0305171 secs]
[Parallel Time: 26.6 ms, GC Workers: 4]

[GC Worker Start (ms): Min: 29960.0, Avg: 29961.0, Max: 29962.1, Diff: 2.1]
[Ext Root Scanning (ms): Min: 0.8, Avg: 3.5, Max: 9.7, Diff: 8.9, Sum: 13.9]
[Update RS (ms): Min: 0.0, Avg: 0.3, Max: 0.4, Diff: 0.4, Sum: 1.1]
[Processed Buffers: Min: 0, Avg: 66.0, Max: 134, Diff: 134, Sum: 264]
[Scan RS (ms): Min: 0.3, Avg: 0.3, Max: 0.3, Diff: 0.1, Sum: 1.1]
[Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]
[Object Copy (ms): Min: 15.8, Avg: 19.0, Max: 20.4, Diff: 4.7, Sum: 76.1]
[Termination (ms): Min: 0.0, Avg: 1.8, Max: 2.9, Diff: 2.9, Sum: 7.3]
[Termination Attempts: Min: 1, Avg: 1.0, Max: 1, Diff: 0, Sum: 4]
[GC Worker Other (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.1]
[GC Worker Total (ms): Min: 23.7, Avg: 24.9, Max: 26.5, Diff: 2.8, Sum: 99.8]
[GC Worker End (ms): Min: 29985.8, Avg: 29986.0, Max: 29986.5, Diff: 0.7]
[Code Root Fixup: 0.0 ms]
[Code Root Purge: 0.0 ms]
[Clear CT: 0.3 ms]
[Other: 3.7 ms]
[Choose CSet: 0.0 ms]
[Ref Proc: 1.4 ms]
[Ref Enq: 0.0 ms]
[Redirty Cards: 0.0 ms]
[Humongous Register: 0.1 ms]
[Humongous Reclaim: 0.0 ms]
[Free CSet: 0.5 ms]
[Eden: 1097.0M(1097.0M)->0.0B(967.0M) Survivors: 13.0M->139.0M Heap: 1694.4M(2048.0M)->736.3M(2048.0M)]
[Times: user=0.08 sys=0.00, real=0.03 secs]
Occupy

Phase 1 Transition

- Old occupancy will continue to grow as Tenured objects are promoted

- At the end of each Young Collection (YC), non-Young occupancy is evaluated against the InitiatingHeapOccupancyPercent (IHOP) (45% default)

- Known as the ‘soft-margin’, passing the IHOP threshold triggers Concurrent Marking
Young Ergonomics

-XX:+PrintAdaptiveSizePolicy - Why is it doing that?

2016-12-30T13:28:18.343-0500: 130.629: [GC pause (G1 Evacuation Pause) (young)
130.629: [G1Ergonomics (CSet Construction) start choosing CSet, _pending_cards: 1792, predicted base time: 2.98 ms, remaining time: 197.02 ms, target pause time: 200.00 ms]

130.629: [G1Ergonomics (CSet Construction) add young regions to CSet, eden: 664 regions, survivors: 112 regions, predicted young region time: 90.15 ms]

130.629: [G1Ergonomics (CSet Construction) finish choosing CSet, eden: 664 regions, survivors: 112 regions, old: 0 regions, predicted pause time: 93.13 ms, target pause time: 200.00 ms]

130.655: [G1Ergonomics (Concurrent Cycles) request concurrent cycle initiation, reason: occupancy higher than threshold, occupancy: 1013972992 bytes, allocation request: 0 bytes, threshold: 966367620 bytes (45.00 %), source: end of GC], 0.0266860 secs]

227.306: [G1Ergonomics (Concurrent Cycles) request concurrent cycle initiation, reason: occupancy higher than threshold, occupancy: 115343360 bytes, allocation request: 530800 bytes, threshold: 115133625 bytes (45.00 %), source: concurrent humongous allocation]
Initial Mark

Phase 2 - Where do I start?

- Stop The World Pause piggybacked on a Young Collection
- Marks all root objects
- Top At Mark Start (TAMS) is set to the current top of each regions

130.726: [G1Ergonomics (Concurrent Cycles) initiate concurrent cycle, reason: concurrent cycle initiation requested]
Concurrent Marking

Phase 2 - What’s the catch?

- Based on a Snapshot-At-The-Beginning (SATB) principal
  - Only objects which exist at the time of the snapshot may be identified as garbage
  - Newly allocated objects are implicitly marked live (above the Next TAMS)
  - Calculates the necessary live data information to collect “Garbage First”
## Concurrent Marking Log

<table>
<thead>
<tr>
<th>Time</th>
<th>EventDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-12-12T10:40:08.363-0500: 19.510:</td>
<td>GC pause (G1 Evacuation Pause) (young) (initial-mark), 0.0387872 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.405-0500: 19.552:</td>
<td>GC concurrent-root-region-scan-end, 0.0030613 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.405-0500: 19.553:</td>
<td>GC concurrent-mark-start</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.711-0500: 19.858:</td>
<td>GC concurrent-mark-end, 0.3055438 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.713-0500: 19.861:</td>
<td>GC remark</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.713-0500: 19.861:</td>
<td>Finalize Marking, 0.0014099 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.715-0500: 19.862:</td>
<td>GC ref-proc, 0.0000480 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.715-0500: 19.862:</td>
<td>Unloading, 0.0025840 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.724-0500: 19.872:</td>
<td>GC cleanup 1757M-&gt;914M(2048M), 0.0023579 secs</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.727-0500: 19.875:</td>
<td>GC concurrent-cleanup-start</td>
</tr>
<tr>
<td>2016-12-12T10:40:08.729-0500: 19.876:</td>
<td>GC concurrent-cleanup-end, 0.0012954 secs</td>
</tr>
</tbody>
</table>

**Times:**
- user=0.01 sys=0.00, real=0.00 secs
Garbage First

Phase 2 Transition

- During **GC Cleanup** the Candidate Old Region list is finalized
  - A Region is a candidate if live objects are < 85% (**G1MixedGCLiveThresholdPercent**)
  - Regions are sorted based on their GC efficiency

- Once CM finishes, an immediate Young Collection occurs
  - Garbage from Old Regions is > 5% (**G1HeapWastePercent**) - Start Mixed Collections

---

2016-12-30T13:28:18.745-0500: 131.030: [GC pause (G1 Evacuation Pause) (young)
131.051: [G1Ergonomics (Mixed GCs) **start mixed GCs**, reason: candidate old regions available, candidate old regions: 740 regions, **reclaimable**: 485716240 bytes (22.62 %), **threshold**: 5.00 %], 0.0101749 secs]
Mixed Collections

Phase 3 - Mixed Collection Pause (MC)

- Mixed Collections are handled incrementally and executed immediately
  - The candidate Old Regions are divided by `G1MixedGCCCountTarget` (default 8)
  - Goal is to collect at least that many Old Regions per cycle
Mixed Collections - Incremental Compaction

Phase 3 - Mixed Collection Pause (MC)

- Mixed Collections provide incremental compaction
  - Remaining live objects from the collected Old Regions are copied into new ‘highly live’ regions
Mixed Ergonomics

What's up with the Old?

2016-12-30T13:28:18.777-0500: 131.063: [GC pause (G1 Evacuation Pause) (mixed)]

131.063: [G1Ergonomics (CSet Construction) start choosing CSet, _pending_cards: 1061, predicted base time: 2.66 ms, remaining time: 197.34 ms, target pause time: 200.00 ms]
131.063: [G1Ergonomics (CSet Construction) add young regions to CSet, eden: 89 regions, survivors: 13 regions, predicted young region time: 11.28 ms]

131.063: [G1Ergonomics (CSet Construction) finish adding old regions to CSet, reason: old CSet region num reached max, old: 205 regions, max: 205 regions]

131.063: [G1Ergonomics (CSet Construction) finish choosing CSet, eden: 89 regions, survivors: 13 regions, old: 205 regions, predicted pause time: 19.04 ms, target pause time: 200.00 ms]

131.073: [G1Ergonomics (Mixed GCs) continue mixed GCs, reason: candidate old regions available, candidate old regions: 535 regions, reclaimable: 305363768 bytes (14.22 %), threshold: 5.00 %], 0.0141132 secs]
Mixed Collections

Phase 3 Transition

- Collections continue until garbage drops below $G1HeapWastePercent$ or 8 iterations

2016-12-30T13:28:18.877-0500: 131.163: [GC pause (G1 Evacuation Pause) (mixed)]

131.187: [G1Ergonomics (Mixed GCs) do not continue mixed GCs, reason: reclaimable percentage not over threshold, candidate old regions: 254 regions, reclaimable: 107174304 bytes (4.99 %), threshold: 5.00 %], 0.0172178 secs]
Humongous Allocation

My object is so big, I cannot lie, a single young region, I shall not try

- Any object larger than 50% of a single Region
  - Allocated directly to Old and tagged as Humongous Start / Continues
- An object larger than a single Region must be allocated into contiguous free Regions
Full GC

Why oh why, a Full GC, did my collector try?

- Same implementation as the Serial Collector
  - Single Threaded
  - Stop The World
- Collects all Regions
- Fully Compacting
- Guarantees all garbage will be removed
- May shrink (MaxHeapFreeRatio) or expand (MinHeapFreeRatio) the heap if you do not have Xms=Xmx
Full GC Ergonomics

Why is it doing that?

106.445: [G1Ergonomics (Heap Sizing) attempt heap expansion, reason: allocation request failed, allocation request: 24 bytes]

106.445: [G1Ergonomics (Heap Sizing) expand the heap, requested expansion amount: 1048576 bytes, attempted expansion amount: 1048576 bytes]

106.445: [G1Ergonomics (Heap Sizing) did not expand the heap, reason: heap already fully expanded]

2016-12-30T13:27:54.160-0500: 106.445: [Full GC (Allocation Failure)]

106.539: [G1Ergonomics (Heap Sizing) attempt heap shrinking, reason: capacity higher than max desired capacity after Full GC, capacity: 2147483648 bytes, occupancy: 391145472 bytes, max desired capacity: 1303818239 bytes (70.00 %)]

106.570: [G1Ergonomics (Heap Sizing) shrink the heap, requested shrinking amount: 843665409 bytes, aligned shrinking amount: 843055104 bytes, attempted shrinking amount: 843055104 bytes]

2047M->373M(1244M), 0.1278200 secs]
Metaspace

The new Perm

- Metaspace lives in native memory and is committed as necessary (non-contiguous)
  - No max size (by default), bound by OS memory and SWAP
  - Grows dynamically until it reaches max size
  - Faster, because it lives in native memory
  - **MetaspaceSize** (high watermark) determines when a collection will happen
    - Depending on the amount freed, the high watermark may increase

- **UseCompressedClassesPointers** creates a separate 1Gig class space
  - **CompressedClassSpaceSize** is reserved in contiguous space at VM initialization
    - This cannot change or grow
  - Committed space counts as part of **MaxMetaspace**
The most common problems
7 Common G1 Issues

And where to start

✓ Collect and analyze the GC logs
  ○ Garbagecat and GCViewer are good options
✓ Calculate the size of your Live Data Set
  ○ At any given time, how much is alive?
✓ Calculate your most common large object sizes
  ○ Does the default `G1HeapRegionSize` align?
✓ Evaluate your promotion rate
  ○ What is dieing young versus what ends up in Old
✓ Map Growth of Young and Old Generations over time
  ○ Is the Eden too compressed?
7 Common G1 Issues

And where to start

1. Promotion Failures / Premature Marking - (to-space exhausted), 0.5669726 secs]
   - Very Long Pause compared to a regular Young Collection
   - Copied objects must be updated
   - Objects which failed to copy are tenured in place (as there are no free Regions)
   - Evaluate Concurrent Marking (InitiatingHeapOccupancyPercent)
   - Mixed Collection Effectiveness
   - Tune Heap Size and Reserve Percentage
Common G1 Issues Cont.

Big Issues from Big Objects

2. Humongous Obj - reason: requested by GC cause GC cause: G1 Humongous Allocation
   - Creates fragmentation
   - Accelerates Old region growth and premature marking
   ✓ Compare and adjust G1HeapRegionSize in relation to the average object size
   ✓ Tune Max Heap to better accommodate common object size

Region Size: 4096 K
Object A: 12800 K
Result: 4 regions and 16384 K
Waste: 3584 K
Common G1 Issues Cont.

Full GC Fail

3. Full GC - 3 Most Common Cases:
   a. Full GC (Metadata GC Threshold)
      ☒ Setting a **MaxMetaspaceSize** that is too small for the workload
      ☒ **UseCompressedClassesPointers** creating tight Metaspace
      ☒ Classloader leaks
      ✓ Tune **Metaspace** for proper sizing and check for leaks

   b. [GC pause (young) (to-space exhausted) and Full GC]
      ☒ Heap can no longer be expanded and there are no free regions for evacuation
      ☒ The **G1ReservePercent** did not provide enough of a promotion buffer
      ☒ Collector could not recover
      ✓ Evaluate Concurrent Marking (**IHOP**) and Mixed Collection **effectiveness**
      ✓ Tune Max Heap Size and Reserve Percentage
Did you actually mark anything?

c. [GC concurrent-mark-start] and [Full GC] and [GC concurrent-mark-abort]
   - Running out of heap before Concurrent Marking can finish
   - Longer lived objects with a promotion rate faster than you can collect
   ✓ Evaluate when Concurrent Marking starts (InitiatingHeapOccupancyPercent)
   ✓ Review how long Concurrent Marking takes
   ✓ Tune Max Heap Size based on your Live Data Set

4. Concurrent Marking - [GC concurrent-mark-end, 25.3988906 secs]
   - Running out of heap before concurrent marking can finish
   - Not collecting a high percentage of garbage
   ✓ Large heap and undersized machine - Not enough CPU
   ✓ Too few concurrent threads - Percentage of Parallel Threads
     ✓ Increasing ConcGCThreads will take away CPU from application threads
   ✓ Object creation rate leading to many interrupting Young Collections
Common G1 Issues Cont.

Why so slow?

5. Long / Inefficient Mixed Collections
   - Leads to Full GC
   - Takes away from Application processing time
   - Collecting too many inefficient regions? Increase `G1HeapWastePercent`
   - Not maximizing the full pause time? Increase `G1OldCSetRegionThresholdPercent`

6. Long Update RS
   - Tune concurrent refinement threads - `G1ConcRefinementThreads`
   - Tune RSet Update time - `G1RSetUpdatingPauseTimePercent`
   - Check for working being pushed to mutator threads

7. Long Scan RS
   - Evaluate the RSet statistics - `G1SummarizeRSetStats`
   - Check for coarsenings in `RSetStats`
Useful *Flags*
## G1 Flags

**Keep it simple and test**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:+UseG1GC</td>
<td>Enable G1</td>
</tr>
<tr>
<td>-XX:MaxGCPauseMillis=200</td>
<td>G1 soft pause target (ms)</td>
</tr>
<tr>
<td>-XX:InitiatingHeapOccupancyPercent=45</td>
<td>Soft margin to initiate marking</td>
</tr>
<tr>
<td>-XX:G1HeapRegionSize=1m</td>
<td>Region size, as a power of 2</td>
</tr>
<tr>
<td>-XX:G1MixedGCCountTarget=8</td>
<td>Target number of mixed collections</td>
</tr>
<tr>
<td>-XX:G1MixedGCLiveThresholdPercent=85</td>
<td>Live byte threshold for Old region CSet inclusion</td>
</tr>
<tr>
<td>-XX:G1HeapWastePercent=5</td>
<td>Amount of heap to waste to avoid expensive regions</td>
</tr>
<tr>
<td>-XX:G1ReservePercent=10</td>
<td>Space reserved for promotion</td>
</tr>
<tr>
<td>-XX:G1EagerReclaimHumongousObjects=true</td>
<td>Reclaim Humongous objects with Young GC</td>
</tr>
</tbody>
</table>
### G1 Flags Cont.

Keep it simple and test

<table>
<thead>
<tr>
<th>Flag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:G1ConcRefinementThreads</td>
<td>Parallel threads for RSet updates</td>
</tr>
<tr>
<td>-XX:G1NewSizePercent=5</td>
<td>Set the minimum Young size</td>
</tr>
<tr>
<td>-XX:G1MaxNewSizePercent=60</td>
<td>Set the maximum Young size</td>
</tr>
<tr>
<td>-XX:G1OldCSetRegionThresholdPercent=10</td>
<td>Max Old regions in CSet as a percent of heap</td>
</tr>
<tr>
<td>-XX:G1RSetUpdatingPauseTimePercent=10</td>
<td>Percent of time for Update RS</td>
</tr>
<tr>
<td>-XX:SurvivorRatio=8</td>
<td>Ratio of Eden to Survivor space</td>
</tr>
<tr>
<td>-XX:MaxTenuringThreshold=15</td>
<td>Number of iterations before promotion to Old</td>
</tr>
<tr>
<td>-XX:ParallelGCThreads='logical CPUs'</td>
<td>Parallel STW threads</td>
</tr>
<tr>
<td>-XX:ConcGCThreads='25% of Parallel'</td>
<td>Concurrent marking threads</td>
</tr>
</tbody>
</table>
## G1 Flags Cont.

*Keep it simple and test*

<table>
<thead>
<tr>
<th>Flag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:MetaspaceSize=</td>
<td>Initial Metaspace high water mark</td>
</tr>
<tr>
<td>-XX:MaxMetaspaceSize=unlimited</td>
<td>Max Metaspace size</td>
</tr>
<tr>
<td>-XX:CompressedClassSpaceSize=1G</td>
<td>Maximum class area for Compressed Class Pointers</td>
</tr>
<tr>
<td>-XX:+UseCompressedOops</td>
<td>Use 32-bit references</td>
</tr>
<tr>
<td>-XX:+UseCompressedClassPointers</td>
<td>Use 32-bit class pointers</td>
</tr>
</tbody>
</table>
# Logging Flags

## Must Use

<table>
<thead>
<tr>
<th>Flag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Xloggc:/path/to/gc.log</code></td>
<td>Path where the GC logs are written</td>
</tr>
<tr>
<td><code>-XX:+UseGCLogFileRotation</code></td>
<td>Enable GC log file rotation</td>
</tr>
<tr>
<td><code>-XX:NumberOfGCLogFiles=&lt;value&gt;</code></td>
<td>Number of rotated GC logs files to retain</td>
</tr>
<tr>
<td><code>-XX:GCLogFileSize=&lt;size&gt;</code></td>
<td>Size of each GC logs file to initiate rotation</td>
</tr>
<tr>
<td><code>-XX:+PrintGCDetails</code></td>
<td>Detailed GC log</td>
</tr>
<tr>
<td><code>-XX:+PrintGCDateStamps</code></td>
<td>Actual date and timestamp of the collection</td>
</tr>
<tr>
<td><code>-XX:+PrintGCAplicationStoppedTime</code></td>
<td>Amount of time the application stopped during GC</td>
</tr>
<tr>
<td><code>-XX:+PrintGCAplicationConcurrentTime</code></td>
<td>Amount of time the application ran between GCs</td>
</tr>
<tr>
<td><code>-XX:-PrintCommandLineFlags</code></td>
<td>Prints all the command line flags in the GC log</td>
</tr>
</tbody>
</table>
## Logging Flags
### For Testing and Analysis

<table>
<thead>
<tr>
<th>Flag</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:+PrintAdaptiveSizePolicy</td>
<td>Details about the collector ergonomics</td>
</tr>
<tr>
<td>-XX:+PrintTenuringDistribution</td>
<td>Survivor space usage and distribution</td>
</tr>
<tr>
<td>-XX:+PrintReferenceGC</td>
<td>Time spent processing references</td>
</tr>
</tbody>
</table>
## Logging Flags

### For Debug

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-XX:+UnlockDiagnosticVMOptions</code></td>
<td></td>
</tr>
<tr>
<td><code>-XX:+G1SummarizeConcMark</code></td>
<td>Summarizes Concurrent Mark at JVM exit</td>
</tr>
<tr>
<td><code>-XX:+G1PrintHeapRegions</code></td>
<td>Print the heap regions selected for allocation, cleanup, reuse, compact, cset, commit, failure etc...</td>
</tr>
<tr>
<td><code>-XX:+G1PrintRegionLivenessInfo</code></td>
<td>Prints previous and next liveness data per Old region before and after every concurrent mark cycle</td>
</tr>
<tr>
<td><code>-XX:+G1SummarizeRSetStats</code></td>
<td>Print RSet processing information every X, where X is measured in GC cycles</td>
</tr>
<tr>
<td><code>-XX:G1SummarizeRSetStatsPeriod=1</code></td>
<td></td>
</tr>
<tr>
<td><code>-XX:+UnlockExperimentalVMOptions</code></td>
<td></td>
</tr>
<tr>
<td><code>-XX:G1LogLevel=fine, finer, finest</code></td>
<td>Increased logging verbosity on collections</td>
</tr>
<tr>
<td><code>-XX:+G1TraceEagerReclaimHumongousObjects</code></td>
<td>Details about live and dead Humongous objects</td>
</tr>
</tbody>
</table>
Supplemental Resources

TAM Blogging

- Part 1: Detailed G1 Introduction
  - https://www.redhat.com/en/about/blog/part-1-introduction-g1-garbage-collector
- Part 2: Collecting and Reading G1 Garbage Collector Logs
  - Publish Date May 9th
- Part 3: Evaluating and Tuning the G1 Garbage Collector
  - Future
- Part 4: A Look Ahead; G1 Changes in JDK9
  - Future
- TAM Blogging Series
  - https://www.redhat.com/en/about/blog/technical-account-managers
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.