An Empirical Study of Out of Memory Errors in Apache Spark

Caused Identified by:

Cause identified by	**No.**
Users themselves	33
Experts	29
Us	5
Total	67

Cases:

Sources	User-definedcode	GraphX	MLlib	#Total	#Reproduced	#Unknown
StackOverflow	16	1	2	19	3	26
Spark mailing list	42	1	5	48	14	80
Total	58	2	7	67	17	106

Cause patterns:

Category	Pattern	Pattern description	No.	Ratio	
Large data stored in memory	Large buffered data		2		
	Large cached data		7		
Abnormal dataflow	Improper data partition		14		
	Hotspot key		7		
	Large single record		1		
Memory-consuming user code	Large external data		0		
	Large intermediate results		2		
	Large accumulated results		10		
Driver	Large generated results		9		
	Large collected results		15		
Total					

Fix suggestions:

Suggestions	Related pattern	No.
Lower buffer size	Large data buffers	2
Lower cache size	Large cached data	2
Lower storage level		3
Add partition number, repartition	Improper data partition	10
Change partition function		1
Change key (add an auxiliary key)	Hotspot key	1
Sub-divide the group, cut down the data associated to a single key		2
Aggregate partial values for each group		1
write multiple times, split the single large record	Large single record	1
Add partition number, change storage	Large intermediate results	
Adjust the parameter, moving the array outside the iterator (for reuse), add partition number (2)	Large accumulated results	
	Large external data	
Reduce partition number	Large generates large results	
Remove collect() (2) Reduce task number Tree reduce, tree aggregation	Large collected results	

Use a small k		

Large buffered data

1. A: org.apache.spark.shuffle.MetadataFetchFailedException: Missing an output location for shuffle 0

User:

We used JavaPairRDD.repartitionAndSortWithinPartitions on 100GB data and it kept failing similarly to your app. Then we looked at the Yarn logs on the specific nodes and found out that we have some kind of out-of-memory problem, so the Yarn interrupted the execution. Our solution was to change/add spark.shuffle.memoryFraction 0 in .../spark/conf/spark-defaults.conf. That allowed us to handle a much larger (but unfortunately not infinite) amount of data this way.

Job type: User-defined (StackOverflow)
Causes: Large buffered data (User)
Fix suggestions: lower the buffer size (user)

2. MLLib /ALS : java.lang.OutOfMemoryError: Java heap space

User:I am running into an out of memory error while running ALS using MLLIB on a reasonably small data set consisting of around 6 Million ratings.
at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:158)
at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:99)
Expert: I am not sure this can help you. I have 57 million rating,about 4million user and 4k items. I used 7-14 total-executor-cores,executal-memory 13g,cluster have 4 nodes,each have 4cores,max memory 16g.
I found set as follows may help avoid this problem:

```
    conf.set("spark.shuffle.memoryFraction","0.65") //default is 0.2
    conf.set("spark.storage.memoryFraction","0.3")//default is 0.6
```

I have to set rank value under 40, otherwise occure this problem.

Job type: User-defined (Mailing list)
Causes: Large buffered data (User)
Fix suggestions: decrease the rank value, lower the buffer size, lower the cache limit (Expert)

Large cached data

1. tiers of caching

User:i noticed that some algorithms such as graphx liberally cache RDDs for efficiency, which makes sense. however it can also leave a long trail of unused yet cached RDDs, that might push other RDDs out of memory.
Expert: I think tiers/priorities for caching are a very good idea and I'd be interested to see what others think. In addition to letting libraries cache RDDs liberally, it could also unify memory management across other parts of Spark. For example, small shuffles benefit from explicitly keeping the shuffle outputs in memory rather than writing it to disk, possibly due to filesystem overhead. To prevent in-memory shuffle outputs from competing with application RDDs, Spark could mark them as lower-priority and specify that they should be dropped to disk when memory runs low.

Job type: User-defined (Mailing list)
Causes: Large cached data (User)
Fix suggestions: no

2. Kyro serialization slow and runs OOM

User:when I load my dataset, transform it with some one to one transformations, and try to **cache the eventual RDD** - it runs really slow and then runs out of memory. When I remove Kyro serializer and default back to java serialization it works just fine and is able to load and cache the 700Gs of resultant data.

Job type: User-defined (Mailing list)
Causes: Large cached data (User)
Fix suggestions: no

3. Problems with broadcast large datastructure

User: Spark repeatedly fails broadcast a large object on a cluster of 25 machines for me. I have a node of 20 machines, and I just run the broadcast example, what I do is just change the data size in the example, to 400M, this is really a small data size. but I occurred the same problem with you .
Expert: 400MB isn't really that big. Broadcast is expected to work with several GB of data and in even larger clusters (100s of machines).
if you are using the default HttpBroadcast, then akka isn't used to move the broadcasted data. **But block manager can run out of memory if you repetitively broadcast large objects.** Another scenario is that the master isn't receiving any heartbeats from the blockmanager because the control messages are getting dropped due to bulk data movement. Can you provide a bit more details on your network setup?
Also, you can try doing a binary search over the size of broadcasted data to see at what size it breaks (i.e, try to broadcast 10, then 20, then 40 etc etc.)? Also, limit each run to a single iteration in the example (right now, it tries to broadcast 3 consecutive times).

If you are using a newer branch, you can also try the new TorrentBroadcast implementation.
your code is broadcasting 400MB 30 times, which are not being evicted from the cache fast enough, which, I think, is causing blockManagers to run out of memory.

Job type: User-defined (Mailing list)
Causes: Large cached data (Expert)
Fix suggestions: Try TorrentBroadcast

4. Spark streaming questions

User: Can someone explain the usage of cache w.r.t spark streaming? For example if we do stream.cache(), will the cache remain constatnt with all the partitions of rdd present across the nodes for that stream, OR will it be regularly updated as in while new batch is coming?
Expert: If you call DStream.persist (persist == cache = true), then all RDDs generated by the DStream will be persisted in the cache (in the BlockManager). As new RDDs are generated and persisted, old RDDs from the same DStream will fall out of memory. either by LRU or explicitly if spark.streaming.unpersist is set to true.
Well it is clear that the combineByKey is taking the most amount of time and 7 seconds. So you need to increase the number of reducers in the reduceByKeyAndWindow operation. That should distribute the computation more to use all the cores, and therefore speed up the processing of each batch.

Job type: Streaming (Mailing list)
Causes: Large cached data (Expert)
Fix suggestions: Change storage level, increase reduce number, add combineByKey()

5. [OOM when calling cache on RDD with big data]

User:I have a very simple job that simply caches the hadoopRDD by calling cache/persist on it. I tried MEMORY_ONLY, MEMORY_DISK and DISK_ONLY for caching strategy, I always get OOM on executors. And it works fine if I do not call cache or persist on the RDD:
Expert:

Job type: User-defined (Mailing list)
Causes: Large cached data (User)
Fix suggestions: no

6. [[Graphx] some problem about using SVDPlusPlus]

User: The implementation of SVDPlusPlus shows that it produces two new graph in each iteration which will also be cached to memory. However, **as the iteration goes on, more and more graph will be cached and out of memory happens.** So I think it maybe need to unpersist old graph which will not be used any more and add a few lines of code, the details are showed as follows:
Expert:

Job type: GraphX (Mailing list)
Causes: Large cached data (User)
Fix suggestions: no

7. [[0.9.0] MEMORY*AND*DISK_SER not falling back to disk]

User: My understanding of the MEMORY_AND_DISK_SER persistence level was that if an RDD could fit into memory then it would be left there (same as MEMORY_ONLY), and only if it was too big for memory would it spill to disk. Here's how the docs describe it:
What I'm observing though is that really large RDDs are actually causing OOMs. I'm not sure if this is a regression in 0.9.0 or if it has been this way for some time.
I dropped down to 0.5 but still OOM'd, so sent it all the way to 0.1 and didn't get an OOM.
Expert: This probably means that there’s not enough free memory for the “scratch” space used for computations, so we OOM before the Spark cache decides that it’s full and starts to spill stuff. Try reducing spark.storage.memoryFraction (default is 0.66, try 0.5).

Job type: User-defined (Mailing list)
Causes: Large cached data (Expert)
Fix suggestions: lower the cache size

Improper data partition

1. [Q: Spark runs out of memory when grouping by key]
User:
I am attempting to perform a simple transformation of common crawl data using Spark host on an EC2 using [this guide], my code looks like this:

So my basic question is, what is necessary to write a Spark task that can group by key with an almost unlimited amount of input without running out of memory?

Expert:
The most common cause of java.lang.OutOfMemoryError exceptions in shuffle tasks (such as groupByKey, reduceByKey, etc.) is low level of parallelism.

Job type: User-defined (StackOverflow)
Causes: Improper data partition (Expert)
Fix suggestions: add partition number

2. Q: Memory efficient way of union a sequence of RDDs from Files in Apache Spark

User:
I often run into out of memory situations even on 100 GB plus Machines. I run Spark in the application itself. I tried to tweak a little bit, but I am not able to perform this operation on more than 10 GB of textual data. The clear bottleneck of my implementation is the union of the previously computed RDDs, that where the out of memory exception comes from.

Expert:
for my use case that issue made the word2vec spark implementation a bit useless. Thus I used spark for massaging my corpus but not for actually getting the vectors.

- As other suggested stay away from calling `rdd.union`.
- Also I think `.toList` will probably gather every line from the RDD and collect it in your Driver Machine (the one used to submit the task) probably this is why you are getting out-of-memory. You should totally avoid turning the RDD into a list!

Job type: User-defined (StackOverflow)
Causes: Improper data partition (Expert)
Fix suggestions: add partition number

3. Q: Regarding spark input data partition and coalesce

User:
partition the input data(80 million records) into partitions using RDD.coalesce(numberOfPArtitions) before submitting it to mapper/reducer function. Without using coalesce() or repartition() on the input data spark executes really slow and fails with out of memory exception.

Expert:
Determining the number of partitions is a bit tricky. Spark by default will try and infer a sensible number of partitions.

Job type: User-defined (StackOverflow)
Causes: Improper data partition (User)
Fix suggestions: repartition() (User)

4. Q: Why does Spark RDD partition has 2GB limit for HDFS

User:
The Integer.MAX_SIZE is 2GB, it seems that some partition out of memory. So i repartiton my rdd partition to 1000, so that each partition could hold far less data as before. Finally, the problem is solved!!!
Expert:
It is a critical issue which prevents use of spark with very large datasets. Increasing the number of partitions can resolve it (like in OP's case), but is not always feasible, for instance when there is large chain of transformations part of which can increase data (flatMap etc) or in cases where data is skewed.

Job type: User-defined (StackOverflow)
Causes: Improper data partition (User)
Fix suggestions: add partition number (User)

5. A: Spark Java Error: Size exceeds Integer.MAX_VALUE

User:
I am trying to use spark for some simple machine learning task. I used pyspark and spark 1.2.0 to do a simple logistic regression problem. I have 1.2 million records for training, and I hashed the features of the records. When I set the number of hashed features as 1024, the program works fine, but when I set the number of hashed features as 16384, the program fails several times with the following error:

Expert:
I'm no Python coder, but when you "hashed the features of the records" you might be taking a very sparse set of records for a sample and creating an non-sparse array. This will mean a lot of memory for 16384 features. Particularly, when you do `zip(line[1].indices, line[1].data)`. The only reason that doesn't get you out of memory right there is the shitload of it you seem to have configured (50G).
Reason:

Thanks. This problem is just fixed by **using more partitions when loading the data**. We are just testing on small data set and gain some idea, then we are going to apply to big data set with much powerful machine.

Job type: User-defined (StackOverflow)
Causes: Improper data partition (User)
Fix suggestions: add partition number (User)

6. Q: Spark out of memory

User: The code I'm using: - reads TSV files, and extracts meaningful data to (String, String, String) triplets - afterwards some **filtering, mapping and grouping** is performed - finally, the data is reduced and some aggregates are calculated

I've been able to run this code with a single file (~200 MB of data), however I get a java.lang.OutOfMemoryError: GC overhead limit exceeded and/or a Java out of heap exception when adding more data (the application breaks with 6GB of data but I would like to use it with 150 GB of data).

I guess I would have to tune some parameters to make this work. I would appreciate any tips on how to approach this problem (how to debug for memory demands). I've tried increasing the 'spark.executor.memory' and using a smaller number of cores (the rational being that each core needs some heap space), but this didn't solve my problems.

typically, when **loading data from disk into a Spark RDD, the data consumes much more space in RAM than on disk**. This is paritally due to the overhead of making byte arrays into Java String objects.

I was thinking of something in the way of taking a chunk of data, processing it, storing partial results on disk (if needed), continuing with the next chunk until all are done, and finally merging partial results in the end.

Expert: If you repartition an RDD, it requires additional computation that has overhead above your heap size, try loading the file with more paralelism by decreasing split-size
in TextInputFormat.SPLIT_MINSIZE and TextInputFormat.SPLIT_MAXSIZE (if you're using TextInputFormat) to elevate the level of paralelism.

Try using mapPartition instead of map so you can handle the computation inside a partition. If the computation uses a temporary variable or instance and you're still facing out of memory, try lowering the number of data per partition (increasing the partition number)

Job type: User-defined (StackOverflow)
Causes: Improper data partition (Expert)
Fix suggestions: increase the partition number

7. Q: spark executor lost failure

User:
I am using the databricks spark cluster (AWS), and testing on my scala experiment. I have some issue when training on a 10 GB data with LogisticRegressionWithLBFGS algorithm. The code block where I met the issue is as follows:

First I got a lot executor lost failure and java out of memory issues, then I **repartitioned my training_set with more partitions and the out of memory issues are gone,** but Still get executor lost failure.
Expert:

Job type:MLlib (StackOverflow)
Causes: Improper data partition (User)
Fix suggestions: add partition number

8. distinct on huge dataset

User:
I have a huge 2.5TB file. When i run:
val t = sc.textFile("/user/hdfs/dump.csv")
t.distinct.count
Got a bit further, i think out of memory error was caused by **setting spark.spill to false**. Now i have this error, is there an easy way to increase file limit for spark
Expert: You might try setting spark.shuffle.spill to false and see if that runs any longer (turning off shuffle spill is dangerous, though, as it may cause Spark to OOM if your reduce partitions are too large).

Job type:User-defined (Mailing list)
Causes: Improper data partition (us, reproduced)
Fix suggestions: set spark.spill = true

9. Spark limitations question

User:
I'm currently testing a join of two data sets, Base and Skewed. They're both 100 million rows and they look like the following.
I have two tests:
1. join Base to itself, sum the "nums" and write out to HDFS
2. same as 1 except join Base to Skewed
(I realize the outputs are contrived and meaningless, but again, I'm testing limits here)
Test 2, however, works well on all but one of the nodes on the cluster. That node runs out of memory quickly and dies. All of those nodes have 10 gigs of memory available to the spark executor and remaining
I'm assuming there's a build up of the **skewed 50million rows on the one particular node** and is running out of memory while it tries to merge them.

Expert:
Job type:User-defined (Mailing list)
Causes: Improper data partition (us, reproduced)
Fix suggestions: no

10. What should happen if we try to cache more data than the cluster can hold in memory?

User: If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.

What I' m seeing per the discussion below is that when I try to cache more data than the cluster can hold in memory, I get:

Trying MEMORY_AND_DISK yields the same error.

Expert: It seems possible that you are running out of memory **unrolling a single partition of the RDD**. This is something that can cause your executor to OOM, especially if the cache is close to being full so the executor doesn't have much free memory left. How large are your executors? At the time of failure, is the cache already nearly full?A (potential) workaround would be to first persist your data to disk, then re-partition it, then cache it. I'm not 100% sure whether that will work though.

Job type:User-defined (Mailing list)
Causes: Improper data partition (Expert)
Fix suggestions: repartition (add partition number)

11. OOM writing out sorted RDD

User: Eventually, the logs show an OOM [1] included at the end of the mail followed by another TID loss to "FileSystem closed" errors indicated in stacktrace [2]. I noticed that **the partitions may be skewed as a result of the sort**, **with one or two paritions having upto 10% of all rows.** I also noticed that the data written out until the 50% stage (when memory shoots up) had a large number of empty part- files followed by a few files of 200M in size. While I could attempt to **partition manually (choosing custom ranges for a range partitioner**), it appears that limiting read sizes (from the earlier shuffle) during the write to HDFS should help successfully write out even overloaded partitions as well.
as expected, switching to kryo merely delays the inevitable. Does anyone have experience controlling memory consumption while processing (e.g. writing out) **imbalanced partitions**?

Job type:User-defined (Mailing list)
Causes: Improper data partition (User)
Fix suggestions: change partition function

12. Only master is really busy at KMeans training

User:
The training data is supplied in this form:
```
var vectors2 = vectors.repartition(1000).persist(org.apache.spark.storage.StorageLevel.MEMORY_AND_DISK_SER)
var broadcastVector = sc.broadcast(vectors2)
```

The 1000 partitions is something that could probably be optimized, but too few will cause OOM errors.As far as I can see, it's the repartitioning that is causingthe problems. However, without that, I have only one partition for further RDD operations on dict, so it seems to be necessary.

Expert:

Job type:User-defined (Mailing list)
Causes: Improper data partition (User)
Fix suggestions: repartition (add partition number)

13. RDD Blocks skewing to just few executors

User:
As you can see most of the **data is skewing to just 2 executors, with 1 getting more than half the Blocks. These become a hotspot and eventually I start seeing OOM errors.** I've tried this a half a dozen times and the 'hot' executors changes, but not the skewing behavior.
Expert:

Job type:User-defined (Mailing list)
Causes: Improper data partition (User)
Fix suggestions: no

14. Using Spark on Data size larger than Memory size

User: Thank you. I'm using mapPartitions() but as you say, it requires that every partition fit in memory. This will work for now but may not always work so I was wondering about another way.
Expert: If an individual partition becomes too large to fit in memory then the usual approach would be to repartition to more partitions, so each one is smaller. Hopefully then it would fit.you can OOM under certain configurations, and it's possible you'll need to change from the default configuration if you're using doing very memory-intensive jobs. However, there are very few cases where Spark would simply fail as a matter of course -- for instance, you can always increase the number of partitions to decrease the size of any given one. or repartition data to eliminate skew.

Job type:User-defined (Mailing list)
Causes: Improper data partition (Expert)
Fix suggestions: repartition (add partition number)

Hotspot key

1. Q: Spark: out of memory exception caused by groupbykey operation

User :
I have a large file where each line is a record with an id as key.
so the lines with the same id will be shuffled to one worker node. a tuple in RDD groupedLines is like *id -> Iterable(line1, line2, ..., lineN)* if **lots of lines have the same id, then the size of the tuple's value *Iterable(...)* will be quite large**, and if it is larger then the JVM memory limit of the process on the machine, out of memory problem may happen.
Expert:
def groupByKey(numPartitions: Int): RDD[(K, Seq[V])]

Try to increase parameter

```
numPartitions
```

Job type: User-defined (StackOverflow)
Causes: Hotspot Key (User, Reproduced)
Fix suggestions: no

2. Partitioning - optimization or requirement?

User :
I'm often running out of memory when doing unbalanced joins (ie. **cardinality of some joined elements much larger than others**). Raising the memory ceiling fixes the problem, but that's a slippery slope.
Expert:
Job type: User-defined (Mailing list)
Causes: Hotspot Key (User, reproduced)
Fix suggestions: no

3. Lag function equivalent in an RDD

User :
Example:
val pairs = PositionPair.mapValues(kv=> List((kv._2.getTime, kv._2.getPos)))
//results an RDD with Int as Key and (Date, Position) as value.

The question: Is there a way to create a line segment for adjacent position for the same vehicle in a memory efficient way?
PS: I have tried reduceByKey and then splitting the List of position in tuples. For me **it runs out of memory every time because of the volume of data.**

Expert: You say you reduceByKey but are you really collecting all the tuples for a vehicle in a collection, like what groupByKey does already? Yes, **if one vehicle has a huge amount of data that could fail.**

Maybe you can consider using something like vehicle and *day* as a key. This would make you process each day of data separately, but if that's fine for you, might **drastically cut down the data associated to a single key**.

Job type: User-defined (Mailing list)
Causes: Hotspot Key (Expert)
Fix suggestions: change key, cut down the data associated to a single key (Expert)

4. Understanding RDD.GroupBy OutOfMemory Exceptions

User :
I'm trying to do a simple groupByKey (see below), but it fails with a java.lang.OutOfMemoryError: GC overhead limit exceeded exception.
I can count the group sizes using reduceByKey without problems, ensuring myself the problem isn't caused by a single excessively large group, nor by an excessive amount of groups.
I've tried reformatting, reshuffling and increasing the groupBy level of parallelism:
```
  keyvals.groupByKey(24).count // fails
  keyvals.groupByKey(3000).count // fails
  keyvals.coalesce(24, true).groupByKey(24).count // fails
  keyvals.coalesce(3000, true).groupByKey(3000).count // fails
```

I've tried playing around with spark.default.parallelism, and increasing spark.shuffle.memoryFraction to 0.8 while lowering spark.storage.memoryFraction to 0.1.

Expert: The groupBy operator in Spark is not an aggregation operator (e.g. in SQL where you do select sum(salary) group by age...) - there are separate more efficient operators for aggregations. Currently groupBy requires that **all of the values for one key can fit in memory**. In your case, it's possible you have a single key with a very large number of values, given that your count seems to be failing on a single task.Within a partition things will spill - so the current documentation is correct. **This spilling can only occur *across keys* at the moment. Spilling cannot occur within a key at present**.

Job type: User-defined (Mailing list)
Causes: Hotspot Key (Expert, reproduced)
Fix suggestions: change key, sub-dividing any very large groups, aggregating partial values for each group (Expert)

5. OutOfMemory in "cogroup"

User:There are too many values for a special key and these values cannot fit into memory. Spilling data to disk helps nothing because cogroup needs to read all values for a key into memory.
Expert: The problem is that the estimated size of used memory is inaccurate. I dig into the codes and found that SizeEstimator.visitArray randomly selects 100 cell and use them to estimate the memory size of the whole array. In our case, most of the biggest cells are not selected by SizeEstimator.visitArray, which causes the big gap between the estimated size (27M) and the real one (5G). spark.shuffle.safetyFraction is useless for this case because the gap is really huge. Here 100 is controlled by SizeEstimator.ARRAY_SAMPLE_SIZE and there is no configuration to change it.

Job type: User-defined (Mailing list)
Causes: Hotspot Key (User, reproduced)
Fix suggestions: no

6. GroupBy Key and then sort values with the group

User:I am having a similar issue, but **I have a lot of data for a group & I cannot materialize the iterable into a List or Seq in memory.** [I tried & it runs into OOM]. is there any other way to do this ? I also tried a secondary-sort, with the key having the "group::time", but the problem with that is the same group-name ends up in multiple partitions & I am having to run sortByKey with one partition - sortByKey(true, 1) which shuffles a lot of data..

Expert: There is a new API called repartitionAndSortWithinPartitions() in master, it may help in this case, then you should do the `groupBy()` by yourself.

Job type: User-defined (Mailing list)
Causes: Hotspot Key (User, reproduced)
Fix suggestions: no

7. Sorting partitions in Java

User:I'm trying to sort data in each partition of an RDD.
I was able to do it successfully in Scala like this:

```
val sorted = rdd.mapPartitions(iter => {
  iter.toArray.sortWith((x, y) => x._2.compare(y._2) < 0).iterator
},
```

preservesPartitioning = true)

I used the same technique as in OrderedRDDFunctions.scala, so I assume it's a reasonable way to do it.
I think the code I have will work for me, but I can imagine conditions where it will run out of memory

Expert:sortByKey currently requires partitions to fit in memory, but there are plans to add external sort, It's an Iterator in both Java and Scala. In both cases you need to copy the stream of values into something List-like to sort it. An Iterable would not change that (not sure the API can promise many iterations anyway).

Job type: User-defined (Mailing list)
Causes: Hotspot Key (us, reproduced)
Fix suggestions: no

Large single record

1. OOM with groupBy + saveAsTextFile

User:I'm trying to run groupBy(function) followed by saveAsTextFile on an RDD of count ~ 100 million. The data size is 20GB and groupBy results in an RDD of 1061 keys with values being Iterable<Tuple4<String, Integer, Double, String>>.My understanding is that if each host is processing a single logical partition to saveAsTextFile and is reading from other hosts to write out the RDD, it is unlikely that it would run out of memory. My interpretation of the spark tuning guide is that the degree of parallelism has little impact in case (1) above since max cores = number of hosts. Can someone explain why there are still OOM's with 100G being available?

Expert: None of your tuning will help here because the problem is actually the way you are saving the output. If you take a look at the stacktrace, it is trying to **build a single string that is too large for the VM** to allocate memory. The VM is actually not running out of memory, but rather, JVM cannot support a single String so large.I suspect this is due to the fact that the **value in your key, value pair after group by is too long (maybe it concatenates every single record)**. Do you really want to save the key, value output this way using a text file? Maybe you can write them out as multiple strings rather than a single super giant string.

Job type: User-defined (Mailing list)
Causes: Large single record (Expert, reproduced)
Fix suggestions: write multiple times (split the single large record)

Large intermediate results

1. OutOfMemory Error

User:I am trying to implement machine learning algorithms on Spark. I am working on a 3 node cluster, with each node having 5GB of memory. Whenever I am working with slightly more number of records, I end up with OutOfMemory Error. Problem is, even if **number of records is slightly high, the intermediate result from a transformation is huge and this results in OutOfMemory Error.** To overcome this, we are partitioning the data such that each partition has only a few records.

Expert: Increase the number of partitions 3. You could try persisting the RDD to use DISK_ONLY. If increasing the partitions is the only the way, then one might end up with OutOfMemory Errors, when working with certain algorithms where intermediate result is huge.
Job type: User-defined (Mailing list)
Causes: Large intermediate results (User)

Fix suggestions: add partition number, change storage level

2. MLLib ALS question

User:I'm trying to use Matrix Factorization over a dataset with like 6.5M users, 2.5M products and 120M ratings over products. The test is done in standalone mode, with unique worker (Quad-core and 16 Gb RAM).
The program runs out of memory, and I think that this happens because flatMap holds data in memory.
(I tried with Movielens dataset that has 65k users, 11k movies and 100M ratings and the test does it without any problem)

Expert:The current ALS implementation constructs all subproblems in memory. With rank=10, that means (6.5M + 2.5M) * 10^2 / 2 * 8 bytes = 3.5GB. The ratings need 2GB, not counting the overhead. ALS creates in/out blocks to optimize the computation, which takes about twice as much as the original dataset. Note that this optimization becomes "overhead" on a single machine. All these factors contribute to the OOM error.ALS still needs to load and deserialize the in/out blocks (one by one)
from disk and then construct least squares subproblems. All happen in
RAM. The final model is also stored in memory.

Job type: MLlib(Mailing list)
Causes: Large intermediate results (Expert, reproduced)
Fix suggestions: no

Large accumulated results

1. Q: Memory issues when running Spark job on relatively large input

User:
Particularly allocating a buffer of size 40M for each file in order to read the content of the file using BufferedInputStream. This causes the stack memory to end at some point.

The thing is:

- If I read line by line (which does not require a buffer), it will be very non-efficient read
- If I allocate one buffer and reuse it for each file read - is it possible in parallelism sense? Or will it get overwritten by several threads?

Expert:
It seems like you are reading the content of all input files into an in-memory ArrayList? This sort of defeats the purpose of working with RDDs/partitions,

Job type: User-defined (StackOverflow)
Causes: Large accumulated results (Expert, Reproduced)
Fix suggestions: moving the byte array allocation outside the iterator, so it gets reused by all partition elements

2. Q: spark aggregatebykey with collection as zerovalue

User:
I'm working with a rdd of tuples [k, v(date, label)] and I'm trying to get all the distinct labels and the min of date for each keys.

I've ended with this piece of code :`aggregateByKey((new DateTime(), new mutable.HashSet[String]()))((acc: (DateTime, mutable.HashSet[String]), v: (DateTime, String)) => (if (acc._1.isBefore(v._1)) acc._1 else v._1, acc._2 + v._2), (acc1: (DateTime, mutable.HashSet[String]), acc2: (DateTime, mutable.HashSet[String])) => (if (acc1._1.isBefore(acc2._1)) acc1._1 else acc2._1, acc1._2 ++ acc2._2))`

Expert:
Ok, there are many things going on here, so let's go one by one:

1. groupByKey will just shuffle all data for a key to a single executor, load it into memory and make it available for you to do whatever you want (aggregation or not). This is an immediate cause of possible OutOfMemoryErrors if there is a lot of data associated with any given key (skewed data).

2. aggregateByKey will try to be smarter. Since it knows that is aggregating, it will try to aggregate locally before shuffling anything. The methods and zero-value you provide are serialize to multiple executors in order to accomplish just this. So your aggregation logic will be distributed even for the same key. Only accumulators will be serialized and merged. So overall, this method is significantly better in most cases, but you have to be careful still if (like in this case) the size of the accumulator itself can grow without bounds. Relevant questions: How many strings you expect per key? How big are these strings? How much de-duplication you expect to happen?

3. Another thing you can do is to take this piece of advice from aggregateByKey's documentation:

Job type: User-defined (StackOverflow)
Causes: Large accumulated results (Expert)
Fix suggestions: increase the number of partitions

3. Q: Spark: Out Of Memory Error when I save to HDFS

User:
I am experiencing OOME when I save big data to hdfs

```
val accumulableCollection = sc.accumulableCollection(ArrayBuffer[String]()) val rdd = textfile.filter(row => {
            if (row.endsWith(",")) {
            accumulableCollection += row
            false
} else if (row.length < 100) {
            accumulableCollection += row
            false
}
            valid
})
```

the accumulableCollection that will be written in HDFS has the max size of 840MB or 1.3M rows. in this scenario I am just writing 146MB of data. yes, the accumulableCollection that will be written in HDFS has the max size of 840MB or 1.3M rows. in this scenario I am just writing 146MB of data.
Expert:
It means pretty much what it says. You are trying to serialize a single object which is very large. You should probably rewrite your code to not do this.

Job type: User-defined (StackOverflow)
Causes: Large accumulated in accumulableCollection (Expert, reproduced)
Fix suggestions: no

4. Common crawl parsing has high fan out and runs out of memory

User:I am trying to parse the commoncrawl.org data, which is stored in amazon s3 as a series of (not so) large (1G) gzip files. My goal is to call flatMap which will receive a gzip file as input and yield a few dozen thousands html blobs as result. See below the code with some boilerplate stripped.

```
WarcReaderCompressed wrc = new WarcReaderCompressed(new FileInputStream(gzipPath));
List<string> htmls = new ArrayList<String>();
```

Expert:

Job type: User-defined (Mailing list)

Causes: Large accumulated results (us, reproduced)

Fix suggestions: no

5. Bulk-load to HBase

User:I have to merge the byte[]s that have the same key. If merging is done with reduceByKey(), a lot of intermediate byte[] allocation and System.arraycopy() is executed, and it is too slow. So I had to resort to groupByKey(), and in the callback allocate the byte[] that has the total size of the byte[]s, and arraycopy() into it. groupByKey() works for this, since the size of the group is manageable in my application.

Expert: The problem is that you will fi**rst collect and allocate many small byte[] in memory, and then merge them.If the total size of the byte[]s is very large, you run out of memory, as you observe.** If you want to do this, use more executor memory. You may find it's not worth the tradeoff of having more, smaller executors merging pieces of the overall byte[] array.

Job type: User-defined (Mailing list)

Causes: Large accumulated results (Expert)

Fix suggestions: no

6. Help alleviating OOM errors

User:our cluster seems to have a really hard time with OOM errors on the executor.

Expert: You need to configure the spark.shuffle.spill true again in the config, What is causing you to OOM, it could be that you are trying to just simply sortbykey & keys are bigger memory of executor causing the OOM, can you put the stack.When OOM occurs it could cause the RDD to spill to disk, the repeat task may be forced to read data from disk & cause the overall slowdown, not to mention the RDD may be send to different executor to be processed, are you seeing the slow tasks as process_local or node_local atleast?

Too few partitions: if one partition is too big, it may cause an OOM if there is not enough space to unroll the entire partition in memory. For the latter, I would reduce spark.storage.memoryFraction.Your application is using a lot of memory on its own: Spark be default assumes that it has 90% of the runtime memory in your JVM. **Ifyour application is super memory-intensive (e.g. creates large data structures),** then I would either try to reduce the memory footprint of your application itself if possible, or reduce the amount of memory Spark thinks it owns.

Job type: User-defined (Mailing list)

Causes: Large accumulated results (Expert)

Fix suggestions: add partition number

7. Memory footprint of Calliope: Spark -> Cassandra writes

User:I'm generating N records of about 50 bytes each and using the UPDATE mutator to insert them into C*.   I get OOM if my memory is below 1GB per million of records, or about 50Mb of raw data (without counting any RDD/structural overhead). (See code [1]). I just tried the code you posted in the gist (https://gist.github.com/maasg/68de6016bffe5e71b78c) and it does give a OOM. It is cause of the data being generated locally and then paralellized -

Expert: You need to configure the spark.shuffle.spill true again in the config, What is causing you to OOM, it could be that you are trying to just simply sortbykey & keys are bigger memory of executor causing the OOM, can you put the stack.When OOM occurs it could cause the RDD to spill to disk, the repeat task may be forced to read data from disk & cause the overall slowdown, not to mention the RDD may be send to different executor to be processed, are you seeing the slow tasks as process_local or node_local atleast?

Job type: User-defined (Mailing list)
Causes: Large accumulated results (us, reproduced)
Fix suggestions: no

8. Folding an RDD in order

User:I have data that looks like this:

user_id, transaction_timestamp, transaction_amount

And I'm interested in doing a foldByKey on user_id to sum transaction amounts - taking care to note approximately when a user surpasses a total transaction threshold. I'm using RangePartitioner to make sure that data is ordered sequentially between partitions, and I'd also make sure that data is sorted within partitions, though I'm not sure how to do this exactly (I was going to look at the code for sortByKey to figure this out - I believe sorting in place in a mapPartitions should work).

Expert:you may use mutable Map for optimized performance. One thing to notice, foldByKey is a transformation, while aggregate is an action. The final result of the code above is a single Map object rather than an RDD. **If this map can be very large (say you have billions of users), then aggregate may OOM**.

Job type: User-defined (Mailing list)
Causes: Large accumulated results (Expert)
Fix suggestions: no

9. [GitHub] incubator-spark pull request: MLLIB-25: Implicit ALS runs out of m...

User: Completely correct, but there's a subtle but quite large memory problem here. **map() is going to create all of these matrices in memory at once, when they don't need to ever all exist at the same time.**
    For example, if a partition has n = 100000 rows, and f = 200, then this intermediate product requires 32GB of heap. The computation will never work unless you can cough up workers with (more than) that much heap.

Expert:
Job type: User-defined (Mailing list)
Causes: Large accumulated results (User)
Fix suggestions: no

10. GroupByKey results in OOM - Any other alternative

User: There is a huge list of (key, value) pairs. I want to transform this to (key, distinct values) and then eventually to (key, distinct values count)

On small dataset

groupByKey().map(x => (x_1, x._2.distinct)) ...map(x => (x_1, x._2.distinct.count))

On large data set I am getting OOM.

Is there a way to represent Seq of values from groupByKey as RDD and then perform distinct over it ?
Expert:
Grouping by key is always problematic **since a key might have a huge number of values**. You can do a little better than grouping *all* values and *then* finding distinct values by using foldByKey, putting values into a Set. At least you end up with only distinct values in memory. (You don't need two maps either, right?)If the number of distinct values is still huge for some keys, consider the experimental method countApproxDistinctByKey
Job type: User-defined (Mailing list)
Causes: Large accumulated results + Hotspot Key (Expert, reproduced)
Fix suggestions: no

Large external data

Large data/results generated at driver

1. Q: Spark mllib svd gives: Java OutOfMemory Error
User:I am using the svd library of mllib to do some dimensionality reduction on a big matrix, the data is about 20G, and the spark memory is 60G, and I got the following warning and error message:
I think the error happens while java is copying an array. So I think this happens on the driver. How should I fix this? Use a larger driver memory?

The reason why I am getting the java memory error is because, **the computation for top eigenvectors are on the driver, so I need to make sure that I have enough memory on the driver node.When using `spark-submit` with `--driver-memory 5G`, the problem is solved.**

Expert:
The SVD operation you are calling happens on the driver. The covariance matrix is calculated on the cluster though. Where are you running out of memory? Driver right?

Job type: MLlib (StackOverflow)
Causes: Driver generate large results (Top eigenvectors) (User)
Fix suggestions: Add driver’ s memory space

2. A: How to use spark to generate huge amount of random integers?
User:
Notes: Now I just want to generate one number per line.

But it seems that when number of numbers gets larger, the program will report an error. Any idea with this piece of code?

Expert:

The current version is materializing the collection of random numbers in the memory of the driver. If that collection is very large, the driver will run out of memory.

Job type: User-defined (StackOverflow)
Causes: Driver generates large results (Expert)
Fix suggestions: no

3. Running out of memory Naive Bayes

User:
I've been trying to use the Naive Bayes classifier. Each example in the dataset is about 2 million features, only about 20-50 of which are non-zero, so the vectors are very sparse. I keep running out of memory though, even for about 1000 examples on 30gb RAM while the entire dataset is 4 million examples. And I would also like to note that I'm using the sparse vector class.
Expert:
Even the features are sparse, the conditional probabilities are stored in a dense matrix. With 200 labels and 2 million features, you need to store at least 4e8 doubles on the driver node. With multiple partitions, you may **need more memory on the driver**. Could you try reducing the number of partitions and giving driver more ram and see whether it can help
Job type: MLlib (Mailing list)
Causes: Driver generates large results (Expert, reproduced)
Fix suggestions: reduce the partition number

4. trouble with broadcast variables on pyspark

User:
I'm running into an issue when trying to broadcast large variables with pyspark.

A ~1GB array seems to be blowing up beyond the size of the driver machine's memory when it's pickled.

I've tried to get around this **by broadcasting smaller chunks of it one at a time. But I'm still running out of memory**, ostensibly because the intermediate pickled versions aren't getting garbage collected.
The driver JVM is hitting OutOfMemoryErrors, but the python process is taking even more memory.

Expert:
Job type: User-defined (Mailing list)
Causes: Driver generates/broadcasts large results (Expert)
Fix suggestions:no

5. advice on maintaining a production spark cluster?

User:I suspect that the loss of workers is tied to
jobs that run out of memory on the client side or our use of very large broadcast variables, but I don't have an isolated test case.
I'm open to general answers here: for example, perhaps we should simply be using mesos or yarn instead of stand-alone mode.

We've had occasional problems with running out of memory on the driver side (esp. **with large broadcast variables**) so that may be related.

Job type: User-defined (Mailing list)

Causes: Driver generates/broadcasts large results (User)
Fix suggestions:no

6. RowMatrix PCA out of heap space error

User:I got this error when trying to perform PCA on a sparse matrix, each row has a nominal length of 8000, and there are 36k rows. each row has on average 3 elements being non-zero.
I guess the total size is not that big.
Expert: The Gramian is 8000 x 8000, dense, and full of 8-byte doubles. It's symmetric so can get away with storing it in ~256MB. The catch is that it's going to send around copies of this 256MB array. You may easily be **running your driver out of memory** given all the overheads and copies, or your executors, of which there are probably 2 by default splitting 1GB.

Job type:MLlib (Mailing list)
Causes: Driver generates large results (User, reproduced)
Fix suggestions:no

7. newbie : java.lang.OutOfMemoryError: Java heap space

User: Disclaimer : Newbie (well, a returning user)
Setup :
20 nodes
-Dspark.executor.memory=40g , essentially tons of space for my usecase

Pretty straight forward join between two inputs
- 17G (distributed in 10 equally sized - 1.7g files)
- 49Mb (1 file)
I just need to join based on the keys and write out values from both as tuples
Expert: From the stack trace, it looks like the **driver program is dying trying to serialize data out to the workers**. My guess is that whatever machine you're running from has a relatively small default maximum heap size and trying to broadcast the 49MB file is causing it to run out of memory

Job type: User code (Mailing list)
Causes: Driver generates large results (Expert)
Fix suggestions:no

8. broadcast: OutOfMemoryError

User:i'm running into this OutOfMemory issue when i'm broadcasting a large array. what is the best way to handle this?

should i split the array into smaller arrays before broadcasting, and then combining them locally at each node?

Job type: User code (Mailing list)
Causes: Driver generates/broadcasts large results (User)
Fix suggestions:no

9. driver memory

User: In the application UI, it says my driver has 295 MB memory. I am trying to broadcast a variable that is 0.15 gigs and it is throwing OutOfMemory errors, so I am trying to see if by increasing the driver memory I can fix this.y trying to broadcast an array with 4 million elements, and a size of approximatively 150 MB. Every time I was trying to broadcast, I got an OutOfMemory error.

Job type: User code (Mailing list)

Causes: Driver generates/broadcasts large results (User)

Fix suggestions:no

Large results collected by driver

1. Q: Spark raises OutOfMemoryError

User: Take(all) used in the code

The file is about 20G and my computer have 8G ram, when I run the program in standalone mode, it raises the OutOfMemoryError:

Expert:

Spark can handle some case. But you are using `take` to f**orce Spark to fetch all of the data to an array(in memory)**. In such case, you should store them to files, like using `saveAsTextFile`.

If you are interested in looking at some of data, you can use `sample` or `takeSample`.

Job type: User-defined (StackOverflow)

Causes: Driver collect results (Expert)

Fix suggestions: no

2. Q: Spark OutOfMemoryError when adding executors

User:

Everything runs smoothly when I use few executors (10). But I got `OutOfMemoryError: Java heap space` on the driver when I try to use more executors (40). I think it might be related to the level of parallelism used (as indicated in https://spark.apache.org/docs/latest/tuning.html#level-of-parallelism).

Expert:

Do you mind testing 1.1-SNAPSHOT and allocating more memory to the driver? I think the problem is with the feature dimension. KDD data has more than 20M features and in v1.0.1, the driver collects the partial gradients one by one, sums them up, does the update, and then sends the new weights back to executors one by one. In 1.1-SNAPSHOT, we switched to multi-level tree aggregation and torrent broadcasting.

Job type: User-defined (StackOverflow)

Causes: Driver collects large results (User)

Fix suggestions: Multi-level tree aggregation (Expert)

3. Q: GraphX does not work with relatively big graphs

User:

Graph has 231359027 edges. And its file weights 4,524,716,369 bytes. Graph is represented in text format:

```
println(graph.edges.collect.length)     println(graph.vertices.collect.length)
```

Expert:

Job type: GraphX (StackOverflow)
Causes: Driver collects large results (User, reproduced)
Fix suggestions: Change collect().count() to count() (User)

4. Q: How to filter a RDD according to a function based another RDD in Spark?

User:
When the input data is very large, > 10GB for example, I always encounter a "java heap out of memory" error. I doubted if it's **caused by "weights.toArray.toMap", because it convert an distributed RDD to an Java object in JVM.** So I tried to filter with RDD directly:
Job type: User-defined (StackOverflow)
Causes: Driver collects large results (User)
Fix suggestions: no

5. A: How to iterate over large Cassandra table in small chunks in Spark

User:
In my test environment I have 1 Cassandra node and 3 Spark nodes. I want to iterate over apparently large table that has about 200k rows, each roughly taking 20-50KB.

I tried to only run `collect`, without `count` - in this case it just fails fast with `NoHostAvailableException`.

Expert:
Furthermore, you shouldn't use the `collect` action in your example because it will fetch all the rows in the driver application memory and may raise an out of memory exception. You can use the`collect` action only if you know for sure it will produce a small number of rows.

Job type: User-defined (StackOverflow)
Causes: Driver collects large results (Expert)
Fix suggestions: remove collect()

6. Memory allocation in the driver

User: It turns out that the second call to "b.first" will cause the spark driver to **allocate a VERY large piece of memory**. The first item of the first partition is very small (less than 200 bytes). However, the size of the entire first partition was about 380 MB. Apparently, my driver prepared itself to receive something large like an entire partition and allocated a chunk of appropriate size. Since I configured the driver to use only 256 MB, the allocation did not succeed and it ran out of memory.

Expert:
Job type: User-defined (Mailing list)
Causes: Driver collects large results (Expert)
Fix suggestions:no

7. something about rdd.collect

User:documents.flatMap(case words => words.map(w => (w, 1))).reduceByKey(_ + _).collect()
In driver's log, reduceByKey() is finished, but collect() seems always in run, just can't be finished.

In additional, there are about 200,000,000 words needs to be collected. Is it too large for collect()? But when words decreases to 1,000,000, it's okay!
Expert:collect essentially transfers all the data to the driver node. You definitely wouldn't want to collect 200 million words. It is a pretty large number and you can run out of memory on your driver with that much data.

Job type: User-defined (Mailing list)
Causes: Driver collects large results (User)
Fix suggestions:no

8. Driver OOM while using reduceByKey

User:I used 1g memory for the driver java process and got OOM error on driver side before reduceByKey. After analyzed the heap dump, the biggest object is org.apache.spark.MapStatus, which occupied over 900MB memory.
Expert: That hash map is just a list of where each task ran, it's not the actual data. How many map and reduce tasks do you have? Maybe you need to give the driver a bit more memory, or use fewer tasks (e.g. do reduceByKey(_ + _, 100) to use only 100 tasks).

Job type: User code (Mailing list)
Causes: Driver collects large results (Expert)
Fix suggestions:reduce task number

9. Beginner Question on driver memory issue (OOM).

User: Run the same action again. (OOM issue).

Observations

Running Linux top, **the driver process is definitely reaching the max memory usage**. My assumption was that calling collect on the RDD would only require high memory pressure while the collect function builds the result set and returns it. However it seems to be residing in memory as if its being cached or still hard-referenced somewhere.
Expert:

Job type: User code (Mailing list)
Causes: Driver collects large results (User)
Fix suggestions:no

10. Does foreach operation increase rdd lineage?

User: I'm writing a paralell mcmc program that having a very large dataset in memory, and need to update the dataset in-memory and avoid creating additional copy. Should I choose a foreach operation on rdd to express the change? or I have to create a new rdd after each sampling process?
Expert: Do you mean "Gibbs sampling" ? Actually, foreach is an action, it will collect all data from workers to driver. You will get OOM complained by JVM.

Job type: User code (Mailing list)
Causes: Driver collects large results (Expert)
Fix suggestions:no

11. RDD with a Map

User: for this to actually materialize I do collect

val groupedAndCollected=groupedWithValues.collect()

I get an Array[String,List[String]].

I am trying to figure out if there is a way for me to get Map[String,List[String]] (a multimap), or to create an RDD[Map[String,List[String]]]
Expert: At last, be careful if you are processing large volume of data, since groupByKey is an expensive transformation, and collecting all the data to driver side may simply cause OOM if the data can' t fit in the driver node.

Job type: User code (Mailing list)
Causes: Driver collects large results (Expert)
Fix suggestions:no

12. How to efficiently join this two complicated rdds

User: For each line in RDD one, we need to use the keys of the line to search the value according to the key in RDD of type two. And finally get the sum of these values. we have implement this way, we use pyspark, and standalone mode**. We collect the new RDD2 in each iteration**. The java heap memory costed by the driver program increases Gradually. And finally Collapse with OutOfMemory Error.

We have done some tests, in each iteration, we simply collect a vector. This Little Simple problem also costed more and more java heap memory, and finally raised OutOfMemory.

Expert: We have done some experiment using data which has much small size but same form. The method above will cost more than 10 mins while using collectAsMap function to collect RDD2 and sending it to each worker will cost 2 mins. But the second method will get outOfMemery error while we try the big data.
Job type: User code (Mailing list)
Causes: Driver collects large results (User)
Fix suggestions:no

13. java.lang.OutOfMemoryError while running SVD MLLib example

User:I am new to Spark. I have downloaded Spark 1.1.0 and trying to run the TallSkinnySVD.scala example with different input data sizes. I tried with input data with 1000X1000 matrix, 5000X5000 matrix.
Though I had faced some Java Heap issues I added following parameters in "spark-defaults.conf"

```
    spark.driver.memory         5g
    spark.executor.memory    6g
```

Now, I am trying with 7000X7000 input matrix, but it fails with OutofMemory error.
I tried by setting executor memoryto 8g but didn't worked.
I also tried by setting persist to MEMORY_AND_DISK level but no luck.

```
    rows.persist(StorageLevel.MEMORY_AND_DISK)
```

Expert:7000x7000 is not tall-and-skinny matrix. Storing the dense matrix requires 784MB. The driver needs more storage for collecting result from executors as well as making a copy for LAPACK's dgesvd. So you need more memory. Do you need the full SVD? If not, try to use a small k

Job type: MLlib (Mailing list)
Causes: Driver collects large results (User, reproduced)
Fix suggestions:use a small k (parameter)

14. Maximum size of vector that reduce can handle

User:I am trying to measure the Spark reduce performance for big vectors. My motivation is related to machine learning gradient. Gradient is a vector that is computed on each worker and then all results need to be summed up and broadcasted back to workers. For example, present machine learning applications involve very long parameter vectors, for deep neural networks it can be up to 2Billions.
"spark.driver.maxResultSize 0" needs to set in order to run this code. I also needed to change "java.io.tmpdir" and "spark.local.dir" folders because my /tmp folder which is default, was too small and Spark swaps heavily into this folder. Without these settings I get either "no space left on device" or "out of memory" exceptions.
Please try treeReduce instead which is what we do in linear regression and logistic regression.

Expert: When you use `reduce` to aggregate the vectors, those will actually be pulled into driver, and merged over there. Obviously, it's not scaleable given you are doing deep neural networks which have so many coefficients. For `reduce`, it's an action that will collect all the data from mapper to driver, and perform the aggregation in driver. As a result, if the output from the mapper is very large, and the numbers of partitions in mapper are large, it might cause a problem.

For `treeReduce`, as the name indicates, the way it works is in the first layer, it aggregates the output of the mappers two by two resulting half of the numbers of output. And then, we continuously do the aggregation layer by layer. The final aggregation will be done in driver but in this time, the numbers of data are small.

60m-vector costs 480MB memory. You have 12 of them to be reduced to the driver. So you need ~6GB memory not counting the temp vectors generated from '_+_'. You need to increase driver memory to make it work. That being said, ~10^7 hits the limit for the current impl of glm.

Job type: MLlib (Mailing list)
Causes: Driver collects large results (User)
Fix suggestions:tree reduce

15. take() reads every partition if the first one is empty

User: Wouldn't a better implementation strategy be

numPartsToTry = partsScanned * 2

instead of numPartsToTry = totalParts - 1
the logic for take() reads ALL partitions if the first one (or first k) are empty. **This has actually lead to OOMs when we had many partitions (thousands) and unfortunately the first one was empty**.

Expert:
Job type: User-defined (Mailing list)
Causes: Driver collects large results (User)
Fix suggestions:no

Suggestions

6. Q: spark java.lang.OutOfMemoryError: Java heap space

User:

First,I read some data(2.19G) from hdfs to RDD:

```
val imageBundleRDD = sc.newAPIHadoopFile(...)
```

Second,do something on this RDD:

```
val res = imageBundleRDD.map(data => {
            val desPoints = threeDReconstruction(data._2, bg)
             (data._1, desPoints)
            })
```

Expert:

- Try using more partitions, you should have 2 - 4 per CPU. IME increasing the number of partitions is often the easiest way to make a program more stable (and often faster). For huge amounts of data you may need way more than 4 per CPU, I've had to use 8000 partitions in some cases!
- Decrease the **fraction of memory reserved for caching**, using spark.storage.memoryFraction. If you don't use cache() or persist in your code, this might as well be 0. It's default is 0.6, which means you only get 0.4 * 4g memory for your heap. IME reducing the mem frac often makes OOMs go away.
- Similar to above but **shuffle memory fraction**. If your job doesn't perform a shuffle then set it to 0.0. Sometimes when it's a shuffle operation that's OOMing you need to do the opposite i.e. set it to something large, like 0.8, or make sure you allow your shuffles to spill to disk.
- Watch out for **memory leaks**, these are often caused by accidentally closing over objects you don't need in your lambdas. The way to diagnose is to look out for the "task serialized as XXX bytes" in the logs, if XXX is larger than a few k or more than an MB, you may have a memory leak.
- Realted to above; use **broadcast variables** if you really do need large objects.

Job type: User-defined (StackOverflow)

Fix suggestions: add partition number, lower cache space, lower buffer size, memory leaks (accidentally closing over objects you don't need in your lambdas),broadcast variables (if the objects are large)

Cases that their root causes are unknown

1. Q: OOM in spark pagerank

User:

When running graphX Page rank algorithm for 60 GB wiki data, the following error occurs. Please help.

The driver memory is 256m and executor memory is 6g. I tried increasing the driver memory as I am sorting the result and displaying first 100 pages.

Expert:

Pattern: Unknown (Broadcast related)

2. Q: In spark join, does table order matter like in pig?

User:

When doing a regular join in pig, the last table in the join is not brought into memory but streamed through instead, so if A has small cardinality per key and B large cardinality, it is significantly better to do `join A, B` than `join A by B`, from performance perspective (avoiding spill and OOM)

Expert:
It does not make a difference, in spark the RDD will only be brought into memory if it is cached. So in spark to achieve the same effect you can cache the smaller RDD. Another thing you can do in spark which I'm not sure that pig does, is if all RDD's being joined have the same partitioner no shuffle needs to be done.

Pattern: Information

3. A: How can I merge spark results files without repartition and copyMerge?

User/Expert:
Unfortunately, there is not other option to get a single output file in Spark. Instead of `repartition(1)` you can use `coalesce(1)`, but with parameter `1` their behavior would be the same. Spark would collect your data in a single partition in memory which might cause OOM error if your data is too big.

Pattern: repartition()

4. A: Cassandra - join two tables and save result to new table

User:
Once the data is uploaded, the users are allowed to build reports, analyze, etc., from within the application. I need a way to allow users to merge/join data from two or more datasets/tables based on matching keys and write the result into a new Cassandra table. Once a dataset/table is created, it will stay immutable and data is only read from it.

Expert:
The only option that you have is to do the join in your application code. There are just few details to suggest a proper solution.

Please add details about table keys, usage patterns... in general, in cassandra you model from usage point of view, i.e. starting with queries that you'll execute on data.

In order to merge 2 tables on this pattern, you have to do it into application, creating the third table (target table) and fill it with data from both tables. You have to make sure that you read the data in pages to not OOM, it really depends on size of the data.

Another alternative is to build the joins into Spark, but maybe is too over-engineering in your case.

Pattern: Join, too large generated intermediate results

5. A: Checking for equality of RDDs

User:
I making some tests in JUnit and I need to check the equality of two Spark RDDs.

Expert:
Once the data is uploaded, the users are allowed to build reports, analyze, etc., from within the application. I need a way to allow users to merge/join data from two or more datasets/tables based on matching keys and write the result into a new Cassandra table. Once a dataset/table is created, it will stay immutable and data is only read from it.

Pattern: collect()

6. Q: Spark groupBy OutOfMemory woes

User:

I'm doing a simple groupBy on a fairly small dataset (80 files in HDFS, few gigs in total). I'm running Spark on 8 low-memory machines in a yarn cluster, i.e. something along the lines of:

Expert:

http://apache-spark-user-list.1001560.n3.nabble.com/Understanding-RDD-GroupBy-OutOfMemory-Exceptions-td11427.html#a11487

Patrick Wendell shed some light on the details of the groupBy operator on the mailing list. The takeaway message is the following:

Within a partition things will spill [...] This spilling can only occur *across keys* at the moment. Spilling cannot occur within a key at present. [...] Spilling within one key for GroupBy's is likely to end up in the next release of Spark, Spark 1.2. [...] If the goal is literally to just write out to disk all the values associated with each group, and the values associated with a single group are larger than fit in memory, this cannot be accomplished right now with the groupBy operator.

He further suggests a work-around:

The best way to work around this depends a bit on what you are trying to do with the data down stream. Typically approaches involve sub-dividing any very large groups, for instance, appending a hashed value in a small range (1-10) to large keys. Then your downstream code has to deal with aggregating partial values for each group. If your goal is just to lay each group out sequentially on disk on one big file, you can call `sortByKey` with a hashed suffix as well. The sort functions are externalized in Spark 1.1 (which is in pre-release).

Pattern: Large group, large data partitions

7. Q: Apache Spark on YARN: Large number of input data files (combine multiple input files in spark)

User:

Expert:

I'm pretty sure the reason your getting OOM is because of handling so many small files. What you want is to combine the input files so you don't get so many partitions. I try to limit my jobs to about 10k partitions.

After `textFile`, you can use `.coalesce(10000, false)` ... not 100% sure that will work though because it's been a while since I've done it, please let me know. So try

```
sc.textFile(path).coalesce(10000, false)
```

It worked! Actually I used coalesce factor 1227, which is the number of partitions when Spark process the big single file that contains the whole records. But the job runs slower(as expected), and still it seems the information of all files is still transferred to the driver process, which can cause OOM when too many files are involved. But 1.68GB for the driver process for 168016 files are not so bad.

Pattern: partition-related

8. Q: How to distribute data to worker nodes

User:

The computation then ends with 'OutOfMemory' exception on nodes. When I parallelize to more partitions (e.g. 600 partitions - to get the 100kB per task). The computations are performed successfully on workers but the 'OutOfMemory' exceptions is raised after some time in the driver. This case, I can open spark UI and observe how te memory of driver is

slowly consumed during the computation. It looks like the driver holds everything in memory and doesn't store the intermediate results on disk.
Expert:
Hard to say without looking at the code. Most operations will spill to disk. If I had to guess, I'd say you are using `groupByKey` ?

Pattern: Unknown

9. Q: Configure Java heap space with Spark

I'm trying to create a file with few hundred mega bytes by oversampling a small array in spark and save as object file to hdfs system created by spark-ec2 script:

```
//Oversampling repNum LabeledPoints from the array above val overSample = labelPts.takeSample(true, repNum, 1)
```

Then it throws a `EXCEPTION: java.lang.OutOfMemoryError: Java heap space`. I don't know what's wrong with it because if my `repNum` is set to 6000000, there will be no error and the output file is around 490m, so I suspect that the java heap space is still capped by 512m, however the I've set `--executor-memory=4g` and the worknode in this cluster has 7.5GB memory. What's the problem here?

Pattern: Unknown

10. Q: Calculate eccentricity of 5 vertices : Java heap space exeption

User :
 have directed graph. Text file contains 5 million edges in format: sourceVertexId targetVertexId

and it size is approximately 54 Gb.

I want to load this graph using GraphLoader.edgeListFile and then using algorithm from this tutorial: https://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel-api, I want to find eccentricity of 5 vertices.

Expert: None
Pattern: Unknown

11. Q: Error when trying to run algorithm in Spark

User:

```
hafidz@localhost dga]$ /opt/dga/dga-mr1-graphx pr -i sna_exp_comma.csv -o pr_sna.txt -n testPageRank -m
spark://localhost.localdomain:7077 --S spark.executor.memory=1g  --S spark.worker.timeout=400  --S
spark.driver.memory=1g
```

Expert:
Pattern: Unknown

12. Q: Estimating required memory for Scala Spark job

User:
I'm logging the computations and after approx 1'000'000 calculations I receive above exception.

The number of calculations required to finish job is 64'000'000

Currently I'm using 2GB of memory so does this mean to run this job in memory without any further code changes will require 2GB * 64 = 128GB or is this a much too simpistic method of anticipating required memory ?

Expert:
Loading an RDD as a Broadcast variable means being able to load the entire RDD in each node. Whereas partitionning splits the file into chunks, each node processing some chunks. The value "61983+2066" is determined by the number of partitions and the file size: there is a `minPartitions` argument in the`.textFile` method.

Pattern: Unknown

13. Q: spark mllib memory error on svd (single machine)
User:
When I try to compute the principal components I get a memory error:

Expert:

Pattern: Unknown

14. Q: Error when running Spark on a google cloud instance
User:
I'm running a standalone application using Apache Spark and when I load all my data to a RDD as a textfile I got the following error:

Expert:

Pattern: Unknown

15. Q: Apache Spark - MLlib - K-Means Input format
User :
I want to perform a K-Means task and fail training the model and get kicked out of Sparks scala shell before I get my result metrics. I am not sure if the input format is the problem or something else. I use Spark 1.0.0 and my input textile (400MB) looks like this:

Expert:

Pattern: Unknown

16. Q: Spark throwing Out of Memory error
User:
When I try to load and process all data in the Cassandra table using spark context object, I'm getting an error during processing. So, I'm trying to use a looping mechanism to read chunks of data at a time from one table, process them and put them in another table.

Expert:
You are running a query inside the for loop. If the 'value' column is not a key/indexed column, Spark will load the table into memory and then filter on the value. This will certainly cause an OOM.

Pattern: Unknown

17. Q: Spark PCA OutOfMemory error on small number of columns and rows

User:

I am attempting to perform Spark MLLib PCA (using Scala) on a RowMatrix with 2168 columns, and a large number of rows. However, I have observed that even with as few as 2 rows in the matrix (a 112KB text file), the following error is always produced, at the same job step:

Pattern: Unknown

18. Q: Spark OutOfMemory error on small text data

User:

I am working on implementing an algorithm and testing it on medium-sized data in Spark (the Scala interface) on a local node. I am starting with very simple processing and I'm getting `java.lang.OutOfMemoryError: Java heap space` even though I'm pretty sure the data isn't big enough for such an error to be reasonable. Here is the minimal breaking code:

Expert:

So, thanks to all those small strings, your data in memory is roughly 5x the size 'at rest'. Still, 200k lines of that data makes up for roughly 500MB. This might indicate that your executor is operating at the default valie of 512MB. Try setting 'spark.executor.memory' to a higher value, but also consider a heap size >8Gb to confortably work with Spark.

Pattern: Unknown

19. Q: Monitor different blocks of memory usage in Spark and know what is running out on OOM?

User:

Apache Spark has 3 blocks of memory:

- Cache - this is where RDDs are put when you call `cache` or `persist`
- Shuffle. This is the block of memory used for shuffle operations (grouping, repartitioning, and `reduceByKey`.
- Heap. This is where normal JVM objects are kept.

Now I would like to monitor the amount of memory in use as a % of each block by a job so I can know what I should be tuning these numbers to so that Cache and Shuffle do not spill to disk and so that Heap doesn't OOM. E.g. every few seconds I get an update like:

```
Cache: 40% use (40/100 GB)
Shuffle: 90% use (45/50 GB)
Heap: 10% use (1/10 GB)
```

I am aware I can experiment to find the sweet spots using other techniques, but I'm finding this very laboured and to just be able to monitor the usage would make writing and tuning Spark jobs much much easier.

Expert:

Pattern: Information

20. Q: Out of memory exception during TFIDF generation for use in Spark's MLlib

User:

https://chimpler.wordpress.com/2014/06/11/classifiying-documents-using-naive-bayes-on-apache-spark-mllib/

Memory overflow and GC issues occur while collecting idfs for all the terms. To give an idea of scale, I am reading around 615,000(around 4GB of text data) small sized documents from HBase and running the spark program with 8 cores and 6GB of executor memory. I have tried increasing the parallelism level and shuffle memory fraction but to no avail.
Expert:
Pattern: Unknown

21. Q: OutOfMemoryError while Logistic regression in SparkR

User:
I have successfully installed Apache Spark, Hadoop over Ubuntu 12.04 (Single standalone mode) for Logistic regression. Also tested with small csv dataset but it doesnt work over large dataset having 269369 rows.

Expert:
As a back-of-the-envelope calculation, assuming each entry in your dataset takes 4 bytes, the whole file in memory would cost `269369 * 541 * 4 bytes ~= 560MB`
Pattern: Unknown

22. Q: Spark - convert string IDs to unique integer IDs

User:
When I try to run this on my cluster (40 executors with 5 GB RAM each), it's able to produce the idx1map and idx2map files fine, but it fails with out of memory errors and fetch failures at the first flatMap after cogroup.

The reason I'm not just using a hashing function is that I'd eventually like to run this on a much larger dataset (on the order of 1 billion products, 1 billion users, 35 billion associations), and number of Int key collisions would become quite large.

Expert:
I looks like you are essentially collecting all lists of users, just to split them up again. Try just using join instead of cogroup, which seems to me to do more like what you want.

23. Q: Running a function against every item in collection

User:
I have tried

```
val dataArray = counted.collect
dataArray.flatMap { x => dataArray.map { y => ((x._1+","+y._1),func) } }
```

which converts the collection to Array type and applies same function. But I run out of memory when I try this method. I think using an RDD is more efficient than using an Array ?

The collection I'm running this function on contain 20'000 entries so 20'000^2 comparisons (400'000'000) but in Spark terms this is quite small?

Expert:
using cartesian seems to be lazy as when I use it it returns straight away but running any functions on the collection generated by cartesian is still struggling with memory

Pattern: collect(), Cartesian()

24. A: Garbage Collection of RDDs

User:
I have a fundamental question in spark. Spark maintains lineage of RDDs to recalculate in case few RDDs get corrupted. So JVM cannot find it as orphan objects. Then how and when the garbage collection of RDDs happen?

Expert:
The memory for RDD storage can be configured using

`"spark.storage.memoryFracion"` property.
If this limit exceeded, older partitions will be dropped from memory.

Pattern: Large data cached in memory

25. Q: Mapping an RDD of value to a cartesian product and grouping by value
User:
Option 1 seems like the natural choice, but what I'm finding is that even for very small sets, e.g., ~500 elements, with each element for example a list of one hundred Doubles, the reduceByKey (or groupBy, which I've also tried) maps to 40000 ShuffleMapTasks that complete at a rate of about 10 per second. After about 30 minutes, when approx. 1/4 are done, the job fails with a GC out of memory error. Is there a way to ensure that the cartesian product preserves partitions? Is there a more efficient way to handle the reduce task? I've also tried different keys (e.g., Ints), but there was no improvement.

The larger context of this particular problem, as I have detailed in an edit, is to do a pairwise computation over each pair of vectors in the collection. In general, though, I need to do other groupBy operations that are not simple aggregations where the number of keys is on the same order of magnitude as the number of records.

Expert:
Pattern: Cartesian() + groupBy() (Further study)

26. Q: I am getting the executor running beyond memory limits when running big join in spark
User:
I am getting the following error in the driver of a big join on spark.

We have 3 nodes with 32GB of ram and total input size of join is 150GB. (The same app is running properly when input file size is 50GB)

I have set storage.memoryFraction to 0.2 and shuffle.memoryFraction to 0.2. But still keep on getting the running beyong physical limits error

Pattern: Big join (Unknown)

27. Running out of memory Naive Bayes

28. Kafka streaming out of memory

29. Serializer or Out-of-Memory issues?

30. OUT OF MEMORY ERROR: HEAP SPACE

31. Out of Memory - Spark Job server

32. Driver fail with out of memory exception

33. Running out of memory on livejournal

34.MLlib Logistic Regression run out of memory

35.Executor out of memory and gets killed

36. out of memory errors -- per core memory limits?

37.help me: Out of memory when spark streaming

38.Out of memory exception in MLlib's naive baye's classification training

39.Meaning of persistence levels -- setting persistence causing out of memory errors with pyspark

40.Spark SQL 1.1.0 - large insert into parquet runs out of memory

41.Long running time for GraphX pagerank in dataset com-Friendster

42.Lost executors

43.how to debug ExecutorLostFailure

44.OutOfMemory of spark streaming on standalone in 5 minutes, but run normal on local mode

45.[SQL] Set Parquet block size?

46.trouble with broadcast variables on pyspark

47.RDD join: composite keys

48.AppMaster OOME on YARN

49.JVM Memory Woes

50.GC issues

51.Pyspark Memory Woes

52."Spilling in-memory..." messages in log even with MEMORY_ONLY

53.Question regarding spark data partition and coalesce. Need info on my use case.

54.driver memory management

55.something about rdd.collect

56.memory leak query

57.OOM with groupBy + saveAsTextFile

58.trouble with "join" on large RDDs

59.Spark app throwing java.lang.OutOfMemoryError: GC overhead limit exceeded

60.java.lang.OutOfMemoryError: Requested array size exceeds VM limit

61.shuffle memory requirements

62.OutofMemoryError when generating output

63.Getting java.lang.OutOfMemoryError when calling mkString on a parition

64. newbie : java.lang.OutOfMemoryError: Java heap space

65.extremely slow k-means version

66.[Streaming]Executor OOM

67.OOM for HiveFromSpark example

68.OOM writing out sorted RDD

69.spark master OOME from maxMbInFlight buffers

70.Spark on Mesos cause mesos-master OOM

71.Beginner Question on driver memory issue (OOM).

72.Spark-Shell: OOM: GC overhead limit exceeded

73.OOM Java heap space error on saveAsTextFile

74.pyspark sc.parallelize running OOM with smallish data

75.[PySpark] large # of partitions causes OOM

76.OOM - Requested array size exceeds VM limit

77.OOM when calling cache on RDD with big data

78.Running Wordcount on large file stucks and throws OOM exception

79.What the initial steps to understand root of this OOM exception

80.how to set spark.executor.memory and heap size

81.How to correctly extimate the number of partition of a graph in GraphX

82.GroupBy Key and then sort values with the group

83.How to compute RDD[(String, Set[String])] that include large Set

84.Problem when sorting big file

85. Questions about productionizing spark

86. ExternalAppendOnlyMap: Spilling in-memory map

87.JVM heap and native allocation questions

88.Spark Bug? job fails to run when given options on spark-submit (but starts and fails without)

89.news20-binary classification with LogisticRegressionWithSGD

90.Use Spark streaming to process events in a stream and maintain aggregate statisitics on it

91.com.google.protobuf out of memory

92.Spark Memory Bounds

93.GC Issues with randomSplit on large dataset

94.Spark SQL Exception

95.OutOfMemory Error

96.OutOfMemory Error

97.Cannot get rid of OutOfMemory

98.OutOfMemory error in Spark Core

99. OutOfMemory : Java heap space error

100.OutofMemory: Failed on spark/examples/bagel/WikipediaPageRank.scala

101.mllib svd gives: Java OutOfMemory Error

102.Scala PCA OutOfMemory error on small number of columns

103.Memory config issues

104.OOM when making bins in BinaryClassificationMetrics ?

105.Too big data Spark SQL on Hive table on version 1.0.2 has some strange output

106.Re: Out of memory on large RDDs