## How to handle out of memory issues?

Out of memory can be solved in different ways.

Please try the following one by one;

* If not yet using Glue 3.0, please enable that. See [How to: Set Glue version](file:///C:\display\DP\How+to:+Set+Glue+version)
  + **IMPORTANT NOTES:** Glue 4.0 is using a different Spark version (Spark 3.3) compared to Glue 3.0 (Spark 3.1). Apache Spark has a nice feature called [Adaptive Query Execution (AQE)](https://spark.apache.org/docs/latest/sql-performance-tuning.html#adaptive-query-execution), which performs optimizations based on runtime statistics and is enabled by default since 3.2.
  + The latter improves query performance most of the time, but may have an opposite effect on long running compute intensive tasks which may cause Out of Memory issues.
* Optimise Spark specifically for your job. See [How to: Configure Spark settings](file:///C:\display\DP\How+to:+Configure+Spark+settings)
* Increase the number of workers. See [publication-details](https://globalitconfluence.us.aegon.com/display/DP/publication-details" \l "expand-options)

## SparkOutOfMemoryError

(List of OutOfMemory issues in Spark: <https://github.com/JerryLead/MyNotes/blob/master/Grind/OOM-Cases-Spark-User-Nabble.md>)

### Example SparkOutOfMemoryError: Unable to acquire 65536 bytes of memory, got 0

The cause of the error is a memory leak in the SortMergeJoin operation. The issue is fixed in spark 3.0. More info: <https://issues.apache.org/jira/browse/SPARK-24657>.

The suggested workaround for spark 2.4 is to broadcast the table you want to join first. In HIVESQL you should use the MAPJOIN operation to achieve this. MAPJOIN uses a hint, which looks like:

/\*+ MAPJOIN(aliasname), MAPJOIN(anothertable) \*/.

This C-style comment should be placed immediately following the SELECT. It directs Hive to load aliasname (which is a table or alias of the query) into memory. see also: <https://grisha.org/blog/2013/04/19/mapjoin-a-simple-way-to-speed-up-your-hive-queries/>.

We suggest to try this first in the Query Interface in a datalab. We can't help you pinpoint for which table exactly this memory leak problem occurs. Another workaround is to look critically to your joins and cancel out joins that are not needed or making joins explicit over implicit

## Optimize Memory Management

Apache Spark provides several knobs to control how memory is [managed](https://spark.apache.org/docs/latest/tuning.html#memory-management-overview)for different workloads.

However, this is not an exact science and applications may still run into a variety of [out of memory (OOM) exceptions](https://github.com/JerryLead/MyNotes/blob/master/Grind/OOM-Cases-Spark-User-Nabble.md)because of inefficient transformation logic.

**Optimize Spark queries**: Inefficient queries or transformations can have a significant impact on Apache Spark driver memory utilization. Common examples include:

* **collect** is a Spark action that collects the results from workers and return them back to the driver. In some cases the results may be very large overwhelming the driver.

It is recommended to be careful while using collect as it can frequently cause Spark driver OOM exceptions as shown below:

An error occurred while calling

z:org.apache.spark.api.python.PythonRDD.collectAndServe.

Job aborted due to stage failure:

Total size of serialized results of tasks is bigger than spark.driver.maxResultSize

**Join Optimizations**: One common reason for Apache Spark applications running out of memory is the use of un-optimized joins across two or more tables. This is typically a result of data skew due to the distribution of join columns or an inefficient choice of [join transforms](https://spark.apache.org/docs/2.4.3/rdd-programming-guide.html). Additionally, ordering of transforms and filters in the user script may limit the Spark query planner’s ability to optimize. There are 3 popular approaches to optimize join’s on AWS Glue.

* **Filter tables before Join:**You should pre-filter your tables as much as possible before joining. This helps to minimize the data shuffled between the executors over the network. You can use AWS Glue [push down predicates](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-partitions.html) for filtering based on partition columns, AWS Glue[exclusions](https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-connect.html) for filtering based on file names, AWS Glue storage class exclusions for filtering based on S3 storage classes, and use columnar storage formats such as Parquet and ORC that support discarding row groups based on column statistics such as min/max of column values.
* **Broadcast Small Tables:**Joining tables can result in large amounts of data being shuffled or moved over the network between executors running on different workers. Because of this, Spark may run out of memory and spill the data to physical disk on the worker.

Above information retrieved from <https://aws.amazon.com/blogs/big-data/optimize-memory-management-in-aws-glue/>