1 Questions

- **Tasks/cores/executors**
  - in databricks, can we specify only 2 (for instance) executors for a node with 8 cpus?
  - if so, are these executors/jvms aware that the node has more cpus available? Are they accessible Use case - want to run some non-spark multithreaded code, e.g. distribute 1 xgboost job to each node, with each job using c cores
  - Ali said that he turned off autoscaling and had one executor per node. (8 cores per node)
- **joining** We have (say) a small list of collector ids, and want to left join with 5-6 large tables (millions of collector ids, thousands of columns) What is the best (optimal) way to do this? More shuffle partitions are better? more data partitions?

2 Joining

2.1 links

- `https://databricks.com/session/optimizing-apache-spark-sql-joins`
  30 minute video, good overview
- `https://sujithjay.com/spark-sql/2018/02/17/Broadcast-Hash-Joins-in-Apache-Spark/`
  part one of a series

2.2 Shuffle Hash join

Pretty (most?) common type of join. Data from table 1 and table 2 are partitioned by key. Matching partitions are sent to the same executor, then the output is merged. This works best when

- data is evenly distributed by key
- good number of keys (unique values - monthid would be a bad key (12 unique values))

When these conditions can't be satisfied, then broadcast hash join might be suitable

Diagnosing problems

- Tasks that take a long time - could be uneven sharding (keys not distributed evenly).

- Speculative tasks - if something takes a long time on one exdecutor, spark might send think theres a problem there and send the task to another to see if ti works there. If one particular task spawns speculative tasks, this can also be a sign that there is a sharding problem.
- Shuffle read/write memory - take a look to see if there is one task sending/receiving a lot of data (more than average)
- ls across the output files - see if one data partition is much larger than others

2.3 broadcast hash join

Small table sent to every executor, each partition is joined with the entire table, then output is returned. Bestest performance. Spark will automatically broadcast tables that are smaller than `spark.sql.autoBroadcastJoinThreshold`, which is 10 MB by default. Giving hints can override this behaviour, but if the broadcasted table is too big you may OOM your cluster. For right outer join, Spark can only broadcast the left side. For left outer, left semi, left anti and the internal join type ExistenceJoin, Spark can only broadcast the right side. I have had cases where I want to join a small list of collectors, tableA, (one column of collector keys) to a bigger table (millions of rows, thousands of columns). Normally I would do `c = tableA.join(tableB,['COLLECTOR_KEY'],how='left')`. To use broadcasting, the join has to be righted: `c = tableB.join(broadcast(tableA),['COLLECTOR_KEY'],h`

`df.explain()` on the output will tell whether a shuffle hash or broadcast hash join is being made (spark can sometimes have a hard time figuring out when to use a broadcast join with stuff in hive. giving a hint might be handy).

2.4 Cartesian join (cross join)

- create an rdd of uid by uid pairs
- broadcast that
- call a udf that retrieves data from the big tables given the uid,uid pairs

2.5 one to many join (one sided cartesian)

- number of rows can explode not a big del with parquet - duplicate data encodes well (for output files)

2.6 theta join

join tableA with TableB on (keyA ¡ keyB +10) - Spark will treat this as a cartesian join - Much better to bucket A and B, and create an intial paritioning based on bucket equality