Understanding Rebalancing, Part 2: Optimizing for Rebalancing

HPE Vertica Analytic Database

Document Release Date: 4/8/2016

Contents

About Rebalancing

After you add or remove one or more nodes in a Vertica cluster, the data on the existing and new nodes must be adjusted. For optimal performance, the data should be balanced across all nodes. Vertica calls this process *rebalancing*.

After you rebalance your cluster, the data storage and workload is balanced across all nodes in the cluster. Rebalancing is complex: CPU-, disk-, and network-intensive. Because rebalancing requires a large amount of data movement, the process can take a long time.

This document is Part 2 in a two-part series about rebalancing. Part 2 describes the steps to take before, during, and after rebalancing to:

- Prepare for rebalancing
- Monitor the rebalance operation
- Review the results of rebalancing

Part 1 explains what happens during rebalancing.

Start the Rebalance Process

The rebalancing process needs to share machine resources with on-going queries and needs exclusive access to resources/locks for small periods of time. To avoid possible concurrency issues, schedule the rebalancing during a light load period.

Some examples of concurrency issues that can occur during load are:

- If rebalancing owns a lock that a COPY operation needs, the COPY operation times out.
- If COPY owns a lock, rebalancing pauses until the COPY completes.

You can start the rebalancing using any of the three ways described in the Vertica documentation:

- Rebalancing Data Using the Administration Tools UI
- Rebalancing Data Using Management Console
- Rebalancing Data Using SQL Functions

Monitor Rebalancing

Monitor the rebalancing process to make sure it is proceeding without problems. The following sections describe techniques for monitoring the rebalancing process.

Monitor Tables Being Rebalanced

To monitor which tables Vertica is rebalancing at any time, run:

```
=> SELECT * FROM rebalance_table_status;
```

If any DML or DDL operations interfere with the rebalancing of certain tables, you see this error message:

```
ERROR 3007: DDL statement interfered with this statement
```

Monitor Vertica System Tables

Review the following system tables during rebalancing:

- REBALANCE_TABLE_STATUS—For each database table, contains:
  - Amount of data that has been separated
  - Amount of data currently being separated
  - Amount of data that still needs to be separated
- REBALANCE_PROJECTION_STATUS—For each projection, contains:
  - Amount of data that has been separated
  - Amount of data currently being separated
  - Amount of data that still needs to be separated

Queries for Monitoring Cluster Rebalance

This section provides sample queries that help you monitor the rebalance operations on your cluster.

Monitor Active Rebalance Operations

The following query provides information about the currently running rebalance operations:

- Node that the rebalance is running on
- Session ID for the rebalance operation
- Time the rebalance started. This can give you an idea about how long the rebalance is taking.
- Currently executing rebalance task

```
=> SELECT node_name, session_id, session_start_timestamp, description
   FROM system_sessions
   WHERE session_type = 'REBALANCE_CLUSTER' AND is_active;
 node_name |      session_id      |    session_start_timestamp    |      description  ----------+---------------------+--------------------------
---+----------------------------------------------------------------------------------------------------
 node001   | user-9956:0x2fb6 | 2016-02-11 21:36:16.045711-05 | Txn: a000000000b167 'RebalanceUnsegmentedTask
 node001   | user-9956:0x2fb6 | 2016-02-11 21:36:16.045711-05 | Txn: a000000000b168 'CREATE PROJECTION public.dim7_node006 /*+basename
(dim7),createtype(P)*/  (  a,  b ) AS  SELECT dim7.a, dim7.b  FROM public.dim7  ORDER BY dim7.a, dim7.b UNSEGMENTED NODE node006;
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b16b 'Refresh: Evaluating which projection to refresh'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b16c 'Refresh: (Table: public.dim7) (Projection: public.dim7_node006)'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b16f 'Refresh: through recovery (Table: public.dim7) (Projection:
public.dim7_node006)'
 node001   | user-9956:0x2fe1 | 2016-02-11 21:36:16.391713-05 | Txn: a000000000b17a 'RebalanceUnsegmentedTask'
 node001   | user-9956:0x2fe1 | 2016-02-11 21:36:16.391713-05 | Txn: a000000000b17b 'CREATE PROJECTION public.dim7_node007 /*+basename
(dim7),createtype(P)*/(a,b) AS  SELECT dim7.a, dim7.b  FROM public.dim7  ORDER BY dim7.a, dim7.b UNSEGMENTED NODE node007;  '
 node001   | user-9956:0x2fe1 | 2016-02-11 21:36:16.391713-05 | Txn: a000000000b17d 'CREATE PROJECTION public.dim7_node007 /*+basename
(dim7),createtype(P)*/(a,b) AS  SELECT dim7.a, dim7.b  FROM public.dim7  ORDER BY dim7.a, dim7.b UNSEGMENTED NODE node007;  '
 node001   | user-9956:0x2fe3 | 2016-02-11 21:36:16.41582-05  | Txn: a000000000b17e 'RebalanceUnsegmentedTask' node001   | user-9956:0x2fc0 | 2016-
02-11 21:36:16.13772-05  | Txn: a000000000b17f 'Refresh: through recovery (Table: public.dim7) (Projection: public.dim7_node006)'
 node001   | user-9956:0x2fe3 | 2016-02-11 21:36:16.41582-05  | Txn: a000000000b180 'CREATE PROJECTION public.dim7_node008 /*+basename
(dim7),createtype(P)*/(a,b) AS  SELECT dim7.a,dim7.b  FROM public.dim7  ORDER BY dim7.a, dim7.b UNSEGMENTED NODE node008;  '
 node001   | user-9956:0x2fed | 2016-02-11 21:36:16.483142-05 | Txn: a000000000b182 'Refresh: Evaluating which projection to refresh'
 node001   | user-9956:0x2fed | 2016-02-11 21:36:16.483142-05 | Txn: a000000000b183 'Refresh: (Table: public.dim7) (Projection: public.dim7_node007)'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b184 'Refresh: Evaluating which projection to refresh'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b18a 'getRowCountsForProj'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b18b 'analyze_row_count'
 node001   | user-9956:0x2fc0 | 2016-02-11 21:36:16.13772-05  | Txn: a000000000b18c 'Refresh: gather PJP statistics (Table: public.dim7) (Projection:
public.dim7_node006)' node001   | user-9956:0x416c | 2016-02-11 21:37:34.654137-05 | Txn: a000000000bb02 'RebalanceElasticTask'
                       ...
```

Monitor the Overall Progress of the Rebalance Operation

The following query monitors the progress of each currently executing rebalance operation and returns:

- Method used to rebalance the current projection:
  - REFRESH
  - REPLICATE
  - ELASTIC_CLUSTER
- Status of that method
- Number of projections

```
=> SELECT rebalance_method Rebalance_method, Status, COUNT(*) as Count
from (
      select rebalance_method,
      case
      when
      (
      separated_percent   =  100
  and transferred_percent =  100
      ) then 'Completed'
      when (
            separated_percent   <> 0
        and separated_percent   <> 100
           )
        or (
            transferred_percent <> 0
        and transferred_percent <> 100
           ) then 'In Progress'
             else 'Queued'
             end as Status
      FROM rebalance_projection_status
      WHERE is_latest
    ) AS tab
GROUP BY 1, 2
ORDER BY 1, 2;
 Rebalance_method |   Status    | Count
------------------+-------------+-------
 ELASTIC_CLUSTER  | Completed   |     8
 ELASTIC_CLUSTER  | In Progress |     2
 ELASTIC_CLUSTER  | Queued      |     2
 REPLICATE        | Completed   |    50
(4 rows)
```

Monitor the Rebalance Activity of Unsegmented Projections

The following query returns information about the unsegmented projections that are being currently refreshed:

- Session ID of the currently executing rebalance operation
- Name of the unsegmented projection being refreshed
- Current status of the refresh operation
- Method being used to refresh the projection
- How far the refresh has progressed. Possible phases are:
  - Current
  - Historical

```
=> SELECT session_id,
   projection_name,
   refresh_status,
   refresh_method,
   refresh_phase
   FROM projection_refreshes
   WHERE refresh_method = 'rebalance'
   AND is_executing;
 session_id      | projection_name | refresh_status | refresh_method | refresh_phase
----------------------+-----------------+----------------+----------------+--------------
user-9956:0x8cf0 | dim2_node004    | refreshing     | rebalance      | current
user-9956:0x8d24 | dim2_node005    | refreshing     | rebalance      | historicaluser-9956:0x8d25 |
dim2_node006    | refreshing     | rebalance      | historical
(3 rows)
```

Monitor the Rebalance Activity of Segmented Projections

The following query returns information about the segmented projections that are being currently refreshed:

- Projection name
- Method being used to refresh the projection
- Percentage of ROS containers separated
- Percentage of ROS containers transferred to their destination node

```
=> SELECT projection_name, rebalance_method, separated_percent, transferred_percent
   FROM rebalance_projection_status
   WHERE rebalance_method    =  'ELASTIC_CLUSTER'
   AND (
       separated_percent   <> 0
   AND separated_percent   <> 100
       )
   OR (
      transferred_percent <> 0
  AND transferred_percent <> 100
      )
  AND is_latest;

 projection_name | rebalance_method | separated_percent | transferred_percent
-----------------+------------------+-------------------+---------------------
fact5_b1        | ELASTIC_CLUSTER  |           100.00 |              13.23
fact4_b1        | ELASTIC_CLUSTER  |            78.77 |               0.00
fact5_b0        | ELASTIC_CLUSTER  |           100.00 |              13.63
(3 rows)
```

Monitor Active Tuple Mover Operations for Separation Phase of Segmented Projections

The following query returns information about how many processes are separating ROS containers on each node:

- Projection name
- Node name
- Time the Tuple Mover started separating ROS containers for the specified projections

```
=> SELECT
   TM.projection_name,
   TM.node_name,
   TM.operation_start_timestamp
   FROM tuple_mover_operations TM
   JOIN system_sessions
   USING (session_id)
   WHERE system_sessions.is_active
   AND
   session_type = 'REBALANCE_CLUSTER'
   AND
   operation_status = 'Running';
 projection_name | node_name |   operation_start_timestamp
-----------------+-----------+------------------------------
 fact1_b0        | node008   | 2016-02-11 22:00:45.589359-05
 fact1_b1        | node008   | 2016-02-11 22:00:45.589632-05
 fact1_b0        | node008   | 2016-02-11 22:00:45.605845-05
 fact3_b0        | node007   | 2016-02-11 21:57:36.339779-05
 fact3_b0        | node008   | 2016-02-11 21:57:36.339917-05
 fact3_b1        | node008   | 2016-02-11 21:57:36.340097-05
 fact1_b0        | node004   | 2016-02-11 22:00:45.588915-05
 fact1_b1        | node004   | 2016-02-11 22:00:45.58907-05
 fact3_b0        | node005   | 2016-02-11 21:57:36.340339-05
(10 rows)
```

Monitor ROS Containers

The following query returns information about ROS containers created and deleted during the rebalance.

- Action taken for that projection's ROS container. Possible values are:
  - Created
  - Deleted
- Name of the projection
- Number of rows stored in the ROS container
- Number of ROS containers

```
=> SELECT CASE
      WHEN (is_destroyed ) then 'deleted'
      ELSE 'created'
   END AS container,
   projection_name,
   SUM(row_count) AS rows_processed,
   COUNT(*) n_containers
   FROM vs_rebalance_separated_storage_containers
   GROUP BY 1, 2
   ORDER BY 1, 2;
 container |    projection_name     | rows_processed | n_containers
-----------+------------------------+----------------+--------------
 created   | fact3_b0               |         465975 |           10
 created   | fact3_b1               |         465975 |           10  deleted   | fact1_b0
  |         196273 |          730  deleted   | fact1_b1               |         195780 |
740
 deleted   | fact2_b0               |         990830 |           71
 deleted   | fact2_b1               |         988365 |           72
 deleted   | fact5_b1               |        6140310 |           72
...
```

```
(20000000 rows)
```

The following query returns details about the ROS containers transferred among segmented and unsegmented projections:

- Projection name
- Source node name
- Target node name
- Number of rows transferred
- Number of bytes transferred

```
=> SELECT
   projection_name,
   from_node_name,
   to_node_name,
   SUM (row_count) rows_transfered,
   SUM (size_in_bytes) bytes_transferred
   FROM vs_rebalance_transferred_storage_containers
   GROUP BY 1, 2, 3
   ORDER BY 1, 2, 3;
 projection_name    | from_node_name | to_node_name | rows_transfered | bytes_transferred
-----------------------+----------------+-------------+-----------------+------------------
dim6_node007          |                | node007      |         3900000 |        1972485567
dim6_node008          |                | node008      |         4000000 |        2023062120
dim7_node001          |                | node001      |          200000 |         101153106
dim7_node002          |                | node002      |          200000 |         101153106
fact1_b1              | node003        | node002      |          104457 |         265520203
fact1_b1              | node003        | node004      |           20671 |          52543973
fact1_b1              | node003        | node005      |           31230 |          79391579
```

Monitor Time Taken to Rebalance Completed Projections

The following query gives you information about the time it took to rebalance each projection:

- Node that the rebalance ran on
- Schema name
- Projection name

- Start time of the rebalance operation
- Time in seconds it took to rebalance that projection

```
=> SELECT
   dc_rebalanced_projections.node_name,
   projection_schema,
   projection_name,
   start_time,
   time -start_time duration
   FROM dc_rebalanced_projections
   ORDER BY 5 DESC;
 node_name | projection_schema | projection_name |          start_time           |    duration
-----------+-------------------+-----------------+-------------------------------+-----------------
node001    | public            | fact3_b1        | 2016-02-11 21:57:36.335312-05 | 00:03:42.695234
node001    | public            | fact3_b0        | 2016-02-11 21:57:36.335312-05 | 00:03:42.69519
node001    | public            | fact1_b1        | 2016-02-11 21:26:05.744551-05 | 00:03:38.692043
node001    | public            | fact1_b0        | 2016-02-11 21:26:05.744551-05 | 00:03:38.691965
node001    | public            | fact1_b1        | 2016-02-11 22:02:13.277392-05 | 00:03:35.077985
node001    | public            | fact1_b0        | 2016-02-11 22:02:13.277392-05 | 00:03:35.077949
```

Rebalance and Lock Contention

If you rebalance your cluster while ETL jobs are running, there could be contention for locks. This contention can cause the job or the rebalance operation to fail.

This topic describes contention issues that might occur if you run a rebalance operation while Vertica is processing an ETL job. It also describes how to prevent them.

Note: Only one cluster-wide rebalance operation can run at a time.

Minimizing Contention

Database operations that might cause lock contention with the Tuple Mover during rebalance are:

- DELETE
- UPDATE
- DROP_PARTION
- SWAP_PARTITION_BETWEEN_TABLES
- MOVE_PARTITION_TO_TABLE

The following sections describe three ways to diagnose, schedule, and manage the rebalance operation to minimize contention:

- Increase Timeout Value
- Give Preference to Rebalance and Tuple Mover Operations
- Manually Rebalance Tables

Increase Timeout Value

If you think your rebalance might contend with data load jobs, you can increase the LockTimeout value. The LockTimeout configuration parameter specifies the amount of time that a query waits for a lock to be released. If the wait time exceeds the LockTimeout value, the query returns an error. The default value is 300 seconds (5 minutes).

Increasing the LockTimeout value improves the chances that your job does not time out while trying to acquire a lock that rebalance is using.

The following query tells you what times of day that ETL jobs held locks for more than five minutes:

```
=> SELECT DATE_TRUNC ('hour', grant_time), node_name,
COUNT(*) number_of_tx, MAX(time - grant_time) max_time_lock_held
FROM dc_lock_releases
WHERE time - grant_time>'5 min'
AND mode IN ('X', 'S', 'O')
AND object_name NOT LIKE  'ElasticCluster'
GROUP BY 1, 2
ORDER BY 4 desc;
```

```
       date_trunc       | node_name | number_of_tx | max_time_lock_held
------------------------+-----------+--------------+--------------------
 2016-03-02 11:00:00-05 | node001   |            2 | 00:43:47.823902
 2016-03-02 11:00:00-05 | node002   |            2 | 00:43:47.691664
 2016-03-02 11:00:00-05 | node003   |            2 | 00:43:47.691906
 2016-03-02 11:00:00-05 | node004   |            1 | 00:30:03.825279
 2016-03-02 12:00:00-05 | node001   |            1 | 00:15:29.677898
 2016-03-02 12:00:00-05 | node002   |            1 | 00:15:29.677404
 2016-03-02 12:00:00-05 | node003   |            1 | 00:15:29.677366
 2016-03-02 12:00:00-05 | node004   |            1 | 00:15:29.677111
(8 rows)
```

The following query returns which specific transactions held locks for more than five minutes:

```
=> \x
Expanded display is on.
```

```
=> SELECT DISTINCT query_requests.transaction_id, statement_id, request
FROM dc_lock_releases JOIN query_requests USING (session_id)
WHERE time - grant_time > '5 min'
AND mode IN ('X','S')
AND object_name NOT LIKE 'ElasticCluster'
ORDER BY statement_id;
```

```
-[ RECORD 1 ]--+-----------------------------------------
transaction_id | 45035996273756069
statement_id   | 1
request        | UPDATE /*+label(UPDATE_u1), DIRECT*/ u1 SET c200 = c200 - 1 where C200 < 100;
-[ RECORD 2 ]--+-----------------------------------------
transaction_id | 45035996273756069
statement_id   | 2
request        | commit;
(2 rows)
```

To avoid contending with these transactions, you can do one of the following:

- Reschedule the rebalance at a time that does not conflict with ETL jobs.
- Increase the LockTimeout parameter so that the query does not time out while you are rebalancing your cluster:

```
=> SELECT GET_CONFIG_PARAMETER('locktimeout');
 GET_CONFIG_PARAMETER
```

```
----------------------
 300
(1 row)

=> SELECT SET_CONFIG_PARAMETER('LockTimeOut, 600)
    SET_CONFIG_PARAMETER
----------------------------
 Parameter set successfully
(1 row)
```

After rebalance completes, remember to reset the LockTimeout parameter to its previous value.

Give Preference to Rebalance and Tuple Mover Operations

Suppose that a DML job tries to access a table that a rebalance has a lock on. By default, the DML job takes that lock and cancels rebalance. The rebalance continues trying to access the table after five minutes and complete the rebalance.

If you want your rebalance to run uninterrupted, give preference to the rebalance process by setting the DMLCancelTM configuration parameter to false. With this setting, DML jobs *cannot* take a lock that an in-progress rebalance holds.

To set DMLCancelTM and start the rebalance, do the following:

```
=> SELECT SET_CONFiG_PARAMETER('DMLCancelTM', false);

....

=> SELECT REBALANCE_CLUSTER();

....
```

If your DML jobs are critical, do *not* change the value of DMLCancelTM to false for a rebalance. Keeping DMLCancelTM set to true allows DML jobs to run.

Consider running the rebalance at a time when it does not conflict with critical DML jobs.

When the rebalance completes, *always* set DMLCancelTM back to true.

```
=> SELECT SET_CONFiG_PARAMETER('DMLCancelTM', true);
```

Manually Rebalance Tables

If you have a lot of tables and need several nights or weekends to rebalance your cluster, you can manually rebalance a fixed number of tables each night or weekend.

You can manually rebalance one or more tables at a time. To avoid contention during manual rebalance, make sure that no ETL job is running.

To rebalance a table, call the REBALANCE_TABLE function:

```
=> SELECT REBALANCE_TABLE('t0');
```

To find out which tables have been rebalanced, are in the processing of being rebalanced, or have not been rebalanced, run this query:

```
=> SELECT table_name,
CASE
WHEN separated_percent + transferred_percent = 20 THEN 'REBALANCED'
WHEN (separated_percent + transferred_percent) < 200
AND (separated_percent + transferred_percent) > 0
THEN 'REBALANCING' ELSE 'NOT REBALANCED YET'
END status
FROM rebalance_table_status WHERE is_latest;

 table_name | case
------------+----------------
t0          | REBALANCED
t1          | NOT REBALANCED YET
t2          | REBALANCING
(3 rows)
```

Contention Errors During Rebalance

There are several contention errors that can occur during rebalance, including the following, which are discussed in the following sections:

- `DDL statement interfered with this statement. Unavailable: lock table for query - Locking failure`
- `Staging table and target table do not match: Projections definition mismatch`
- `Unavailable: [Txn 0xa0000000010113] S lock table - timeout error Timed out`

Changing the Schema

The following actions can occur by another job during rebalance:

- Add or delete columns
- Add or delete projections
- Swap or move partitions

When rebalancing, you may see a locking error. This error occurs when another job tries to lock a table that is already locked by the rebalance operation:

```
ERROR 3007:  DDL statement interfered with this statement
ERROR 5157:  Unavailable: lock table for query - Locking failure: Timed out X locking
Table:public.t0. T held by [user condor (RebalanceElasticTask)]. Your current transaction isolation
level is SERIALIZABLE
```

Swapping Partitions

If you try to swap partitions between two tables, one that has been rebalanced and another that has not been rebalanced, you see the following error:

```
=> SELECT SWAP_PARTITIONS_BETWEEN_TABLES('t0', 1, 1, 't1');
ERROR 7121: Staging table and target table do not match: Projections definition mismatch
```

You can only swap partitions between two tables that have both been rebalanced or neither of which has been rebalanced.

Get Information About Contention Errors

If rebalance fails due to an error or because of lock contention, to get information about the error, run the following query:

```
=> SELECT time, session_id, error_level, node_name, log_message
FROM dc_errors WHERE session_id IN
(
SELECT DISTINCT session_id
FROM dc_session_starts
WHERE session_type = 'REBALANCE_CLUSTER'
)
ORDER BY time DESC;
-[RECORD 1]-------------------------------
time        | 2016-03-05 10:47:08.517557-05
session_id  | eng-g9-046.verticac-2421955:0xad43
error_level | 20
node_name   | node001
log_message | Unavailable: [Txn 0xa0000000010113] S lock table - timeout error Timed out S locking
Table:public.t1. I held
```

Rebalance Restart After Error

If rebalance fails due to an error or is canceled by a DML operation, Vertica tries to rerun the rebalance after 300 seconds (5 minutes). This means that Vertica waits 5 minutes before trying to restart the rebalance.

```
=> SELECT LIST_SERVICES('TM');
```

```
                                      list_services
-------------------------------------------------------------------------------------
Service: 'RebalanceCluster' is enabled , interval 300 second(s)
```

After Rebalancing

After rebalancing completes, check the following:

- Did the rebalancing complete successfully?
- Is the K-safety correct?
- What is the baseline performance?

Success or Failure

If a failure occurs while rebalancing the database, you can rebalance again. If the cause of the failure has been resolved, the rebalance operation continues from where it failed. However, a failed data rebalance can result in out-of-date projections that Vertica cannot remove automatically.

To locate any such projections, query the V_CATALOG.PROJECTIONS system table as follows:

```
=> SELECT projection_name, anchor_table_name, is_prejoin, is_up_to_date
   FROM projections WHERE is_up_to_date = FALSE;
```

To remove out-of-date projections, use the DROP PROJECTION command.

Monitor System Performance with Vertica Validation Utilities

Vertica provides two tools to monitor system performance. However, they can significantly impact system performance, so do not run them during rebalancing.

- `vioperf`: Measures the speed and consistency of your hard drives.
- `vnetperf`: Measures the latency and throughput of your network between nodes.

Use these tools after you add a node to create a performance baseline for the newly expanded cluster.

For More Information

See Understanding Rebalancing Part 1: What Happens During Rebalancing.

For information about rebalancing in the Vertica documentation, see Rebalancing Data Across Nodes.