An Empirical Study of Out of Memory Errors in Apache Spark

1

[Q: Spark out of memory](http://stackoverflow.com/questions/24570532/spark-out-of-memory?s=1|4.3320)

**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.

**Expert:** 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.

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)

**Pattern:** Large input partition read into memory

1

[Q: spark java.lang.OutOfMemoryError: Java heap space](http://stackoverflow.com/questions/21138751/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.

**Pattern: Unknown**

1

[Q: Configure Java heap space with Spark](http://stackoverflow.com/questions/28469084/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: Further study

1

[Q: Calculate eccentricity of 5 vertices : Java heap space exeption](http://stackoverflow.com/questions/26488255/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**

1

[Q: Spark mllib svd gives: Java OutOfMemory Error](http://stackoverflow.com/questions/28974037/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:

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?

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?

Reason:

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.

**Source code: Stack Trace**

1

[Q: Spark raises OutOfMemoryError](http://stackoverflow.com/questions/23601607/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 force 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.

Pattern: Collect large data in Driver

1

[Q: Error when trying to run algorithm in Spark](http://stackoverflow.com/questions/27964130/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 (Probably related to broadcast)**

1

[Q: Spark OutOfMemoryError when adding executors](http://stackoverflow.com/questions/26466866/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.

**Pattern: Collect large data in driver (gradients)**

1

[Q: Estimating required memory for Scala Spark job](http://stackoverflow.com/questions/28479483/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**

1

[Q: spark mllib memory error on svd (single machine)](http://stackoverflow.com/questions/29348301/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**

1

[Q: Error when running Spark on a google cloud instance](http://stackoverflow.com/questions/28773955/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**

1

[Q: Apache Spark - MLlib - K-Means Input format](http://stackoverflow.com/questions/25185278/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**

1

[Q: Memory issues when running Spark job on relatively large input](http://stackoverflow.com/questions/26509682/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,

**Pattern: Large intermediate results (large buffer + large accumulated results) (Reproduced)**

1

[Q: spark aggregatebykey with collection as zerovalue](http://stackoverflow.com/questions/29011117/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:

**Source code: large accumulated results (OOM caused by aggregateByKey()) (Further study)**

1

[Q: OutOfMemoryError while Logistic regression in SparkR](http://stackoverflow.com/questions/26225001/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: Large intermediate results (Large matrix generated in the code) (Further study only related to the current record?)**

1

[Q: GraphX does not work with relatively big graphs](http://stackoverflow.com/questions/29050904/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:

**Pattern: Probably the reason of collect(), since large vertices and edges are collected in the driver.**

1

[Q: Spark: out of memory exception caused by groupbykey operation](http://stackoverflow.com/questions/25709465/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

**Pattern: improper data partition**

1

[Q: Spark throwing Out of Memory error](http://stackoverflow.com/questions/25404263/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**

1

[Q: Spark: Out Of Memory Error when I save to HDFS](http://stackoverflow.com/questions/29540257/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.

**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.

**Pattern: Serialize large object (large intermediate data)**

1

[Q: Spark runs out of memory when grouping by key](http://stackoverflow.com/questions/22637518/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](http://spark-summit.org/2013/exercises/launching-a-bdas-cluster-on-ec2.html), 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](https://spark.apache.org/docs/latest/tuning.html#level-of-parallelism).

**Pattern: Improper data partition**

1

[Q: Monitor different blocks of memory usage in Spark and know what is running out on OOM?](http://stackoverflow.com/questions/24189822/monitor-different-blocks-of-memory-usage-in-spark-and-know-what-is-running-out-o)

**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**

1

[Q: Out of memory exception during TFIDF generation for use in Spark's MLlib](http://stackoverflow.com/questions/25910642/out-of-memory-exception-during-tfidf-generation-for-use-in-sparks-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**

1

[Q: spark executor lost failure](http://stackoverflow.com/questions/29566522/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:**

**Pattern: Unknown (Further study)**

1

[Q: Memory efficient way of union a sequence of RDDs from Files in Apache Spark](http://stackoverflow.com/questions/28343181/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:**

o 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!

**Pattern: probably collect() in driver**

1

[Q: Regarding spark input data partition and coalesce](http://stackoverflow.com/questions/25336181/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.

**Pattern: Improper data partition**

1

[Q: Why does Spark RDD partition has 2GB limit for HDFS](http://stackoverflow.com/questions/29689719/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](https://issues.apache.org/jira/browse/SPARK-1476) 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.

**Pattern: partition and flatMap() can generate large intermediate results**

1

[Q: Spark - convert string IDs to unique integer IDs](http://stackoverflow.com/questions/28097333/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.

**Pattern: OOM in Cogroup()**

1

[Q: How to filter a RDD according to a function based another RDD in Spark?](http://stackoverflow.com/questions/26035582/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:

**Source code: Large data collect() at driver (using weights.toArray.toMap)**

1

[Q: Running a function against every item in collection](http://stackoverflow.com/questions/23574379/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()**

1

[A: How to use spark to generate huge amount of random integers?](http://stackoverflow.com/questions/29069080/how-to-use-spark-to-generate-huge-amount-of-random-integers/29070197#29070197)

**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.

**Pattern: large data generated at driver**

1

[A: org.apache.spark.shuffle.MetadataFetchFailedException: Missing an output location for shuffle 0](http://stackoverflow.com/questions/28901123/org-apache-spark-shuffle-metadatafetchfailedexception-missing-an-output-locatio/28964213#28964213)

**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.

**Pattern: repartitionAndSortWithinPartition(), large framework buffer**

1

[A: How to iterate over large Cassandra table in small chunks in Spark](http://stackoverflow.com/questions/28726044/how-to-iterate-over-large-cassandra-table-in-small-chunks-in-spark/28728332#28728332)

**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 thecollect action only if you know for sure it will produce a small number of rows.

**Source code: collect()**

1

[A: Garbage Collection of RDDs](http://stackoverflow.com/questions/29057461/garbage-collection-of-rdds/29058160#29058160)

**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 framework buffer**

1

[Q: Mapping an RDD of value to a cartesian product and grouping by value](http://stackoverflow.com/questions/25799746/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)**

1

[Q: I am getting the executor running beyond memory limits when running big join in spark](http://stackoverflow.com/questions/29494411/i-am-getting-the-executor-running-beyond-memory-limits-when-running-big-join-in)

**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)**

1

[A: Spark Java Error: Size exceeds Integer.MAX\_VALUE](http://stackoverflow.com/questions/28967111/spark-java-error-size-exceeds-integer-max-value/28974721#28974721)

**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.

**Pattern: partition related**

1

[Q: OOM in spark pagerank](http://stackoverflow.com/questions/28552865/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)**

1

[Q: In spark join, does table order matter like in pig?](http://stackoverflow.com/questions/28694523/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**

1

[A: How can I merge spark results files without repartition and copyMerge?](http://stackoverflow.com/questions/29025147/how-can-i-merge-spark-results-files-without-repartition-and-copymerge/29025398#29025398)

**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()**

1

[A: Cassandra - join two tables and save result to new table](http://stackoverflow.com/questions/29166566/cassandra-join-two-tables-and-save-result-to-new-table/29166684#29166684)

**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**

1

[A: Checking for equality of RDDs](http://stackoverflow.com/questions/27213742/checking-for-equality-of-rdds/27217784#27217784)

**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()**

1

[Q: Spark OutOfMemory error on small text data](http://stackoverflow.com/questions/24394637/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 (Further study)**

1

[Q: Spark groupBy OutOfMemory woes](http://stackoverflow.com/questions/25136064/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](http://apache-spark-user-list.1001560.n3.nabble.com/Understanding-RDD-GroupBy-OutOfMemory-Exceptions-td11427.html#a11487). 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**

1

[Q: Spark PCA OutOfMemory error on small number of columns and rows](http://stackoverflow.com/questions/29684406/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**

1

[Q: Apache Spark on YARN: Large number of input data files (combine multiple input files in spark)](http://stackoverflow.com/questions/24623402/apache-spark-on-yarn-large-number-of-input-data-files-combine-multiple-input-f)

**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**

1

[Q: How to distribute data to worker nodes](http://stackoverflow.com/questions/27546157/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**