03: Java GC tuning for low latency applications

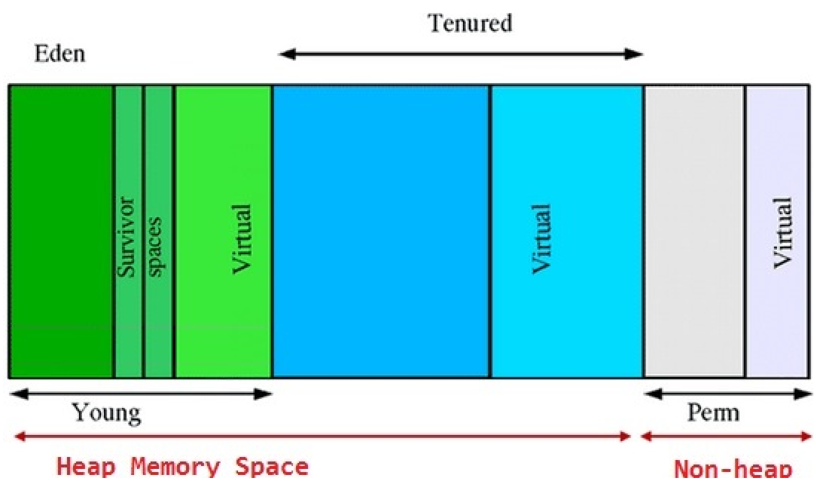
Posted on February 8, 2016 by Arulkumaran Kumaraswamipillai

This assumes that you have read the basics on Java GC at Java Garbage Collection interview Q&A to ascertain your depth of Java knowledge. This is a must know topic for those who like to work on low latency applications.

Q1. In what ways does GC impact latency and throughput of your application?
A1.

1. CPU & **memory** overheads due to parallel garbage collection (GC) algorithms like Concurrent Mark & Sweep (CMS), G1, etc and other reasons listed below.

2. Longer GC **pauses** due to **a)** heap memory fragmentation requiring old space to be compacted, **b)** when more young objects in the "Eden" space need to be copied over to the "Survivor" spaces, and then to the old (i.e. tenured) space.



Java heap sections

3. Frequent minor GC cycles. The Survivor space GC calls are less frequent than in the Eden space.

4. Frequent **major GC** cycles where the long lived objects in the tenured space are collected and possibly compacted.

Q2. In your experience, what are the common practical reasons?

A2. Common reason is incorrect GC configuration or leaving it at a default configuration without tuning it for a particular application.

1) Young generation being too small. Generally 40% is to 60% is required between young and old spaces respectively.

2) Heap size is too small (i.e -Xmx). The application footprint is larger than the allocated heap size.

3) Use of wrong GC algorithm. Algorithm that causes the application threads to pause to collect garbage (aka Stop-the-world collectors) or **Non-compacting** algorithms that cause heap fragmentation. Some GC algorithms have a higher footprint than the others.

5) Incorrectly creating and discarding objects without astutely reusing them with a flyweight design pattern or proper caching strategy.

6) Other OS activities like swap space or networking activities during GC can make GC pauses last longer.

7) Wrong use of libraries taking up lots of the heap space. For example, XML based report generation using DOM parser as opposed to StAX for large reports generated concurrently by multiple users. DOM is very memory hungry.

Q3. How will you go about tuning GC?
A3. You need to have a good understanding of **how to log GC?** and also know **what JVM parameters to use?** to tune the behavior.

Logging GC

1) Add the verbose flags:

```

-verbose:gc (print the GC logs)
-Xloggc: (comprehensive GC logging)
-XX:+PrintGCDetails (for more detailed output)
-XX:+PrintTenuringDistribution (tenuring threshol

```

2) Apply the **GC visualization techniques** to turn **gc.log** into **GC event charts** (plotted on x/y time series) and **heap size charts**.

Tools to visualize GC log files:

1. GCViewer (open-source)

2. IBM's GC toolkit.

3. Netflix **gcviz** in GitHub.

Senior Java developers must have a good handle on

open all | close all
- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience I
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction Managen

80+ step by step Java Tutorials

open all | close all
- Setting up Tutorial (6)
- Tutorial - Diagnosis (2
- Akka Tutorial (9)
- Core Java Tutorials (2
- Hadoop & Spark Tuto
- JEE Tutorials (19)
- Scala Tutorials (1)
- Spring & HIbernate Tu
- Tools Tutorials (19)
- Other Tutorials (45)

4. HPjmeter.

5) Another useful JVM parameter is to dump the heap on OutOfMemory error.

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=
```

4) If you want to look at what is going on in the VM memory whilst the application is running, **VisualVM** is a good tool that gets shipped with the JDK.

Q4. What tips do you give to reduce GC frequency & length?
A4. There are 2 GC events that take place:

1) Minor: Eden and survivor spaces are collected.

2) Major: In addition to "Eden and survivor" spaces, the tenured space where the long lived objects reside gets cleaned and (possibly) compacted. During a major GC there are 2 steps:

Step 1: Survivors from **young space** are copied to the **old space**.

Step 2: Clean the unused references from the old generation and **compact the old space**. This requires precious CPU cycles.

So, to reduce GC frequency & length, you need to focus on limiting the **number of objects getting tenured**. This will reduce the frequency & length of the major GC.

Increasing the young generation size may result in longer **young GC pauses** if more data survives the eden space and gets copied to the survivor spaces. So, it is a balancing act and don't blindly increase the young generation to minimize major GC.

Preparing for Java written & coding tests

open all | close all
♦ Complete the given
Can you write code? (
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your…to go places?

open all | close all
Career Making Know-
Job Hunting & Resum

The heap fragmentation and associated major (aka full) GC pauses can be minimized by controlling the object promotion rate from young to old space and by reducing the -XX:CMSInitiatingOccupancyFraction=60, which means run GC after the occupancy of the old generation reaches 60%.

Another option to minimize **heap fragmentation** is to deploy the application to multiple JVMs instead of a single JVM, but this will increase maintenance and deployment tasks.

So, the real answer to reduce GC pauses is to tune the parameters to determine the perfect configuration for your application.

Q5. How do you specify the JVM of the different heap sizes?
A5. Using the following JVM arguments:

```

-Xmx2g -Xms2g -XX:NewSize=757MB -XX:MaxNewSize=75

```

-Xmx is the total heap, and is 2GB.

-XX:NewSize to **-XX:MaxNewSize** is the range of the young (aka new) generation's minimum and maximum size.

Old space: is the difference between total heap and new generation.

-XX:+UseTLAB is the **T**hread **L**ocal **A**llocation **B**uffer

-XX:PermSize to **-XX:MaxPermSize**: is the non-heap space.

A rule of thumb is to maintain 40% vs 60% ratio between Young Generation and Old Generation

Q6. What parameters do you use to tune?
A6.

```
-XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEna
-XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiat
-XX:+CMSPermGenSweepingEnabled -XX:+CMSClassUnloa

```

+UseConcMarkSweepGC: Old generation Garbage collection algorithm that attempts to do most of the garbage collection work in the background without stopping application threads while it works. All of the garbage collection algorithms except ConcurrentMarkSweep are stop-the-world, i.e. they stop all application threads while they operate – the stop is known as ‘pause’ time.

-XX:+CMSParallelRemarkEnabled : Reduces the remark pauses in the mark and sweep process.

+UseParNewGC: Young generation parallel garbage collection. Uses multiple threads in parallel.

-XX:SurvivorRatio=2: Segregate the memory between Eden and 2 Survivor spaces. As shown above in the diagram, the young generation contains three sub areas **1)** Eden Space **2)** Survivor One **3)** Survivor Two. This ratio is calculated based on “(young size/survivor size) – 2”. If you decrease the **SurvivorRatio** value then you will get more Survivor Space, and less Survivor Space if you decrease the SurvivorRatio.

Note: SurvivorRatio=6 then the ratio of SurvivorSpace : Eden = 1 : 6 . So two Survivor will take 2 : 6, which means if you have 756 MB of young space, then Eden is (6/8 * 756) and survivor is (1/8 * 756).

-XX:+UseCMSInitiatingOccupancyOnly : Prevents the usage of default major GC trigger based on heuristics. Turning this off will use the next parameter to trigger based on old generation occupancy percentage.

CMSInitiatingOccupancyFraction=40: Run GC after the occupancy of the old generation reaches 40%.

-XX:+CMSPermGenSweepingEnabled and **-XX:+CMSClassUnloadingEnabled**: are used to mitigate against perm gen space OutOfMemory errors.

What is the "Garbage First" (G1) collector?

"Garbage First" (G1) collector was introduced in Java 7. "G1" GC is an incremental parallel compacting GC that provides more predictable pause times compared to CMS GC and Parallel GC. "G1" GC works with much larger heap sizes, and it requires you to set your min/max heap sizes via **-Xms** and **-Xmx** along with a realistic max pause time using **-XX:MaxGCPauseMillis** for it to do its job.

String deduplication feature was added in Java 8 update 20, and it is a part of "G1" garbage collector, so it should be turned on with G1 collector: **-XX:+UseG1GC -XX:+UseStringDeduplication**

Popular Posts

- ♦ 11 Spring boot interview questions & answers
  861 views
- ♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers
  829 views
- 18 Java scenarios based interview Questions and Answers
  448 views
- 001A: ♦ 7+ Java integration styles & patterns interview questions & answers
  407 views
- ♦ 7 Java debugging interview questions & answers
  311 views
- ♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers
  303 views
- 01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers
  294 views
- 01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

288 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ 3. Apache Pig: XPath for XML

♥♦ HashMap & HashSet and how do they internally work? What is a hashing function? ›

Posted in GC, Low Latency, member-paid

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from "Blah blah" to "Wow"? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer