OOM caused by the memory contention and memory leak in TaskMemoryManager

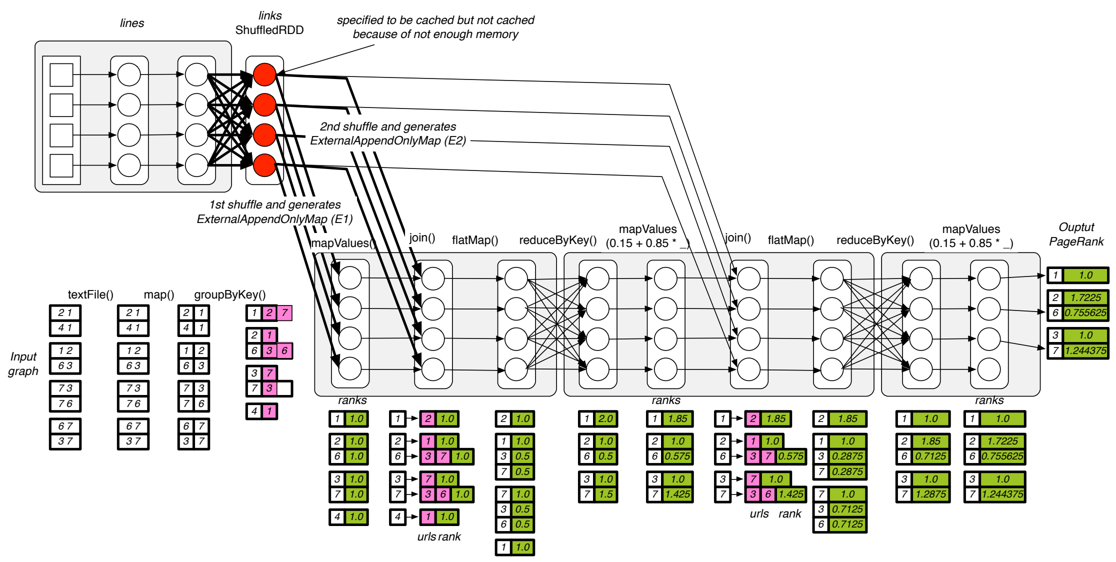
Lijie Xu

[Abstract] I recently encountered an OOM error in a PageRank application (*org.apache.spark.examples.SparkPageRank*). After profiling the application, I found the OOM error is related to the memory contention in shuffle spill phase. Here, the memory contention means that a task tries to release some old memory consumers from memory for keeping the new memory consumers. After analyzing the OOM heap dump, I found the root cause is a memory leak in *TaskMemoryManager*. Since memory contention is common in shuffle phase, this is a critical bug/defect. In the following sections, I will use the application dataflow, execution log, heap dump, and source code to identify the root cause.

[Application]

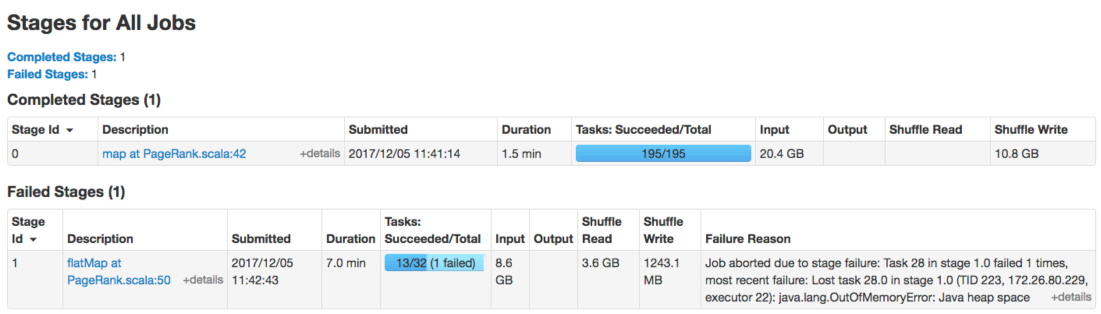
This is a PageRank application from Spark's example library. The following figure shows the application dataflow. The source code is available at [1].

PageRank Dataflow


lines
links
ShuffledRDD
specified to be cached but not cached because of not enough memory
2nd shuffle and generates ExternalAppendOnlyMap (E2)
1st shuffle and generates ExternalAppendOnlyMap (E1)
mapValues()
join()
flatMap()
reduceByKey()
mapValues (0.15 + 0.85 * _)
join()
flatMap()
reduceByKey()
mapValues (0.15 + 0.85 * _)
Ouptut PageRank
textFile()
map()
groupByKey()
Input graph
ranks
urls rank
urls rank

[Failure symptoms]

This application has a map stage and many iterative reduce stages. An OOM error occurs in a reduce task (Task-28) as follows.



Stages for All Jobs

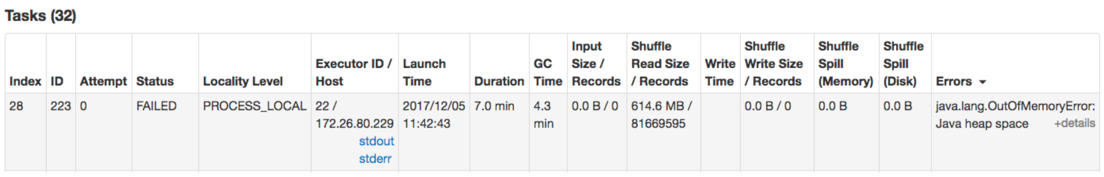
Completed Stages: 1
Failed Stages: 1

Completed Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	map at PageRank.scala:42 +details	2017/12/05 11:41:14	1.5 min	195/195	20.4 GB			10.8 GB

Failed Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
1	flatMap at PageRank.scala:50 +details	2017/12/05 11:42:43	7.0 min	13/32 (1 failed)	8.6 GB		3.6 GB	1243.1 MB	Job aborted due to stage failure: Task 28 in stage 1.0 failed 1 times, most recent failure: Lost task 28.0 in stage 1.0 (TID 223, 172.26.80.229, executor 22): java.lang.OutOfMemoryError: Java heap space +details



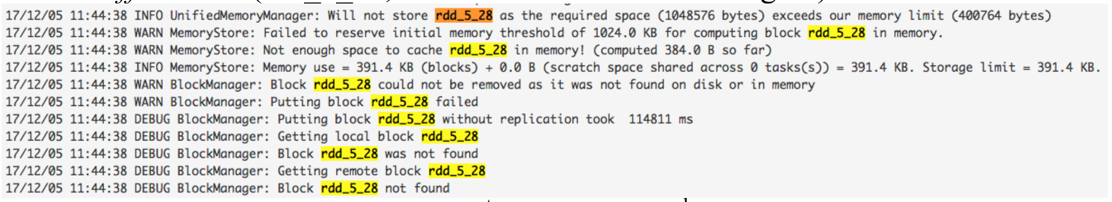
Tasks (32)

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)	Errors ▾
28	223	0	FAILED	PROCESS_LOCAL	22 / 172.26.80.229 stdout stderr	2017/12/05 11:42:43	7.0 min	4.3 min	0.0 B / 0	614.6 MB / 81669595		0.0 B / 0	0.0 B	0.0 B	java.lang.OutOfMemoryError: Java heap space +details

[OOM root cause identification]

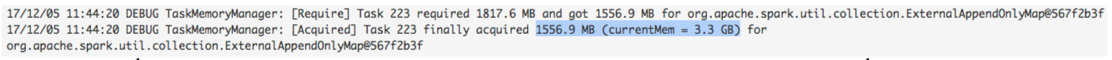
Each executor has 1 CPU core and 6.5GB memory, so it only runs one task at a time. After analyzing the application dataflow, error log, heap dump, and source code, I found the following steps lead to the OOM error.

=> The MemoryManager found that there is not enough memory to cache the *links:ShuffledRDD* (rdd_5_28, red circles in the dataflow figure).



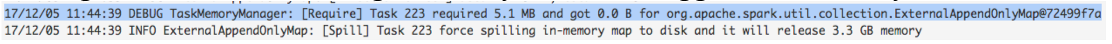
```
17/12/05 11:44:38 INFO UnifiedMemoryManager: Will not store rdd_5_28 as the required space (1048576 bytes) exceeds our memory limit (400764 bytes)
17/12/05 11:44:38 WARN MemoryStore: Failed to reserve initial memory threshold of 1024.0 KB for computing block rdd_5_28 in memory.
17/12/05 11:44:38 WARN MemoryStore: Not enough space to cache rdd_5_28 in memory! (computed 384.0 B so far)
17/12/05 11:44:38 INFO MemoryStore: Memory use = 391.4 KB (blocks) + 0.0 B (scratch space shared across 0 tasks(s)) = 391.4 KB. Storage limit = 391.4 KB.
17/12/05 11:44:38 WARN BlockManager: Block rdd_5_28 could not be removed as it was not found on disk or in memory
17/12/05 11:44:38 WARN BlockManager: Putting block rdd_5_28 failed
17/12/05 11:44:38 DEBUG BlockManager: Putting block rdd_5_28 without replication took  114811 ms
17/12/05 11:44:38 DEBUG BlockManager: Getting local block rdd_5_28
17/12/05 11:44:38 DEBUG BlockManager: Block rdd_5_28 was not found
17/12/05 11:44:38 DEBUG BlockManager: Getting remote block rdd_5_28
17/12/05 11:44:38 DEBUG BlockManager: Block rdd_5_28 not found
```

=> The task needs to shuffle twice (1st shuffle and 2nd shuffle in the dataflow figure).
=> The task needs to generate two *ExternalAppendOnlyMap* (E1 for 1st shuffle and E2 for 2nd shuffle) in sequence.
=> The 1st shuffle begins and ends. E1 aggregates all the shuffled data of 1st shuffle and achieves 3.3 GB.



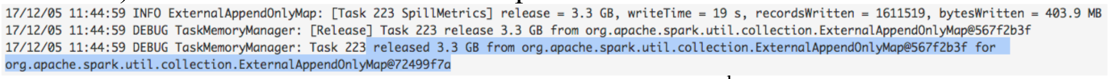
```
17/12/05 11:44:20 DEBUG TaskMemoryManager: [Require] Task 223 required 1817.6 MB and got 1556.9 MB for org.apache.spark.util.collection.ExternalAppendOnlyMap@567f2b3f
17/12/05 11:44:20 DEBUG TaskMemoryManager: [Acquired] Task 223 finally acquired 1556.9 MB (currentMem = 3.3 GB) for
org.apache.spark.util.collection.ExternalAppendOnlyMap@567f2b3f
```

=> The 2nd shuffle begins. E2 is aggregating the shuffled data of 2nd shuffle, and finding that there is not enough memory left. This triggers the memory contention.



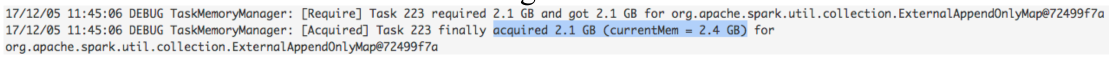
```
17/12/05 11:44:39 DEBUG TaskMemoryManager: [Require] Task 223 required 5.1 MB and got 0.0 B for org.apache.spark.util.collection.ExternalAppendOnlyMap@72499f7a
17/12/05 11:44:39 INFO ExternalAppendOnlyMap: [Spill] Task 223 force spilling in-memory map to disk and it will release 3.3 GB memory
```

=> To handle the memory contention, the TaskMemoryManager releases E1 (spills it onto disk) and assumes that the 3.3GB space is free now.



```
17/12/05 11:44:59 INFO ExternalAppendOnlyMap: [Task 223 SpillMetrics] release = 3.3 GB, writeTime = 19 s, recordsWritten = 1611519, bytesWritten = 403.9 MB
17/12/05 11:44:59 DEBUG TaskMemoryManager: [Release] Task 223 release 3.3 GB from org.apache.spark.util.collection.ExternalAppendOnlyMap@567f2b3f
17/12/05 11:44:59 DEBUG TaskMemoryManager: Task 223 released 3.3 GB from org.apache.spark.util.collection.ExternalAppendOnlyMap@567f2b3f for
org.apache.spark.util.collection.ExternalAppendOnlyMap@72499f7a
```

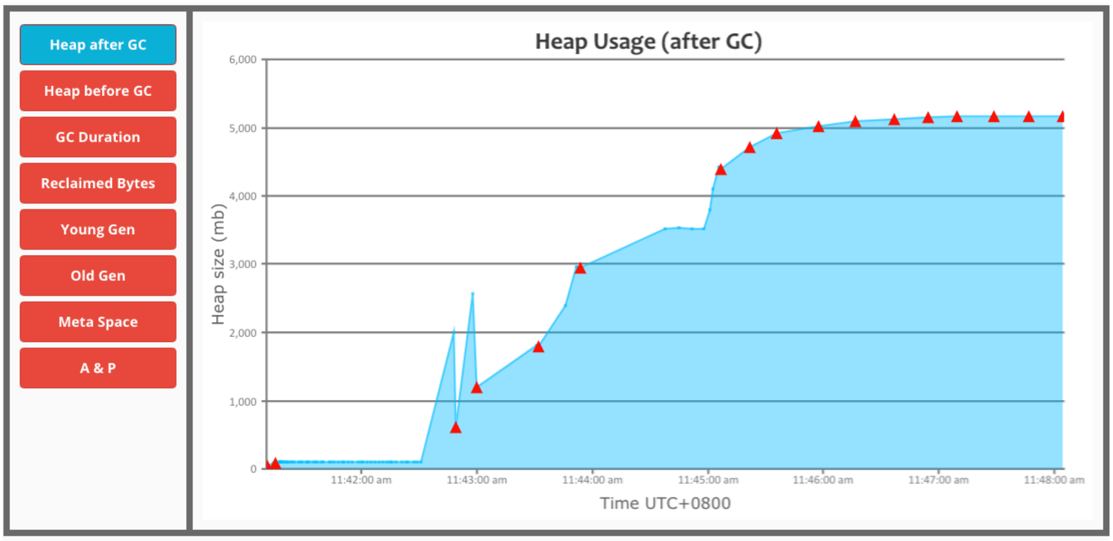
=> E2 continues to aggregates the shuffled records of 2nd shuffle. However, E2 encounters an OOM error while shuffling.



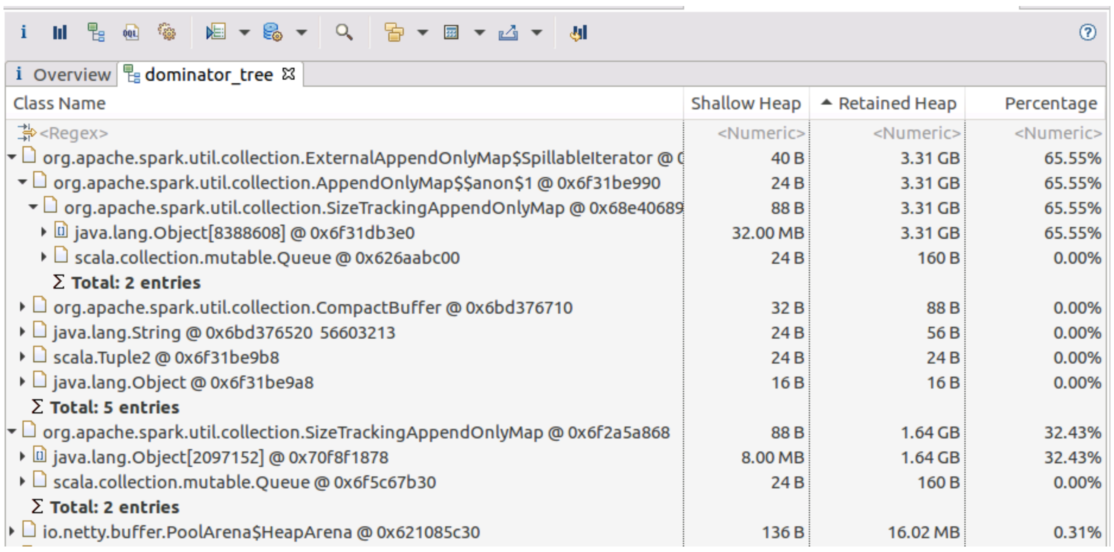
```
17/12/05 11:45:06 DEBUG TaskMemoryManager: [Require] Task 223 required 2.1 GB and got 2.1 GB for org.apache.spark.util.collection.ExternalAppendOnlyMap@72499f7a
17/12/05 11:45:06 DEBUG TaskMemoryManager: [Acquired] Task 223 finally acquired 2.1 GB (currentMem = 2.4 GB) for
org.apache.spark.util.collection.ExternalAppendOnlyMap@72499f7a
```

```
17/12/05 11:49:43 ERROR Executor: Exception in task 28.0 in stage 1.0 (TID 223)
java.lang.OutOfMemoryError: Java heap space
        at org.apache.spark.util.collection.AppendOnlyMap.growTable(AppendOnlyMap.scala:218)
        at org.apache.spark.util.collection.SizeTrackingAppendOnlyMap.growTable(SizeTrackingAppendOnlyMap.scala:38)
        at org.apache.spark.util.collection.AppendOnlyMap.incrementSize(AppendOnlyMap.scala:204)
        at org.apache.spark.util.collection.AppendOnlyMap.changeValue(AppendOnlyMap.scala:147)
        at org.apache.spark.util.collection.SizeTrackingAppendOnlyMap.changeValue(SizeTrackingAppendOnlyMap.scala:32)
        at org.apache.spark.util.collection.ExternalAppendOnlyMap.insertAll(ExternalAppendOnlyMap.scala:192)
        at org.apache.spark.Aggregator.combineValuesByKey(Aggregator.scala:41)
        at org.apache.spark.shuffle.BlockStoreShuffleReader.read(BlockStoreShuffleReader.scala:91)
        at org.apache.spark.rdd.ShuffledRDD.compute(ShuffledRDD.scala:109)
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:323)
        at org.apache.spark.rdd.RDD$$anonfun$8.apply(RDD.scala:336)
        at org.apache.spark.rdd.RDD$$anonfun$8.apply(RDD.scala:334)
        at org.apache.spark.storage.BlockManager$$anonfun$doPutIterator$1.apply(BlockManager.scala:1005)
        at org.apache.spark.storage.BlockManager$$anonfun$doPutIterator$1.apply(BlockManager.scala:996)
        at org.apache.spark.storage.BlockManager.doPut(BlockManager.scala:936)
        at org.apache.spark.storage.BlockManager.doPutIterator(BlockManager.scala:996)
        at org.apache.spark.storage.BlockManager.getOrElseUpdate(BlockManager.scala:700)
        at org.apache.spark.rdd.RDD.getOrCompute(RDD.scala:334)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:285)
        at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:38)
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:323)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:287)
        at org.apache.spark.rdd.CoGroupedRDD$$anonfun$compute$2.apply(CoGroupedRDD.scala:141)
        at org.apache.spark.rdd.CoGroupedRDD$$anonfun$compute$2.apply(CoGroupedRDD.scala:137)
        at scala.collection.TraversableLike$WithFilter$$anonfun$foreach$1.apply(TraversableLike.scala:733)
        at scala.collection.immutable.List.foreach(List.scala:381)
        at scala.collection.TraversableLike$WithFilter.foreach(TraversableLike.scala:732)
        at org.apache.spark.rdd.CoGroupedRDD.compute(CoGroupedRDD.scala:137)
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:323)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:287)
        at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:38)
```

[Guess] The task memory usage below reveals that there is not memory drop down. So, the cause may be that the 3.3GB *ExternalAppendOnlyMap* (E1) is not actually released by the TaskMemoryManger.

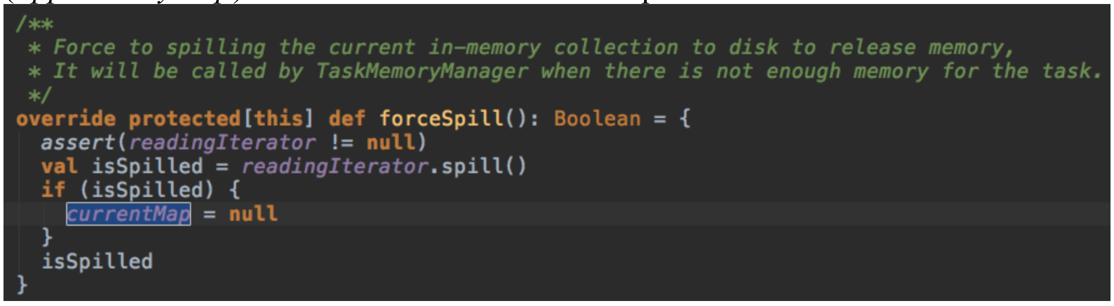


[Root cause] After analyzing the heap dump, I found the guess is right (the 3.3GB *ExternalAppendOnlyMap* is actually not released). The 1.6GB object is *ExternalAppendOnlyMap (E2).*



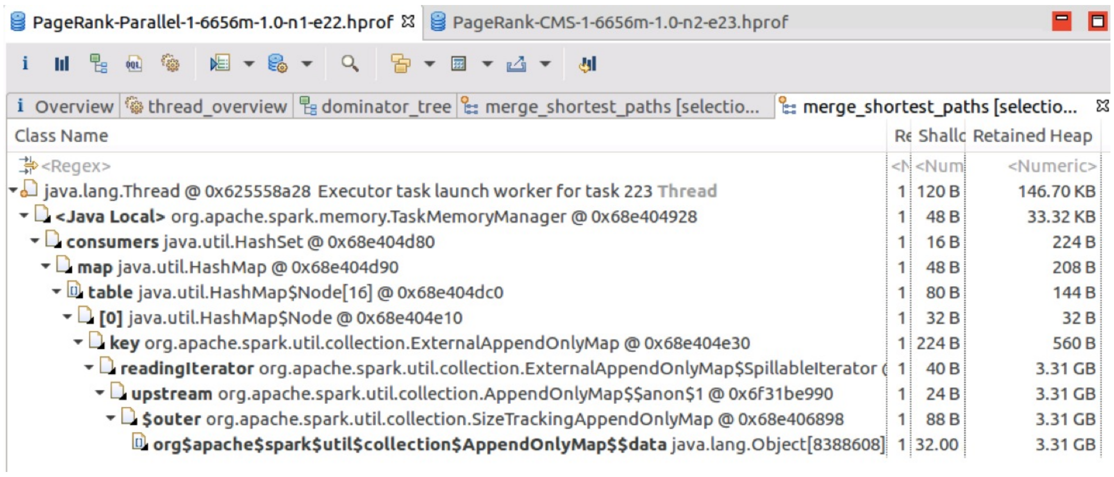
```
i Overview | dominator_tree

Class Name                                                                   Shallow Heap  Retained Heap  Percentage
<Regex>                                                                         <Numeric>      <Numeric>   <Numeric>
▾ org.apache.spark.util.collection.ExternalAppendOnlyMap$SpillableIterator @ (        40 B        3.31 GB      65.55%
  ▾ org.apache.spark.util.collection.AppendOnlyMap$$anon$1 @ 0x6f31be990               24 B        3.31 GB      65.55%
    ▾ org.apache.spark.util.collection.SizeTrackingAppendOnlyMap @ 0x68e40689          88 B        3.31 GB      65.55%
      ▸ java.lang.Object[8388608] @ 0x6f31db3e0                                     32.00 MB        3.31 GB      65.55%
      ▸ scala.collection.mutable.Queue @ 0x626aabc00                                   24 B          160 B       0.00%
        Σ Total: 2 entries
  ▸ org.apache.spark.util.collection.CompactBuffer @ 0x6bd376710                      32 B           88 B       0.00%
  ▸ java.lang.String @ 0x6bd376520  56603213                                          24 B           56 B       0.00%
  ▸ scala.Tuple2 @ 0x6f31be9b8                                                        24 B           24 B       0.00%
  ▸ java.lang.Object @ 0x6f31be9a8                                                    16 B           16 B       0.00%
    Σ Total: 5 entries
▾ org.apache.spark.util.collection.SizeTrackingAppendOnlyMap @ 0x6f2a5a868            88 B        1.64 GB      32.43%
  ▸ java.lang.Object[2097152] @ 0x70f8f1878                                        8.00 MB        1.64 GB      32.43%
  ▸ scala.collection.mutable.Queue @ 0x6f5c67b30                                      24 B          160 B       0.00%
    Σ Total: 2 entries
▸ io.netty.buffer.PoolArena$HeapArena @ 0x621085c30                                  136 B       16.02 MB       0.31%
```

[Question] Why the released *ExternalAppendOnlyMap* is still in memory? The source code of *ExternalAppendOnlyMap* shows that the *currentMap* (*AppendOnlyMap*) has been set to *null* when the spill action is finished.



```
/**
 * Force to spilling the current in-memory collection to disk to release memory,
 * It will be called by TaskMemoryManager when there is not enough memory for the task.
 */
override protected[this] def forceSpill(): Boolean = {
  assert(readingIterator != null)
  val isSpilled = readingIterator.spill()
  if (isSpilled) {
    currentMap = null
  }
  isSpilled
}
```

[Root cause in the source code] I further analyze the reference chain of unreleased *ExternalAppendOnlyMap*. The reference chain shows that the 3.3GB *ExternalAppendOnlyMap* is still referenced by the *upstream/readingIterator* and further referenced by *TaskMemoryManager* as follows. So, the root cause in the source code is that the *ExternalAppendOnlyMap* is still referenced by other iterators (setting the *currentMap* to *null* is not enough).



```
PageRank-Parallel-1-6656m-1.0-n1-e22.hprof | PageRank-CMS-1-6656m-1.0-n2-e23.hprof

i Overview | thread_overview | dominator_tree | merge_shortest_paths [selectio... | merge_shortest_paths [selectio...

Class Name                                                                                         Re  Shallc  Retained Heap
<Regex>                                                                                            <N  <Num       <Numeric>
▾ java.lang.Thread @ 0x625558a28  Executor task launch worker for task 223 Thread                  1  120 B       146.70 KB
  ▾ <Java Local> org.apache.spark.memory.TaskMemoryManager @ 0x68e404928                          1   48 B        33.32 KB
    ▾ consumers java.util.HashSet @ 0x68e404d80                                                   1   16 B           224 B
      ▾ map java.util.HashMap @ 0x68e404d90                                                       1   48 B           208 B
        ▾ table java.util.HashMap$Node[16] @ 0x68e404dc0                                          1   80 B           144 B
          ▾ [0] java.util.HashMap$Node @ 0x68e404e10                                              1   32 B            32 B
            ▾ key org.apache.spark.util.collection.ExternalAppendOnlyMap @ 0x68e404e30            1  224 B           560 B
              ▾ readingIterator org.apache.spark.util.collection.ExternalAppendOnlyMap$SpillableIterator (  1   40 B   3.31 GB
                ▾ upstream org.apache.spark.util.collection.AppendOnlyMap$$anon$1 @ 0x6f31be990   1   24 B         3.31 GB
                  ▾ $outer org.apache.spark.util.collection.SizeTrackingAppendOnlyMap @ 0x68e406898  1   88 B      3.31 GB
                      org$apache$spark$util$collection$AppendOnlyMap$$data java.lang.Object[8388608]  1  32.00   3.31 GB
```

[Potential solution]

Setting the *upstream/readingIterator* to *null* after the *forceSpill*() action. I will try this solution in these days.

[References]

[1] PageRank source code. https://github.com/JerryLead/SparkGC/blob/master/src/main/scala/applications/graph/PageRank.scala
[2] Task execution log. https://github.com/JerryLead/Misc/blob/master/OOM-TasksMemoryManager/log/TaskExecutionLog.txt