Hive on Tez Performance Tuning - Determining Reducer Counts

How Does Tez determine the number of reducers? How can I control this for performance?

In this article, I will attempt to answer this while executing and tuning an actual query to illustrate the concepts. Then I will provide a summary with a full explanation. if you wish, you can advance ahead to the summary.

----

0. Prep Work and Checklist

We followed the Tez Memory Tuning steps as outlined in
https://community.hortonworks.com/content/kbentry/14309/demystify-tez-tuning-step-by-step.html
We setup our environment, turning CBO and Vectorization On.
set hive.support.sql11.reserved.keywords=false;

set hive.execution.engine=tez; set hive.cbo.enable=true; set hive.compute.query.using.stats=true; set hive.stats.fetch.column.stats=true; set hive.stats.fetch.partition.stats=true; set hive.vectorized.execution.enabled=true; set hive.vectorized.execution.reduce.enabled = true; set hive.vectorized.execution.reduce.groupby.enabled = true;

set hive.exec.parallel=true; set hive.exec.parallel.thread.number=16; We create Orc tables and did an Insert Overwrite into Table with Partitionsset hive.exec.dynamic.partition.mode=nonstrict;

There is a danger with many partition columns to generate many broken files in ORC. To prevent that

gt; set hive.optimize.sort.dynamic.partition=true;

if hive jobs previously ran much faster than in the current released

ersion, look into potentially setting property > hive.optimize.sort.dynamic.partition = false .

> insert overwrite table benchmark_rawlogs_orc partition (p_silo,p_day,p_clienthash) select * FROM <original table>; We generated the statistics we needed for use in the Query Execution-- // generate statistics for the ORC table

set hive.stats.autogather=true;-- // To Generate Statistics for Entire Table and Columns for All Days (Longer)

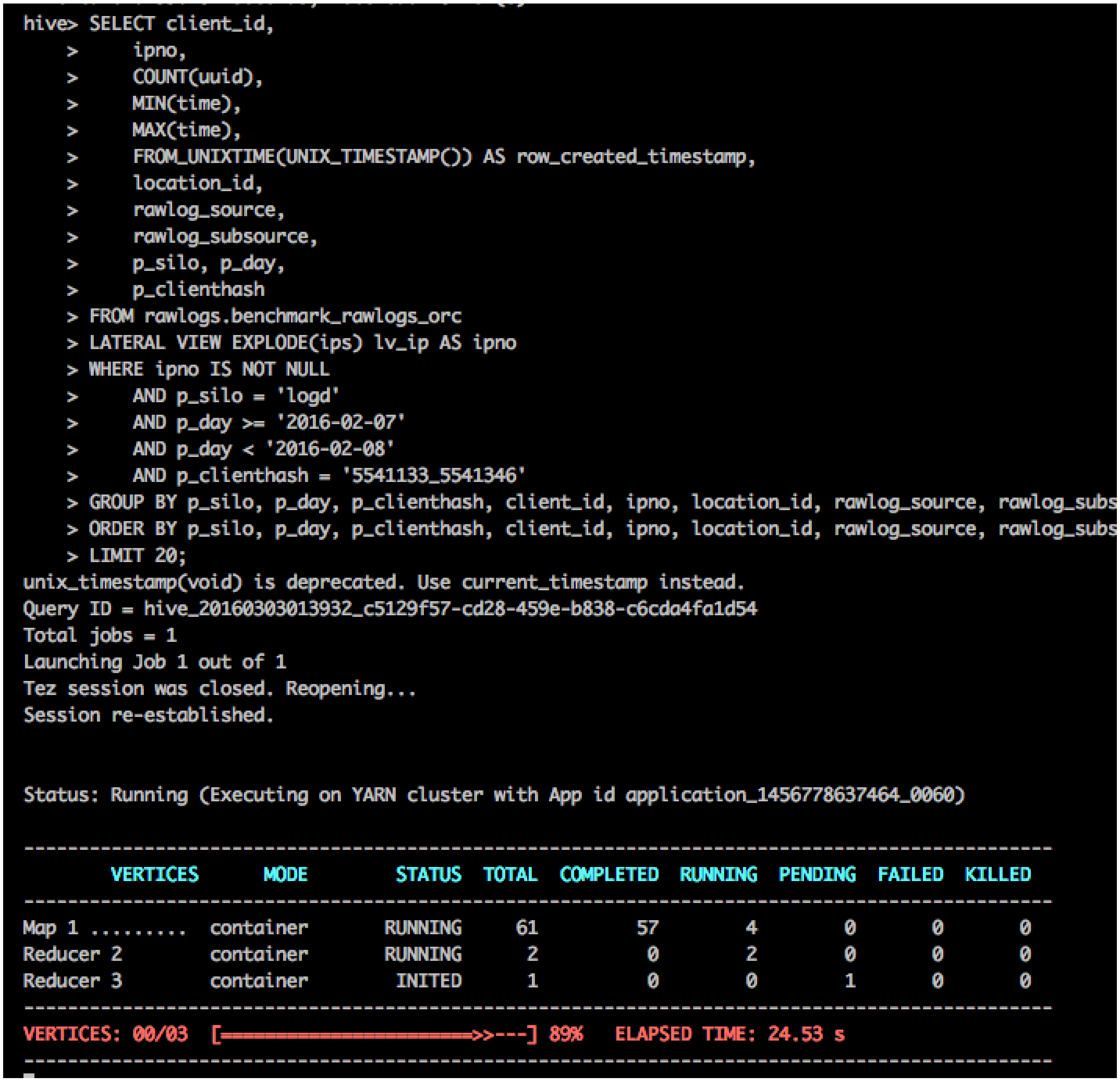
ANALYZE TABLE rawlogs.benchmark_rawlogs_orc partition (p_silo, p_day, p_clienthash) COMPUTE

STATISTICS;

ANALYZE TABLE rawlogs.benchmark_rawlogs_orc partition (p_silo, p_day, p_clienthash) COMPUTE STATISTICS for columns;

----

1. First Execution of Query



```
hive> SELECT client_id,
    >     ipno,
    >     COUNT(uuid),
    >     MIN(time),
    >     MAX(time),
    >     FROM_UNIXTIME(UNIX_TIMESTAMP()) AS row_created_timestamp,
    >     location_id,
    >     rawlog_source,
    >     rawlog_subsource,
    >     p_silo, p_day,
    >     p_clienthash
    > FROM rawlogs.benchmark_rawlogs_orc
    > LATERAL VIEW EXPLODE(ips) lv_ip AS ipno
    > WHERE ipno IS NOT NULL
    >     AND p_silo = 'logd'
    >     AND p_day >= '2016-02-07'
    >     AND p_day < '2016-02-08'
    >     AND p_clienthash = '5541133_5541346'
    > GROUP BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subs
    > ORDER BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subs
    > LIMIT 20;
unix_timestamp(void) is deprecated. Use current_timestamp instead.
Query ID = hive_20160303013932_c5129f57-cd28-459e-b838-c6cda4fa1d54
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1456778637464_0060)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .........   container     RUNNING     61         57        4        0       0       0
Reducer 2         container     RUNNING      2          0        2        0       0       0
Reducer 3         container      INITED      1          0        0        1       0       0
--------------------------------------------------------------------------------
VERTICES: 00/03  [=======================>>---] 89%   ELAPSED TIME: 24.53 s
--------------------------------------------------------------------------------
```

Here we can see 61 Mappers were created, which is determined by the group splits and if not grouped, most likely corresponding to number of files or split sizes in the Orc table. For a discussion on the number of mappers determined by Tez see How are Mappers Determined For a Query and How initial task parallelism works

The mappers complete quickly but the the execution is stuck on 89% for a long time.

We observe that there are three vertices in this run, one Mapper stage and two reducer stages.

The first reducer stage ONLY has two reducers that have been running forever? hmmmm...

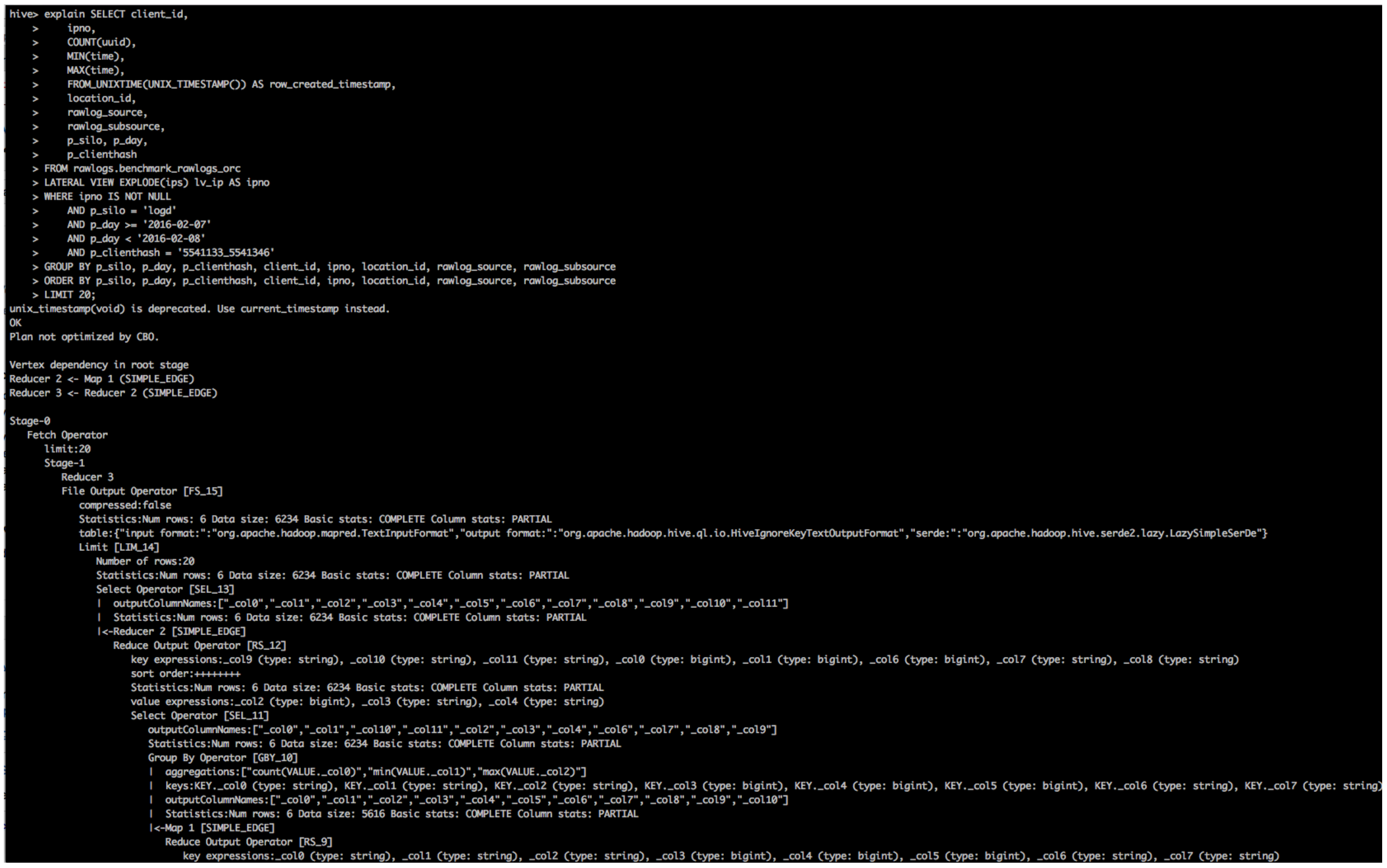
Query finally completed in 60 secs.

What gives? Why only 2 Reducers?

Let's look at the Explain plan.

----

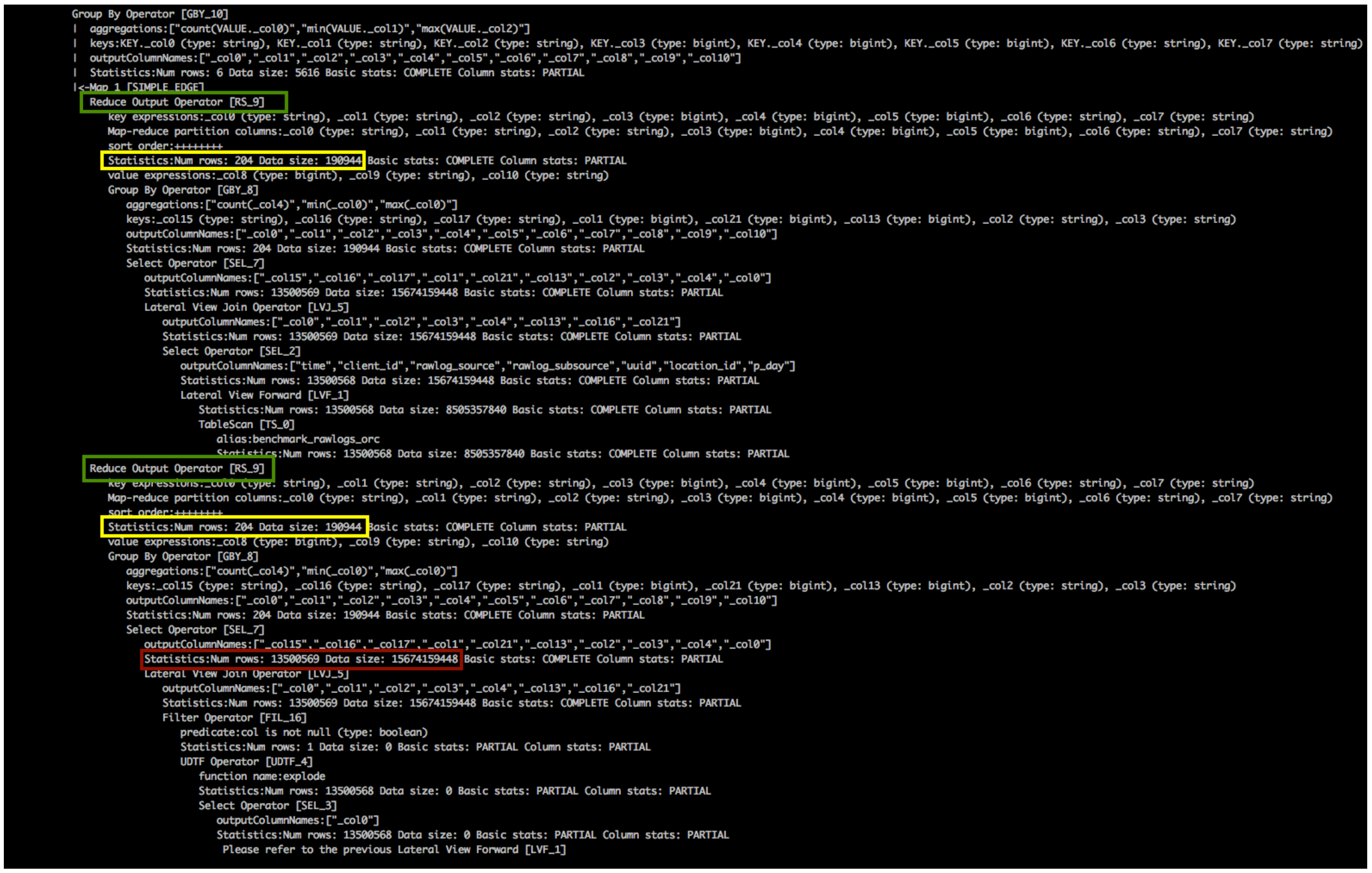
2. The LONGGGGGG Explain Plan



```
hive> explain SELECT client_id,
    >     ipno,
    >     COUNT(uuid),
    >     MIN(time),
    >     MAX(time),
    >     FROM_UNIXTIME(UNIX_TIMESTAMP()) AS row_created_timestamp,
    >     location_id,
    >     rawlog_source,
    >     rawlog_subsource,
    >     p_silo, p_day,
    >     p_clienthash
    > FROM rawlogs.benchmark_rawlogs_orc
    > LATERAL VIEW EXPLODE(ips) lv_ip AS ipno
    > WHERE ipno IS NOT NULL
    >     AND p_silo = 'logd'
    >     AND p_day >= '2016-02-07'
    >     AND p_day < '2016-02-08'
    >     AND p_clienthash = '5541133_5541346'
    > GROUP BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subsource
    > ORDER BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subsource
    > LIMIT 20;
unix_timestamp(void) is deprecated. Use current_timestamp instead.
OK
Plan not optimized by CBO.

Vertex dependency in root stage
Reducer 2 <- Map 1 (SIMPLE_EDGE)
Reducer 3 <- Reducer 2 (SIMPLE_EDGE)

Stage-0
   Fetch Operator
      limit:20
      Stage-1
         Reducer 3
         File Output Operator [FS_15]
            compressed:false
            Statistics:Num rows: 6 Data size: 6234 Basic stats: COMPLETE Column stats: PARTIAL
            table:{"input format:":"org.apache.hadoop.mapred.TextInputFormat","output format:":"org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat","serde:":"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe"}
            Limit [LIM_14]
               Number of rows:20
               Statistics:Num rows: 6 Data size: 6234 Basic stats: COMPLETE Column stats: PARTIAL
               Select Operator [SEL_13]
               |  outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col5","_col6","_col7","_col8","_col9","_col10","_col11"]
               |  Statistics:Num rows: 6 Data size: 6234 Basic stats: COMPLETE Column stats: PARTIAL
               |<-Reducer 2 [SIMPLE_EDGE]
                  Reduce Output Operator [RS_12]
                     key expressions:_col9 (type: string), _col10 (type: string), _col11 (type: string), _col0 (type: bigint), _col1 (type: bigint), _col6 (type: bigint), _col7 (type: string), _col8 (type: string)
                     sort order:++++++++
                     Statistics:Num rows: 6 Data size: 6234 Basic stats: COMPLETE Column stats: PARTIAL
                     value expressions:_col2 (type: bigint), _col3 (type: string), _col4 (type: string)
                     Select Operator [SEL_11]
                        outputColumnNames:["_col0","_col1","_col10","_col11","_col2","_col3","_col4","_col6","_col7","_col8","_col9"]
                        Statistics:Num rows: 6 Data size: 6234 Basic stats: COMPLETE Column stats: PARTIAL
                        Group By Operator [GBY_10]
                        |  aggregations:["count(VALUE._col0)","min(VALUE._col1)","max(VALUE._col2)"]
                        |  keys:KEY._col0 (type: string), KEY._col1 (type: string), KEY._col2 (type: string), KEY._col3 (type: bigint), KEY._col4 (type: bigint), KEY._col5 (type: bigint), KEY._col6 (type: string), KEY._col7 (type: string)
                        |  outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col5","_col6","_col7","_col8","_col9","_col10"]
                        |  Statistics:Num rows: 6 Data size: 5616 Basic stats: COMPLETE Column stats: PARTIAL
                        |<-Map 1 [SIMPLE_EDGE]
                           Reduce Output Operator [RS_9]
                              key expressions:_col0 (type: string), _col1 (type: string), _col2 (type: string), _col3 (type: bigint), _col4 (type: bigint), _col5 (type: bigint), _col6 (type: string), _col7 (type: string)
```



```
Group By Operator [GBY_10]
|  aggregations:["count(VALUE._col0)","min(VALUE._col1)","max(VALUE._col2)"]
|  keys:KEY._col0 (type: string), KEY._col1 (type: string), KEY._col2 (type: string), KEY._col3 (type: bigint), KEY._col4 (type: bigint), KEY._col5 (type: bigint), KEY._col6 (type: string), KEY._col7 (type: string)
|  outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col5","_col6","_col7","_col8","_col9","_col10"]
|  Statistics:Num rows: 6 Data size: 5616 Basic stats: COMPLETE Column stats: PARTIAL
|<-Map 1 [SIMPLE_EDGE]
   Reduce Output Operator [RS_9]
      key expressions:_col0 (type: string), _col1 (type: string), _col2 (type: string), _col3 (type: bigint), _col4 (type: bigint), _col5 (type: bigint), _col6 (type: string), _col7 (type: string)
      Map-reduce partition columns:_col0 (type: string), _col1 (type: string), _col2 (type: string), _col3 (type: bigint), _col4 (type: bigint), _col5 (type: bigint), _col6 (type: string), _col7 (type: string)
      sort order:++++++++
      Statistics:Num rows: 204 Data size: 190944 Basic stats: COMPLETE Column stats: PARTIAL
      value expressions:_col8 (type: bigint), _col9 (type: string), _col10 (type: string)
      Group By Operator [GBY_8]
         aggregations:["count(_col4)","min(_col0)","max(_col0)"]
         keys:_col15 (type: string), _col16 (type: string), _col17 (type: string), _col1 (type: bigint), _col21 (type: bigint), _col13 (type: bigint), _col2 (type: string), _col3 (type: string)
         outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col5","_col6","_col7","_col8","_col9","_col10"]
         Statistics:Num rows: 204 Data size: 190944 Basic stats: COMPLETE Column stats: PARTIAL
         Select Operator [SEL_7]
            outputColumnNames:["_col15","_col16","_col17","_col1","_col21","_col13","_col2","_col3","_col4","_col0"]
            Statistics:Num rows: 13500569 Data size: 15674159448 Basic stats: COMPLETE Column stats: PARTIAL
            Lateral View Join Operator [LVJ_5]
               outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col13","_col16","_col21"]
               Statistics:Num rows: 13500569 Data size: 15674159448 Basic stats: COMPLETE Column stats: PARTIAL
               Select Operator [SEL_2]
                  outputColumnNames:["time","client_id","rawlog_source","rawlog_subsource","uuid","location_id","p_day"]
                  Statistics:Num rows: 13500568 Data size: 15674159448 Basic stats: COMPLETE Column stats: PARTIAL
                  Lateral View Forward [LVF_1]
                     Statistics:Num rows: 13500568 Data size: 8505357840 Basic stats: COMPLETE Column stats: PARTIAL
                     TableScan [TS_0]
                        alias:benchmark_rawlogs_orc
                        Statistics:Num rows: 13500568 Data size: 8505357840 Basic stats: COMPLETE Column stats: PARTIAL
   Reduce Output Operator [RS_9]
      key expressions:_col0 (type: string), _col1 (type: string), _col2 (type: string), _col3 (type: bigint), _col4 (type: bigint), _col5 (type: bigint), _col6 (type: string), _col7 (type: string)
      Map-reduce partition columns:_col0 (type: string), _col1 (type: string), _col2 (type: string), _col3 (type: bigint), _col4 (type: bigint), _col5 (type: bigint), _col6 (type: string), _col7 (type: string)
      sort order:++++++++
      Statistics:Num rows: 204 Data size: 190944 Basic stats: COMPLETE Column stats: PARTIAL
      value expressions:_col8 (type: bigint), _col9 (type: string), _col10 (type: string)
      Group By Operator [GBY_8]
         aggregations:["count(_col4)","min(_col0)","max(_col0)"]
         keys:_col15 (type: string), _col16 (type: string), _col17 (type: string), _col1 (type: bigint), _col21 (type: bigint), _col13 (type: bigint), _col2 (type: string), _col3 (type: string)
         outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col5","_col6","_col7","_col8","_col9","_col10"]
         Statistics:Num rows: 204 Data size: 190944 Basic stats: COMPLETE Column stats: PARTIAL
         Select Operator [SEL_7]
            outputColumnNames:["_col15","_col16","_col17","_col1","_col21","_col13","_col2","_col3","_col4","_col0"]
            Statistics:Num rows: 13500569 Data size: 15674159448 Basic stats: COMPLETE Column stats: PARTIAL
            Lateral View Join Operator [LVJ_5]
               outputColumnNames:["_col0","_col1","_col2","_col3","_col4","_col13","_col16","_col21"]
               Statistics:Num rows: 13500569 Data size: 15674159448 Basic stats: COMPLETE Column stats: PARTIAL
               Filter Operator [FIL_16]
                  predicate:col is not null (type: boolean)
                  Statistics:Num rows: 1 Data size: 0 Basic stats: PARTIAL Column stats: PARTIAL
                  UDTF Operator [UDTF_4]
                     function name:explode
                     Statistics:Num rows: 13500568 Data size: 0 Basic stats: PARTIAL Column stats: PARTIAL
                     Select Operator [SEL_3]
                        outputColumnNames:["_col0"]
                        Statistics:Num rows: 13500568 Data size: 0 Basic stats: PARTIAL Column stats: PARTIAL
                         Please refer to the previous Lateral View Forward [LVF_1]
```

Let's look at the relevant portions of this explain plan. We see in Red that in the Reducers stage, **14.5 TB of data, across 13 million rows are processed**. This is a lot of data to funnel through just two reducers.

The final output of the reducers is just **190944 bytes (in yellow)**, after initial group bys of count, min and max.

We need to increase the number of reducers.

----

3. Set Tez Performance Tuning Parameters

When Tez executes a query, it initially determines the number of reducers it needs and automatically adjusts as needed based on the number of bytes processed.

- Manually set number of Reducers (not recommended)

To manually set the number of reduces we can use parameter **mapred.reduce.tasks.**

By default it is set to -1, which lets Tez automatically determine the number of reducers.

However you are manually set it to the number of reducer tasks (not recommended)

```
> set mapred.reduce.tasks = 38;
```

It is better let Tez determine this and make the proper changes within its framework, instead of using the brute force method.

```
> set mapred.reduce.tasks = -1;
```

- How to Properly Set Number of Reducers

First we double check if auto reducer parallelism is on. The parameter is **hive.tez.auto.reducer.parallelism**

See

https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-hive.tez.a

It is set to true.

#Turn on Tez' auto reducer parallelism feature. When enabled, Hive will still estimate data sizes and set parallelism estimates. Tez will sample source vertices' output sizes and adjust the estimates at runtime as necessary.

> set hive.tez.auto.reducer.parallelism; > set hive.tez.auto.reducer.parallelism = true;

This is the first property that determines the initial number of reducers once Tez starts the query.

Then, there are two boundary parameters

- **hive.tex.min.partition.factor**
- **hive.tez.max.partition.factor**

#When auto reducer parallelism is enabled this factor will be used to put a lower limit to the number of reducers that Tez specifies.

> hive.tez.min.partition.factor=0.25;

# When auto reducer parallelism is enabled this factor will be used to over-partition data in shuffle edges.

gt; hive.tez.max.partition.factor=2.0;

More on this parameter later.

The third property is hive.exec.reducers.max which determines the maximum number of reducers. **By default it is 1099.**

The final parameter that determines the initial number of reducers is **hive.exec.reducers.bytes.per.reducer**

By default **hive.exec.reducers.bytes.per.reducer is set to 256MB, specifically 258998272 bytes.**

The FORMULA

So to put it all together Hive/ Tez estimates number of reducers using the following formula and then schedules the Tez DAG.

```
Max(1, Min(hive.exec.reducers.max [1099], ReducerStage estimate/hive.exec.reducers.bytes.pe
```

So in our example since the RS output is **190944 bytes,** the number of reducers will be:

> Max(1, Min(1099, 190944/258998272)) x 2

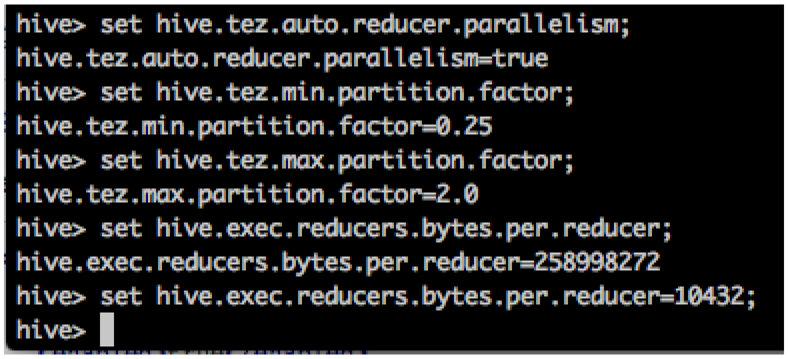
> Max (1, Min(1099, 0.00073)) x 2 = 1 x 2 = 2

Hence the 2 Reducers we initially observe.

----

4. Increasing Number of Reducers, the Proper Way

Let's set hive.exec.reducers.bytes.per.reducer to 10 MB about 10432

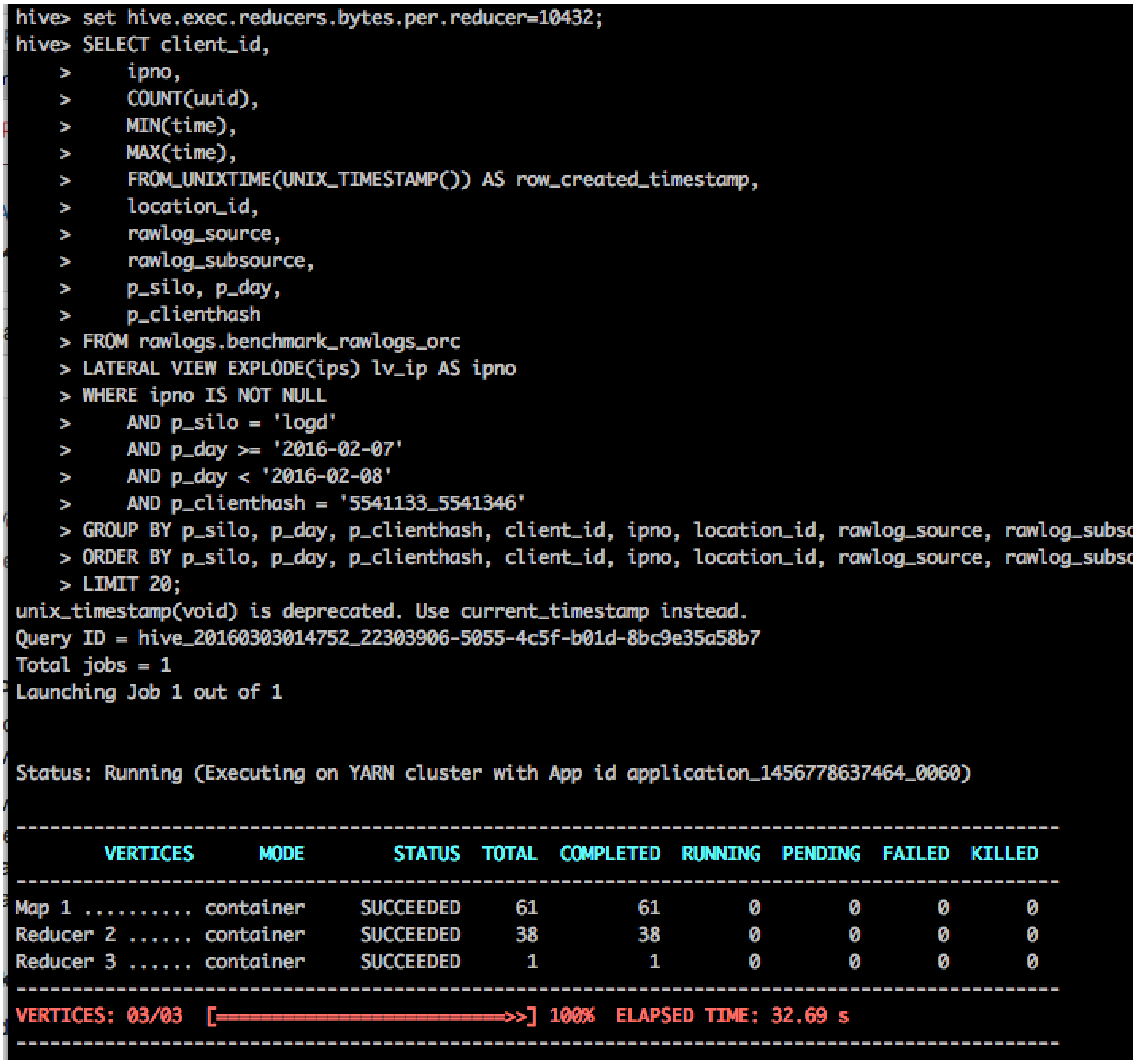


```
hive> set hive.tez.auto.reducer.parallelism;
hive.tez.auto.reducer.parallelism=true
hive> set hive.tez.min.partition.factor;
hive.tez.min.partition.factor=0.25
hive> set hive.tez.max.partition.factor;
hive.tez.max.partition.factor=2.0
hive> set hive.exec.reducers.bytes.per.reducer;
hive.exec.reducers.bytes.per.reducer=258998272
hive> set hive.exec.reducers.bytes.per.reducer=10432;
hive>
```

The new number of reducers count is

> Max(1, Min(1099, 190944/10432)) x 2

> Max (1, Min(1099, 18.3)) x 2 = 19 (rounded up) x 2 = 38



```
hive> set hive.exec.reducers.bytes.per.reducer=10432;
hive> SELECT client_id,
    >     ipno,
    >     COUNT(uuid),
    >     MIN(time),
    >     MAX(time),
    >     FROM_UNIXTIME(UNIX_TIMESTAMP()) AS row_created_timestamp,
    >     location_id,
    >     rawlog_source,
    >     rawlog_subsource,
    >     p_silo, p_day,
    >     p_clienthash
    > FROM rawlogs.benchmark_rawlogs_orc
    > LATERAL VIEW EXPLODE(ips) lv_ip AS ipno
    > WHERE ipno IS NOT NULL
    >     AND p_silo = 'logd'
    >     AND p_day >= '2016-02-07'
    >     AND p_day < '2016-02-08'
    >     AND p_clienthash = '5541133_5541346'
    > GROUP BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subsc
    > ORDER BY p_silo, p_day, p_clienthash, client_id, ipno, location_id, rawlog_source, rawlog_subsc
    > LIMIT 20;
unix_timestamp(void) is deprecated. Use current_timestamp instead.
Query ID = hive_20160303014752_22303906-5055-4c5f-b01d-8bc9e35a58b7
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1456778637464_0060)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     61         61        0        0       0       0
Reducer 2 ...... container     SUCCEEDED     38         38        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 32.69 s
--------------------------------------------------------------------------------
```

Query takes 32.69 seconds now, an improvement.

----

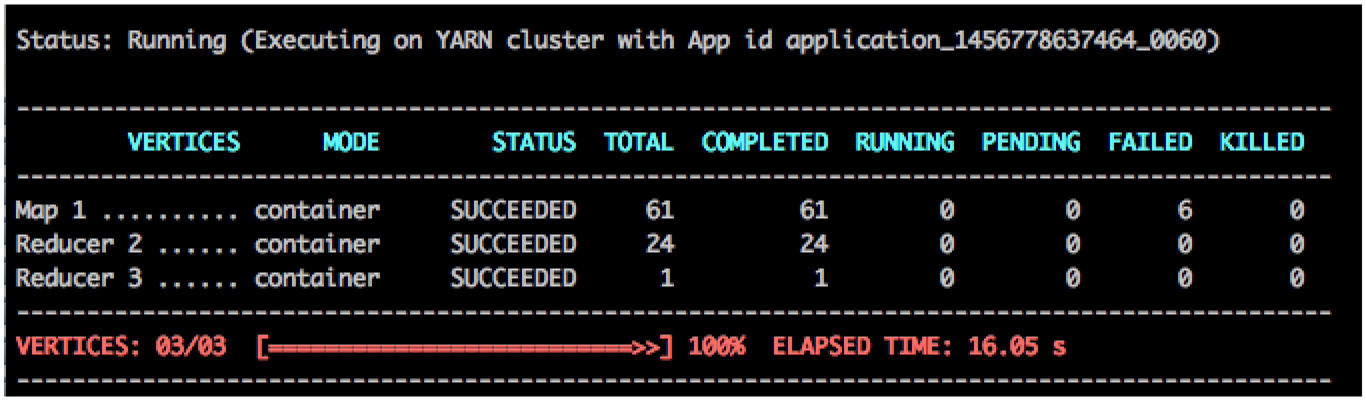
5. More reducers does not always mean Better performance

Let's set hive.exec.reducers.bytes.per.reducer to 15.5 MB about 15872

The new number of reducers count is

> Max(1, Min(1099, 190944/15360)) x 2

> Max (1, Min(1099, 12)) x 2 = 12 x 2 = 24



```
Status: Running (Executing on YARN cluster with App id application_1456778637464_0060)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     61         61        0        0       6       0
Reducer 2 ...... container     SUCCEEDED     24         24        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 16.05 s
--------------------------------------------------------------------------------
```

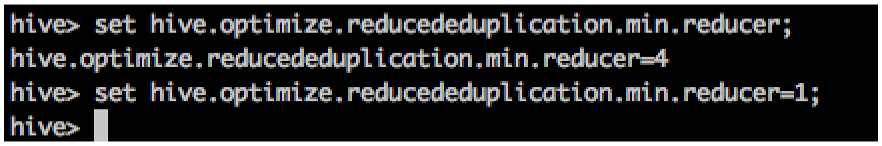
Performance is BETTER with 24 reducers than with 38 reducers.

----

7. Reducing number of Reducer Stages

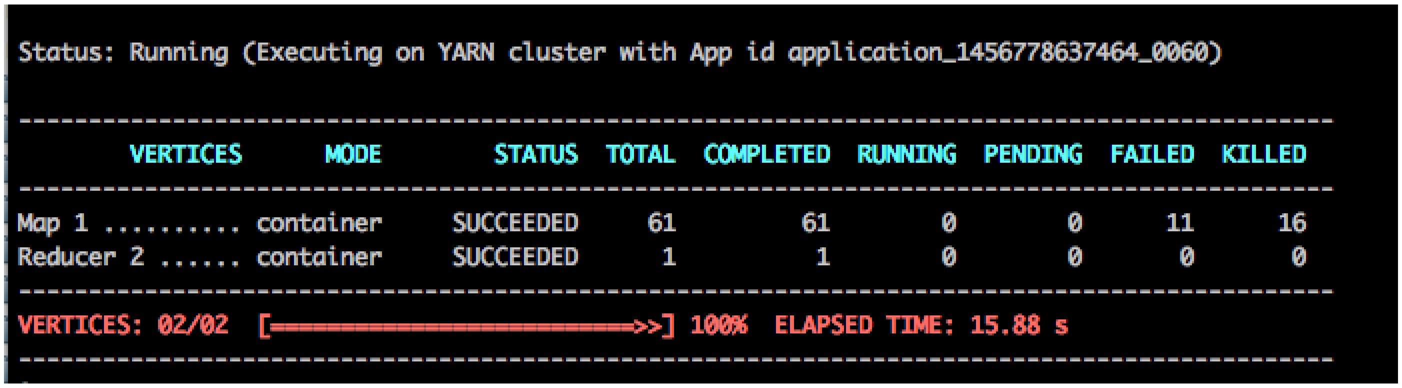
Since we have BOTH a Group By and an Order by in our query, looking at the explain plan, perhaps we can combine that into one reducer stage.

The parameter for this is **hive.optimize.reducededuplication.min.reducer which by default is 4.**



```
hive> set hive.optimize.reducededuplication.min.reducer;
hive.optimize.reducededuplication.min.reducer=4
hive> set hive.optimize.reducededuplication.min.reducer=1;
hive>
```

Setting this to 1, when we execute the query we get



```
Status: Running (Executing on YARN cluster with App id application_1456778637464_0060)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     61         61        0        0      11      16
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 15.88 s
--------------------------------------------------------------------------------
```

Performance is BETTER with ONE reducer stage at 15.88 s.

NOTE: Because we also had a LIMIT 20 in the statement, this worked also. When LIMIT was removed, we have to resort to estimated the right number of reducers instead to get better performance.

Summary

While we can set manually the number of reducers mapred.reduce.tasks, this is **NOT RECOMMENDED**

```
set mapred.reduce.tasks = 38;
```

Tez does not actually have a reducer count when a job starts – it always has a maximum reducer count and that's the number you get to see in the initial execution, which is controlled by 4 parameters.

The 4 parameters which control this in Hive are

hive.tez.auto.reducer.parallelism=true;

hive.tez.min.partition.factor=0.25;

hive.tez.max.partition.factor=2.0;

hive.exec.reducers.bytes.per.reducer=1073741824; // 1gb

You can get wider or narrower distribution by messing with those last 3 parameterss (preferably only the min/max factors, which are merely guard rails to prevent bad guesses).

Hive/ Tez estimates number of reducers using the following formula and then schedules the Tez DAG.

```
Max(1, Min(hive.exec.reducers.max [1099], ReducerStage estimate/hive.exec.reducers.bytes.pe
```

Then as map tasks finish, it inspects the output size counters for tasks to estimate the final output size then reduces that number to a lower number by combining adjacent reducers.

The total # of mappers which have to finish, where it starts to decide and run reducers in the nest stage is determined by the following parameters.

tez.shuffle-vertex-manager.min-src-fraction=0.25;

tez.shuffle-vertex-manager.max-src-fraction=0.75;

This indicates that the decision will be made between 25% of mappers finishing and 75% of mappers finishing, provided there's at least 1Gb of data being output (i.e if 25% of mappers don't send 1Gb of data, we will wait till at least 1Gb is sent out).

Once a decision has been made once, it cannot be changed as some reducers will already be running & might lose state if we do that. You can get more & more accurate predictions by increasing the fractions.

APPENDIX

Hive-2.0 (only) improvements

Now that we have a total # of reducers, but you might not have capacity to run all of them at the same time - so you need to pick a few to run first, the ideal situation would be to start off the reducers which have the most amount of data (already) to fetch first, so that they can start doing useful work instead of starting reducer #0 first (like MRv2) which may have very little data pending.

tez.runtime.report.partition.stats=true;

tez.runtime.pipelined-shuffle.enabled=true;

The first flag there is pretty safe, but the second one is a bit more dangerous as it allows the reducers to fetch off tasks which haven't even finished (i.e mappers failing cause reducer failure, which is optimistically fast, but slower when there are failures – bad for consistent SLAs).

Finally, we have the sort buffers which are usually tweaked & tuned to fit, but you can make it much faster by making those allocations lazy (i.e allocating 1800mb contigously on a 4Gb container will cause a 500-700ms gc pause, even if there are 100 rows to be processed).

```
tez.runtime.pipelined.sorter.lazy-allocate.memory=true;
```

Reference:

https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

http://hortonworks.com/blog/apache-tez-dynamic-graph-reconfiguration/

http://www.slideshare.net/t3rmin4t0r/hivetez-a-performance-deep-dive **and**

http://www.slideshare.net/ye.mikez/hive-tuning **(Mandatory)**

See also

http://www.slideshare.net/AltorosBY/altoros-practical-steps-to-improve-apache-hive-performance

http://www.slideshare.net/t3rmin4t0r/data-organization-hive-meetup

http://www.slideshare.net/InderajRajBains/using-apache-hive-with-high-performance

Special thanks also to Gopal for assisting me with understanding this.