OOM error caused by large array allocation in G1

I recently encountered an OOM error in a Spark application using G1 collector. This application launches multiple JVM instances to process the large data. Each JVM has 6.5GB heap size and uses G1 collector. A JVM instance throws an OOM error during allocating a large (570MB) array. However, this JVM has about 3GB free heap space at that time. After analyzing the application logic, heap usage, and GC log, I guess the root cause may be the lack of consecutive space for holding this large array in G1. I want to know whether my guess is right and why G1 has this defect.

In the following sections, I will detail the JVM info, application, OOM phase, and heap usage. Any suggestions will be appreciated.

[JVM info]

java version "1.8.0_121"

Oracle Java(TM) SE Runtime Environment (build 1.8.0_121-b13)

Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

[Application]

This is a SVM machine learning Spark application available at https://github.com/JerryLead/SparkGC/blob/master/src/main/scala/applications/ml/SVMWithSGDExample.scala.

[OOM phase]

This application generates large task result (about 400MB *nio.ByteBuffe*r) and uses *JavaSerializer* to serialize this task result. During serlializing, the JavaSerializer needs to allocate (expand) a large *SerializeBuffer* (570MB) to hold the task result.

```
318.800: [G1Ergonomics (Heap Sizing) attempt heap expansion, reason:
allocation request failed, allocation request: 573046800 bytes]  318.800:
[G1Ergonomics (Heap Sizing) expand the heap, requested expansion amount:
573046800 bytes, attempted expansion amount: 573571072 bytes]
2017-11-17T09:58:17.362+0800: 318.802: [Full GC (Allocation Failure)  3895M-
>3855M(6656M), 1.7516132 secs]
```

However, though this JVM has about 3GB free space, it throws an OOM error during the expansion as follows.

```
17/11/17 09:59:07 ERROR Executor: Exception in task 1.0 in stage 10.0 (TID
1048) java.lang.OutOfMemoryError: Java heap space  at
java.util.Arrays.copyOf(Arrays.java:3236)  at
java.io.ByteArrayOutputStream.grow(ByteArrayOutputStream.java:118)  at
java.io.ByteArrayOutputStream.ensureCapacity(ByteArrayOutputStream.java:93)
        at
java.io.ByteArrayOutputStream.write(ByteArrayOutputStream.java:153)
        at
org.apache.spark.util.ByteBufferOutputStream.write(ByteBufferOutputStream.sc
ala:41)  at
```

```
java.io.ObjectOutputStream$BlockDataOutputStream.write(ObjectOutputStream.ja
va:1853)         at
java.io.ObjectOutputStream.write(ObjectOutputStream.java:709)       at
org.apache.spark.util.Utils$.writeByteBuffer(Utils.scala:238)       at
org.apache.spark.scheduler.DirectTaskResult$$anonfun$writeExternal$1.apply$m
cV$sp(TaskResult.scala:66)        at
org.apache.spark.scheduler.DirectTaskResult$$anonfun$writeExternal$1.apply(T
askResult.scala:62)       at
org.apache.spark.scheduler.DirectTaskResult$$anonfun$writeExternal$1.apply(T
askResult.scala:62)       at
org.apache.spark.util.Utils$.tryOrIOException(Utils.scala:1269)     at
org.apache.spark.scheduler.DirectTaskResult.writeExternal(TaskResult.scala:6
2)       at
java.io.ObjectOutputStream.writeExternalData(ObjectOutputStream.java:1459)
        at
java.io.ObjectOutputStream.writeOrdinaryObject(ObjectOutputStream.java:1430)
        at
java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1178)
        at
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:348)
        at
org.apache.spark.serializer.JavaSerializationStream.writeObject(JavaSerializ
er.scala:43)     at
org.apache.spark.serializer.JavaSerializerInstance.serialize(JavaSerializer.
scala:100)       at
org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:384)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:11
42)      at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:6
17)      at java.lang.Thread.run(Thread.java:745) 17/11/17 09:59:07 DEBUG
BlockManagerSlaveEndpoint: removing broadcast 14
```

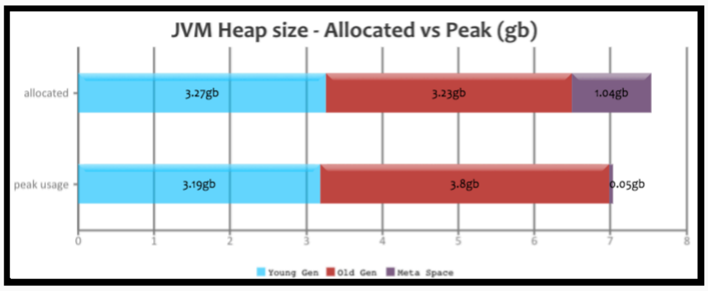
[G1 configuration]

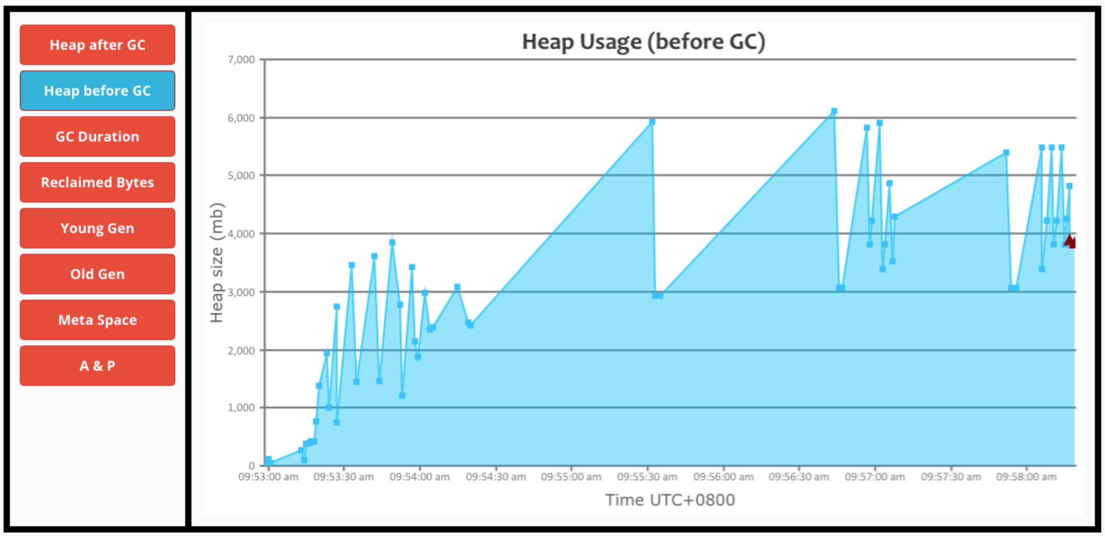
-Xmx6.5G -XX:+UseG1GC -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintGCCause -XX:+PrintGCApplicationStoppedTime -XX:+PrintAdaptiveSizePolicy -XX:+PrintSafepointStatistics -XX:PrintSafepointStatisticsCount=1 -XX:+HeapDumpOnOutOfMemoryError

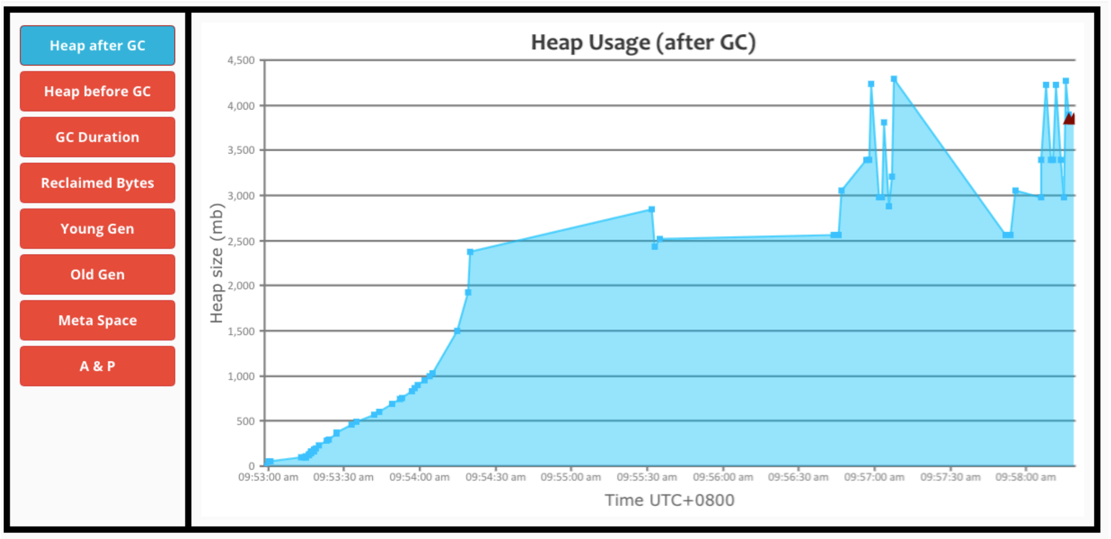
[Heap size and usage]

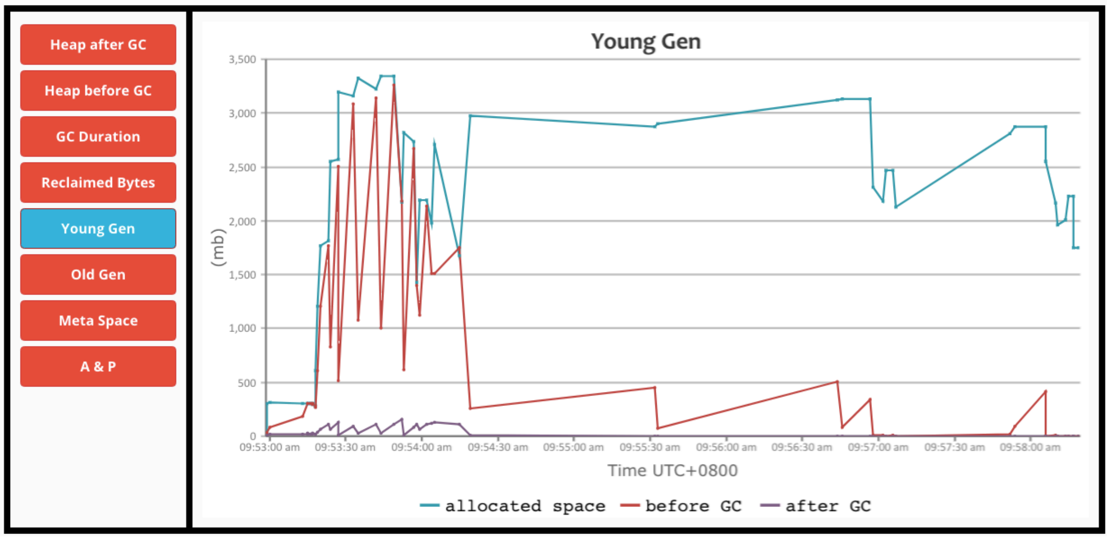
☐ JVM Heap Size

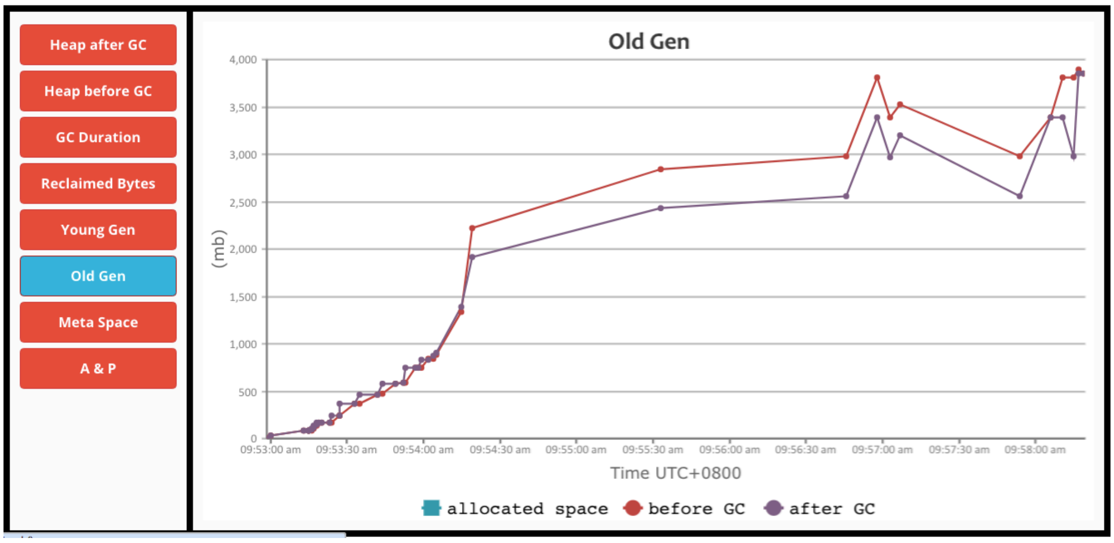
Generation	Allocated ☐	Peak ☐
Young Generation	3.27 gb	3.19 gb
Old Generation	3.23 gb	3.8 gb
Meta Space	1.04 gb	46.36 mb
Young + Old + Meta space	7.54 gb	6.02 gb




Heap after GC
Heap before GC
GC Duration
Reclaimed Bytes
Young Gen
Old Gen
Meta Space
A & P
Heap Usage (before GC)
Heap size (mb)
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
09:53:00 am
09:53:30 am
09:54:00 am
09:54:30 am
09:55:00 am
09:55:30 am
09:56:00 am
09:56:30 am
09:57:00 am
09:57:30 am
09:58:00 am
Time UTC+0800


Heap after GC
Heap before GC
GC Duration
Reclaimed Bytes
Young Gen
Old Gen
Meta Space
A & P
Heap Usage (after GC)
Heap size (mb)
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
09:53:00 am
09:53:30 am
09:54:00 am
09:54:30 am
09:55:00 am
09:55:30 am
09:56:00 am
09:56:30 am
09:57:00 am
09:57:30 am
09:58:00 am
Time UTC+0800


Heap after GC
Heap before GC
GC Duration
Reclaimed Bytes
Young Gen
Old Gen
Meta Space
A & P
Young Gen
(mb)
0
500
1,000
1,500
2,000
2,500
3,000
3,500
09:53:00 am
09:53:30 am
09:54:00 am
09:54:30 am
09:55:00 am
09:55:30 am
09:56:00 am
09:56:30 am
09:57:00 am
09:57:30 am
09:58:00 am
Time UTC+0800
allocated space
before GC
after GC



[GC log]

https://github.com/JerryLead/Misc/blob/master/OOM-SVM-G1-E1/GClog