Vertica におけるノードリカバリ

November, 2018

原文はこちら

MICRO FOCUS

目次

Vertica データベースでは、DOWN 状態のノードは、ノードがダウンした後にコミットされたトランザクションには参加しません。DOWN ノードを再起動した後、そのノードは UP 状態に移行する前にバディーノードから逃したデータをリカバリする必要があります。ノードが UP 状態になると、ノードは完全に回復し、すべてのデータベーストランザクションを処理できる状態になります。

本ドキュメントは、熟練の Vertica ユーザーを対象としており、ノードのリカバリ経験があることを前提としています。本ドキュメントの目的は、次のことについての洞察を提供することです。

- Vertica 9.1.x のノードのリカバリ処理
- ノードのリカバリをモニタリングするツール
- ノードのリカバリに関する問題を診断して解決するためのトラブルシューティングのヒント

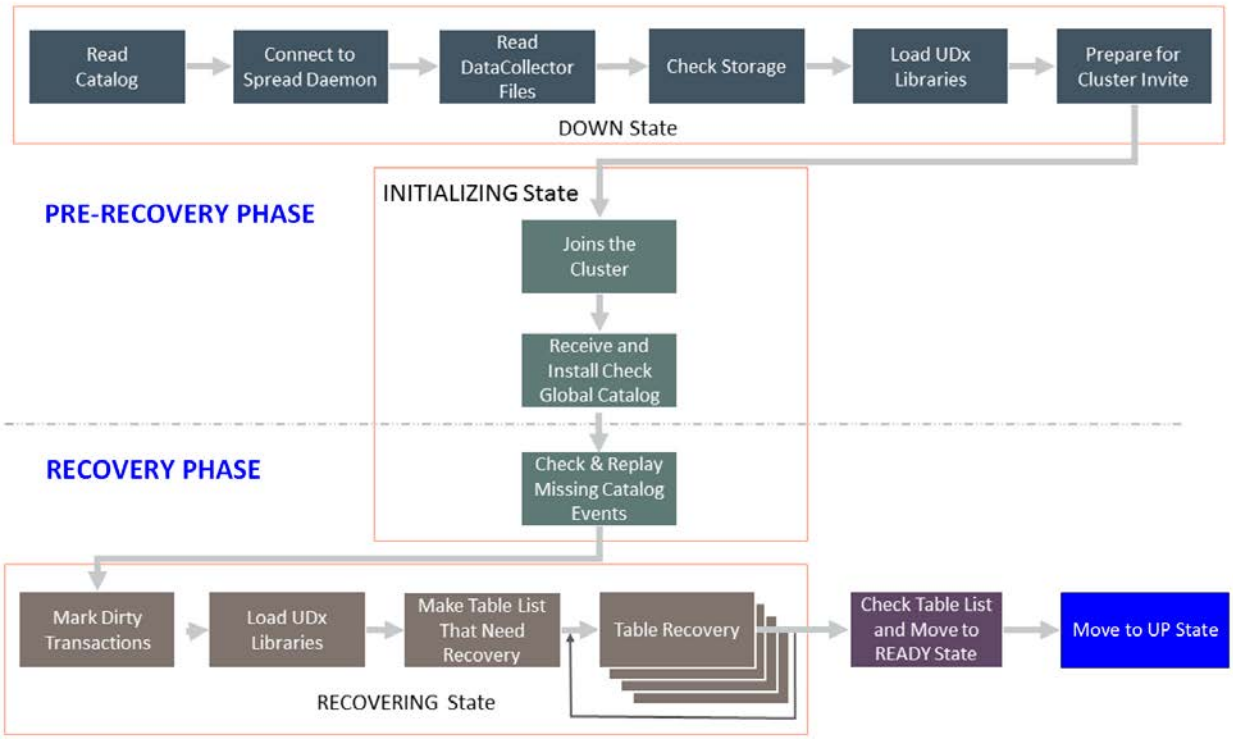
ノードのリカバリ処理

Vertica は、ノードリカバリを 2 つのフェーズで実行します。

- **リカバリ前フェーズ**: このフェーズでは、リカバリノードは次のタスクを実行します。
    - spread デーモンを再起動
    - ディスク上のカタログの読み込み
    - データストレージの検証
    - クラスタへの参加
- **リカバリフェーズ**: このフェーズでは、リカバリノードが次のタスクを実行します。
    - クラスターにロードされた新しいデータの受信
    - UP 状態に移行する前にノードが DOWN 状態になったときに逃したデータのリカバリ

データベースノードが RECOVERING 状態の場合、Tuple Mover はリカバリ中に必要に応じて moveout および mergeout 処理を実行できます。Tuple Mover は、リカバリ中に mergeout を実行し、ROS コンテナをマージし、ROS プッシュバックを防止します。さらに、Tuple Mover は、WOS から ROS へデータを移動する moveout を実行します。そうすることで、リカバリ時にトリクルローディングなどのタスクを実行できるため、持続可能なロード処理が作成され、ROS コンテナが少なくなります。mergeout と moveout の両方が、ノードリカバリのパフォーマンスを向上させることができます。

次の図は、ノードリカバリ処理のすべての段階を示す詳細なワークフローを示しています。



リカバリ前フェーズ

リカバリ前フェーズは、ノードの再起動後直ちに開始され、ノードがクラスターに再び参加するまで実行されます。このフェーズでノードの進行状況を追跡するには、カタログディレクトリにある startup.log ファイルを使用します。startup.log ファイルは、vertica.log ファイルのサブセットです。

リカバリ前フェーズには次の段階があります。

ステージ	説明
カタログの読み込み	リカバリノードはカタログチェックポイントを読み取り、トランザクションログを適用し、カタログオブジェクトを索引付けします。
Spread デーモンの起動と接続	Vertica はリカバリノード上で Spread デーモンを開始し、Spread デーモンに接続します。 ただし、データベースがlarge cluster modeで実行されている場合、制御ノードと呼ばれるノードのサブセットがSpreadデーモンを実行します。
データコレクターファイルの読み込み	リカバリノードは、データコレクターファイルのインベントリを作成します。
データストレージのチェック	• リカバリノードは、カタログで指定されたデータファイルをチェックします。ファイルの存在、権限、および正確性を検証します。 • リカバリノードは、カタログで参照されていないデータファイルを削除します。

UDx ライブラリのロード	リカバリノードは、ユーザー定義拡張（UDx）ライブラリをロードします。
Cluster Invite の準備	リカバリノードは、Spread グループに参加し、メッセージを他のノードにブロードキャストします。リカバリ中のノードは、他のノードからの招待を待って、クラスターに再参加します。
クラスターに参加	招待されると、ノードはクラスターに参加し、ノード状態は INITIALIZING 状態に変わります。
新しいグローバルカタログの受信とインストール	リカバリノードは、クラスター内の他のノードとカタログバージョンを共有します。 リカバリ中のノードのカタログバージョンが UP 状態のノードよりも低い場合、UP 状態のノードは、グローバルカタログを 1 GB のチャンクでリカバリノードに送信します。 リカバリノードは、UP ノードの 1 つから受信した新しいカタログをインストールします。

リカバリフェーズ

リカバリフェーズでは、リカバリノードはデータベースにロードされた新しいデータを受信します。リカバリ中のノードは、リカバリ中のノードのバディーノードからこの時点まで DOWN 状態の間に欠落したデータもリカバリします。

このフェーズの進行状況は、次のシステムテーブルを使用して追跡できます。

- TABLE_RECOVERY_STATUS
- TABLE_RECOVERIES
- PROJECTION_RECOVERIES

注意

Vertica 7.2.x 以降、RECOVERY_STATUS システムテーブルは廃止されました。

リカバリフェーズには次の段階があります。

ステージ	説明
実行されていないカタログイベントをチェックしてリプレイ	リカバリノードは、実行されていないイベントをチェックし、実行されていないイベントをコミットされた順で再実行します。 Vertica カタログには、以下のタイプのカタログイベント、関連するテーブル、およびイベントがコミットされたエポックのログがあります。

	• パーティションの変更 • テーブルのリストア • ノードの置換
ダーティートランザクションをマーク	ダーティートランザクションとは、RECOVERY フェーズの開始前に実行開始されたコミットされていないトランザクションのことです。 リカバリ中、Vertica はダーティトランザクションをマークし、リカバリノードの状態は INITIALIZING から RECOVERING に変わります。RECOVERING 状態のノードは、すべてのデータロード処理の実行対象となります。
UDx ライブラリのロード	リカバリノードは、UP 状態のノードから受信した UDx ライブラリをロードします。
リカバリが必要なテーブル一覧を作成	リカバリノードは、リカバリが必要なテーブル一覧を作成します。
テーブルリカバリ	テーブルリカバリの詳細な手順については、Steps for Table Recovery を参照ください。 Vertica 7.2.2 以降、Vertica は複数のテーブルを同時にリカバリします。Vertica が同時にリカバリできるテーブルの数は、次の項目によって決まります。 • RECOVERY リソースプールの MAXCONCURRENCY • テーブル毎のプロジェクション数 Vertica 7.2.x 以降では、テーブルリカバリの順序を指定可能です。詳細については、Vertica ドキュメントの Prioritizing Table Recovery を参照ください。
テーブル一覧を確認し、READY 状態に遷移	リカバリが必要なテーブルのリストが空の場合、Vertica はノードの状態を RECOVERING から READY に変更します。 リストが空でない場合、Vertica は最大 20 回までリカバリ手順を再試行します。これらの繰り返し試行が失敗すると、ノードのリカバリは失敗し、ノードの状態は DOWN に変わります。
UP 状態に遷移	リカバリフェーズの最後のステップにおいて、Vertica はノードの状態を READY から UP に変更します。この時点から、リカバリノードは新しい接続を受け入れ、ロード処理のみだけでなく全てのデータベース上の処理に参画します。

テーブルリカバリの手順

1. **Replay catalog events（カタログイベントのリプレイ）**: リカバリノードは、ノードが DOWN 状態の間にリカバリテーブルが逃したカタログイベントをリプレイします。リカバリノードは、リカバリテーブルの以下のカタログイベントをチェックします。

- Truncate table
- Add column
- Rebalance table
- Alter partition
- Drop partition
- Restore table
- Move partition
- Swap partition
- Database rollback
- Replace node
- Merge projection

2. **Historical recovery フェーズ（履歴リカバリフェーズ）**: 各プロジェクションについて、Vertica は各ノードのチェックポイントエポックを保持します。チェックポイントエポックは、すべてのデータがディスクに保存された時点を表します。
リカバリテーブルに紐付けられたプロジェクションは、テーブルが逃した履歴データをリカバリします。このステップの間、Vertica はプロジェクションのチェックポイントエポックからカレントエポックにデータをリカバリします。
プロジェクションのチェックポイントエポックが 0 の場合、ノードは recovery-by-container の実行計画を使ってプロジェクションデータをリカバリします。Recovery-by-container の実行計画では、リカバリノードはバディノードからストレージコンテナとデリートベクターをコピーします。Incremental 方式では、Vertica は最初にバディープロジェクションのストレージコンテナをスキャンし、必要なプロジェクションデータを取得します。その後、プロジェクションデータは要求に応じてソートされ、リカバリノード上のディスクに書き込まれます。Incremental 方式は、ストレージコンテナのみをリカバリします。したがって、リカバリノードがダウンしているときにコミットされたデリートベクターがあれば、incremental-replay-delete 方式を使用して、incremental 方式に従ってデリートベクターをリカバリします。 Vertica は履歴リカバリの段階ではロックを取得しません。

3. **Recover dirty transactions（ダーティートランザクションのリカバリ）**: RecoveryDirtyTxnWait 構成パラメーターを使用して、リカバリノードがダーティートランザクションのコミットを待つ時間を制御します。

   - リカバリノードがコミットされていないダーティトランザクションを検出することがあります。その場合、デフォルトでは、ノードはトランザクションがコミットされるのを 5 分間待ちます。5 分後、Vertica のリカバリプロセスはコミットされていないすべてのダーティトランザクションを終了します。
   - ダーティトランザクションがコミットされた後、リカバリノードは、ダーティトランザクションによってロードされたデータをリカバリします。Vertica は、この段階でデータのリカバリに incremental および incremental-replay-delete 方式を使用できます。ダーティトランザクションフェーズでロックは取得されません。

4. **Replay delete phase（リプレイデリートフェーズ）**: ノードのリカバリ中に削除が実行されると、手順 2 と 3 ではロックが取得されません。しかし、ノードが RECOVERING 状態に入った後に開始される削除処理は、リカバリノード上にデリートベクターを作成しません。Vertica は、リプレイデリーとフェーズでそれらの欠落したデリートベクターをリカバリします。この段階で、Vertica はリカバリ中のテーブルに対して T

ロックを取り、incremental-replay-delete 方式を使用して、欠落したデリートベクターをリカバリします。この段階で、リカバリノードは、カタログイベントの最後のリプレイからカタログイベントにテーブルに対する矛盾があるかどうかも判断します。リカバリ中のノードが欠落したイベントを検出した場合、このテーブルのリカバリは失敗し、テーブルは「Failed to recover（リカバリーに失敗しました）」とマークされます。その後、リカバリ中のノードは、失敗したテーブルに対してステップ 1 からテーブルリカバリプロセス全体を再試行します。Vertica は、ノードリカバリに失敗する前に最大 20 のリカバリの試行を許可します。

5. **Finish Table Recovery（テーブルリカバリの終了）**: 前述の手順が正常に完了すると、Vertica はテーブルをリカバリ済みとしてマークし、リカバリする必要があるテーブルのリストからテーブルを削除します。この時点から、リカバリされたテーブルはすべての DDL および DML の実行計画に参加します。

テーブルリカバリの例

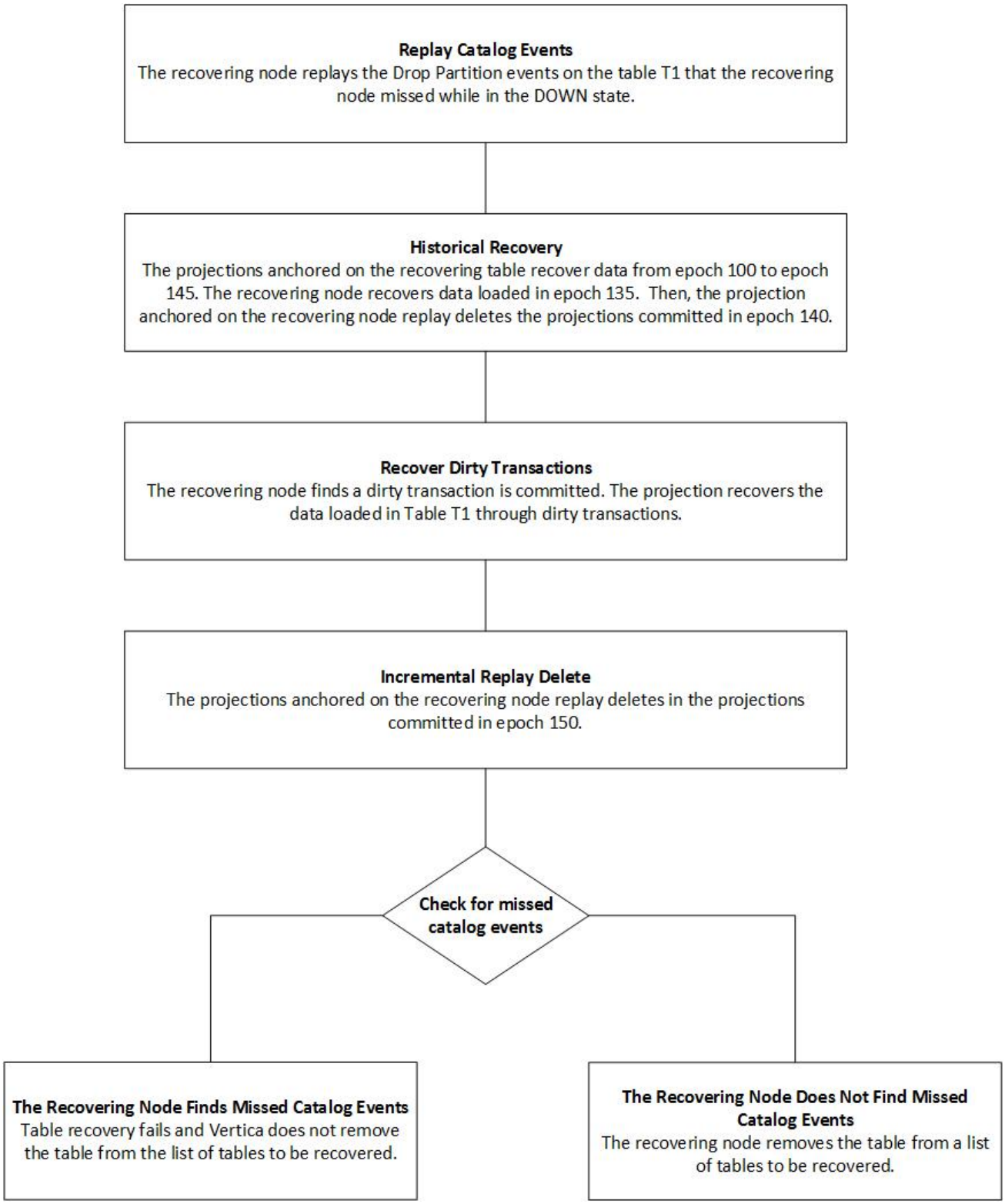
シナリオ: テーブル T1 を持つ 6 ノードクラスターがあります。ノード 5 はエポック 100 でダウンしました。テーブル T1 に紐付けられたプロジェクションのチェックポイントエポックはノード 5 で 99 でした。

想定: Vertica は、ノード 5 が DOWN 状態にあった際に、テーブル T1 上で以下のトランザクションを実行しました。

- エポック 125 で、テーブル T1 のパーティションを削除（Drop）
- エポック 135 で、テーブル T1 にデータをロード
- エポック 140 で、テーブル T1 のデータを削除（Delete）

Vertica はノード 5 を再起動し、ノードの状態をエポック 145 で RECOVERING に変更しました。Vertica は、ノードが RECOVERING 状態に移行した時点で、テーブル T1 にデータをロードするダーティトランザクションを記録しました。ダーティートランザクションのリカバリフェーズ中に、データは、テーブル T1 からエポック 150 で削除されます。

次の図は、テーブルリカバリのさまざまなステップに関する詳細情報を示しています。


Replay Catalog Events
The recovering node replays the Drop Partition events on the table T1 that the recovering node missed while in the DOWN state.
Historical Recovery
The projections anchored on the recovering table recover data from epoch 100 to epoch 145. The recovering node recovers data loaded in epoch 135. Then, the projection anchored on the recovering node replay deletes the projections committed in epoch 140.
Recover Dirty Transactions
The recovering node finds a dirty transaction is committed. The projection recovers the data loaded in Table T1 through dirty transactions.
Incremental Replay Delete
The projections anchored on the recovering node replay deletes in the projections committed in epoch 150.
Check for missed catalog events
The Recovering Node Finds Missed Catalog Events
Table recovery fails and Vertica does not remove the table from the list of tables to be recovered.
The Recovering Node Does Not Find Missed Catalog Events
The recovering node removes the table from a list of tables to be recovered.

ノードリカバリのモニタリング

DOWN 状態のノードがある場合、リカバリプロセスをモニタリングするには、次の手順を実行します。

1. 次のコマンドを実行して、リカバリ中のノードの状態を確認します。

```
$ /opt/vertica/bin/admintools -t view_cluster
     DB    |     Host      |   State
-----------+---------------+------------
mydatabase | 112.17.31.10 | UP
mydatabase | 112.17.31.11 | UP
mydatabase | 112.17.31.12 | INITIALIZING
mydatabase | 112.17.31.17 | UP
_____________________________________
```

2. ノード状態が DOWN または INITIALIZING の場合、tail startup.log を使用してリカバリノードの進行をモニタリングします。

```
$ tail catalog-path/database-name/v_database-
name_node_catalog/startup.log
```

ノードリカバリ中、Vertica はリカバリノード上の startup.log を更新します。これらの更新は、次に示す JSON 情報ブロックの形式で実施されます。

```
{
  "goal" : 477688606,
  "node" : "v_newdb2_node0003",
  "progress" : 146732903,
  "stage" : "Read DataCollector",
  "text" : "Inventory files (bytes)",
  "timestamp" : "2016-03-16 17:28:32.016"
}
```

TABLE_RECOVERY_STATUS、TABLE_RECOVERIES、および PROJECTION_RECOVERIES システムテーブルでは、ノードに関するリカバリステータスは、そのノードの状態が DOWN または INITIALIZING の場合は使用できません。

3. ノード状態が RECOVERING の場合、システムテーブルを使用してノードの進行状況を追跡します

   - 集計されたステータスを表示するには、次のクエリを実行します。

   ```
   => SELECT * FROM TABLE_RECOVERY_STATUS;
   ```

   - テーブルとプロジェクションのリカバリの詳細を表示するには、次のクエリを実行します。

   ```
   => SELECT * FROM TABLE_RECOVERIES;
   => SELECT * FROM PROJECTION_RECOVERIES;
   ```

たとえば、recovery-by-container 方式を使用する場合、履歴リカバリのフェーズでは、次の内容がリカバリシステムテーブルに表示されます。リカバリフェーズおよびリカバリ方式の詳細は、それぞれシステムテーブル TABLE_RECOVERIES および PROJECTION_RECOVERIES に示されています。

```
=> SELECT * FROM TABLE_RECOVERIES;
node_name | table_oid | table_name | status | phase | thread_id |
start_time |
end_time | recover_priority | recover_error | is_historical
-----------+------------------+-----------+-----------+-----------
+-------
-------+-------------------------------+----------+-------------------
--+----
-----------+---------------
node02 | 45035996273819958 | public.t1 | recovering | historical |
7f3abf7fe700
| 2018-04-22 21:41:25.134359-04 | | -9223372036854775807 | | f
(1 row)

=> SELECT * FROM PROJECTION_RECOVERIES;
node_name | projection_id | projection_name | transaction_id |
statement_id |
method | status | progress | detail | start_time | end_time |
runtime_priority
-----------+------------------+----------------+------------------+-
--------
-----+----------------------+----------+----------+----------------+-
-------
----------------------+-------------------------------+--------------
---
node02 | 45035996273820032 | public.t1_b0 | 49539595901075518 | 1 |
recovery-by-container
| finished | | | 2018-04-22 21:41:25.146027-04 | 2018-04-22
21:41:25.166691-04 |
node02 | 45035996273820042 | public.t1_b1 | 49539595901075517 | 1 |
recovery-by-container
| running | | CopyStorage:0/0 | 2018-04-22 21:41:25.145689-04 | |
(2 rows)
```

リカバリで incremental 方式が採用されている場合、PROJECTION_RECOVERIES システムテーブルに次の内容が表示されます。

```
=>SELECT * FROM PROJECTION_RECOVERIES;
node_name | projection_id | projection_name | transaction_id |
statement_id |
```

```
method | status | progress | detail | start_time | end_time |
runtime_priority
-----------+------------------+----------------+------------------+-
-------
------+-------------+----------+----------+-------------------------+-
-------
----------------------+----------+------------------
node02 | 45035996273824238 | public.t1_b1 | 49539595901075634 | 1 |
incremental
| running | 0 | Scan:0% Sort:0% Write:0% | 2018-04-22 21:42:07.151166-
04 | |
node02 | 45035996273824228 | public.t1_b0 | 49539595901075633 | 1 |
incremental
| running | 0 | Scan:0% Sort:0% Write:0% | 2018-04-22 21:42:07.151473-
04 | |
(2 rows)
```

incremental-replay-delete が適用されると、PROJECTION_RECOVERIES システムテーブルにより次の詳細を確認できます。

```
=>SELECT * FROM PROJECTION_RECOVERIES;
node_name | projection_id | projection_name | transaction_id |
statement_id |
method | status | progress | detail | start_time | end_time |
runtime_priority
-----------+------------------+----------------+------------------+-
-------
------+-------------+----------+----------+-------------------------+-
-------
----------------------+----------+------------------
node02 | 45035996273824238 | public.t1_b1 | 49539595901075634 | 1 |
incremental-replay-delete
| finished | | | 2018-04-22 21:42:07.151166-04 | 2018-04-22
21:42:11.976041-04 |
node02 | 45035996273824228 | public.t1_b0 | 49539595901075633 | 1 |
incremental-replay-delete
| running | 46 | Delete:0/4 | 2018-04-22 21:42:07.151473-04 | |
(2 rows)
```

ノードリカバリがダーティートランザクションのリカバリフェーズに入ると、TABLE_RECOVERIES システムテーブルは *phase* 列の変更を取得します。

```
=> SELECT * FROM TABLE_RECOVERIES;
node_name | table_oid | table_name | status | phase | thread_id |
start_time |
end_time | recover_priority | recover_error | is_historical
-----------+------------------+-----------+-----------+-------------
-----+-
-------------+-------------------------------+------------------------
-----+--
--------------------+--------------+--------------
node02 | 45035996273825192 | public.t1 | recovering | historical dirty
| 7f159f7fe700
| 2018-04-25 15:46:13.096937-04 | | -9223372036854775807 | | f
node02 | 45035996273825196 | public.t2 | recovered | | 7f16037fe700 | |
2018-04-25
15:46:13.09384-04 | -9223372036854775807 | |
(2 rows)
```

ノードリカバリが replay delete ステージに入ると、phase 列が TABLE_RECOVERIES システムテーブルの current replay delete となることが期待されます。

```
=>SELECT * FROM TABLE_RECOVERIES;
node_name | table_oid | table_name | status | phase | thread_id |
start_time |
end_time | recover_priority | recover_error | is_historical
-----------+------------------+-----------+-----------+-------------
-----+-
-------------+-------------------------------+------------------------
-----+--
--------------------+--------------+--------------
node02 | 45035996273826218 | public.t1 | recovering | current replay
delete |
7f60faffd700 | 2018-04-25 15:46:38.130421-04 | | -9223372036854775807 |
| f
(1 row)

=> SELECT * FROM PROJECTION_RECOVERIES;
node_name | projection_id | projection_name | transaction_id |
statement_id |
method | status | progress | detail | start_time | end_time |
runtime_priority
-----------+------------------+----------------+------------------+-
-------
```

```
------+--------------------------+----------+----------+-----------+-
-------
----------------------+------------------------------+--------------
---
node02 | 45035996273826230 | public.t1_b1 | 0 | | | queued | | | | |
node02 | 45035996273826230 | public.t1_b1 | 49539595901075718 | 1 |
incremental
| finished | | | 2018-04-25 15:46:38.142405-04 | 2018-04-25
15:46:38.168844-04 |
node02 | 45035996273826220 | public.t1_b0 | 49539595901075719 | 1 |
incremental
| finished | | | 2018-04-25 15:46:38.142025-04 | 2018-04-25
15:46:38.168814-04 |
node02 | 45035996273826220 | public.t1_b0 | 49539595901075725 | 2 |
incremental-replay-delete
| running | 20 | Delete:0/4 | 2018-04-25 15:46:38.285413-04 | |
(4 rows)
```

4. 場合によってはリカバリの失敗が発生します。リカバリの失敗の原因としては、DML または DDL の同時実行、ダーティートランザクションの停止、カタログイベントの適用の失敗、カタログイベントの欠落の検出などが原因で、replay delete ステージで T ロックを取得できないことがあります。ノード状態が RECOVERING または UP の場合、リカバリノードのリカバリ失敗を確認するには 2 つの方法があります。1 つは、TABLE_RECOVERIES システムテーブルの *recover_error 列*をチェックすることです。たとえば、TABLE_RECOVERIES に次のようにカタログイベントを適用できないことがあります。

```
=>SELECT * FROM TABLE_RECOVERIES;
node_name | table_oid | table_name | status | phase | thread_id |
start_time |
end_time | recover_priority | recover_error | is_historical
-----------+------------------+------------+------------+---------------
--+-
-------------+------------------------------+---------------------------
--+--
--------------------+---------------+--------------
node04 | 45035996273845482 | public.t2 | error-retry | | 7f3551ffb700 |
2018-02-21 10:40:17.887369-05
| 2018-02-21 10:57:52.290204-05 | -9223372036854775807 | Event apply
failed | t
node04 | 45035996273845460 | public.t1 | error-retry | | 7f35517fa700 |
2018-02-21 10:40:17.887322-05
| 2018-02-21 10:57:52.801538-05 | -9223372036854775807 | Event apply
failed | t
(2 rows)
```

リカバリ失敗を確認し、ノードをリカバリしようとするもう 1 つの方法は、vertica.log ファイルで次のキーワードを指定して grep コマンドを実行することです。

```
$ grep “incrCatchUpFailureCount” vertica.log
```

リカバリ失敗のためにノードがシャットダウンされた場合、リカバリの失敗原因を調査するために 2 番目の方法が役に立つでしょう。

ノードリカバリのトラブルシューティング

以下のリストは、最も一般的なトラブルシューティングのユースケースと、問題を解決するための推奨事項を示しています。

1. **事象**: Spread デーモンが起動せず、Vertica に接続できません。

   概要: Spread デーモンの起動に失敗した場合、ノードのリカバリは失敗し、リカバリノードは引き続き DOWN 状態のままです。

   アクション: 問題を特定するには、リカバリノードの startup.log ファイルまたは vertica.log ファイルを確認します。次のような解決策が考えられます。

   - リカバリノードのカタログディレクトリ内の spread.conf ファイルが、クラスター内の他のノード上のすべての spread.conf ファイルと同一であることを確認します。
   - リカバリノードの IP アドレスが、カタログディレクトリの spread.conf ファイル内で適切な状態であることを確認します。
   - テンポラリディレクトリ（/tmp）からファイル 4803 を削除し、ノードを再起動します。

2. **事象**: リカバリノードは、リカバリ前フェーズの「check data strorage（データストレージのチェック）」ステージで異常に長い時間を費やしています。

   概要: リカバリノードは、次のような場合において、リカバリ前のフェーズで異常に長い時間を費やす可能性があります。

   - カタログサイズが大きく、数百万の ROS ファイルがあります。
   - リカバリノードは、カタログから参照されていない多数の ROS ファイルを除去する必要があります。

   アクション: startup.log ファイルを使用して、リカバリノードの進行状況をモニタリングします。

3. **事象**: リカバリノードがエラーで失敗する。

   概要: エラーメッセージに、「 Data consistency problems found; startup aborted（データー貫性の問題が見つかったため、スタートアップが中止されました。）」といった内容が表示される。

アクション: この問題を解決するには、admintools の CLI から--force フラグを使用してノードを再起動します。

```
$ /opt/vertica/bin/admintools -t restart_node -d <database_name> --hosts
<ip address> --force
```

4. **事象**: リカバリノードは、クラスターへの招待を待っています。

   概要: リカバリノードはクラスター内の他のノードと通信できず、次の状況で他のノードからクラスタへの招待を待っています。

   - ファイアウォールが有効になっている、あるいは、ポートがブロックされています。
   - リカバリノードが異なるサブネットに属します。

   アクション: これらの問題を解決するには、次のタスクを実行します。

   - クラスター内の他のノードとの通信を確認するには、netcat（$ nc）を使用します。
   - リカバリノードがクラスター内の他のノードと異なるサブネットに属している場合、point-to-point モードで spread を設定します。

5. **事象**: カタログイベントが原因でテーブルリカバリが複数回失敗します。

   概要: 現在リカバリ中のテーブルで TRUNCATE TABLE または DROP PARTITION などのカタログイベントを実行すると、テーブルリカバリに失敗する可能性があります。
   20 回の試行後にテーブルのリカバリが失敗した場合、Vertica のリカバリが失敗します。複数のリカバリの試行が失敗すると、ノードリカバリが遅延することがあります。

   アクション: この問題を解決するには、次のタスクを実行します。

   - リカバリの試行を追跡するには、次のコマンドを使用します。

```
$ grep “incrCatchUpFailureCount” vertica.log
```

   例えば、次のメッセージが示すように、Vertica はノードのリカバリを 17 回試行しました。

```
[Recover] <INFO> incrCatchUpFailureCount: 17 failures, max 20
```

   - 上記のステップから複数回の再試行を検出した場合、カタログイベントを多く実行する ETL プロセスを停止します。

6. **事象**: UPDATE および DELETE ステートメントが、実行されるのに異常に長い時間がかかっています。

概要: リカバリノードがリカバリ中のテーブルで T ロックを 5 分間以上取得できない場合、テーブルリカバリが失敗します。LOCKTIMEOUT 構成パラメーターのデフォルト設定は 5 分です。
20 回の試行後にテーブルがリカバリしない場合、Vertica のリカバリが失敗する。複数回リカバリの試行が失敗した場合、ノードリカバリを遅延させることがある。

アクション: この問題を解決するには、次のタスクを実行します。

- リカバリの試行を追跡するには、次のコマンドを使用します。

```
$ grep “incrCatchUpFailureCount” vertica.log
```

- 長時間実行されている DML 処理をキャンセルします。

Vertica 7.2.1 を使用している場合、Vertica 7.2.2 以降にアップグレードします。これにより、複数のテーブルを並列でリカバリ可能です。詳細については、Vertica documentation の Recovery by Table を参照してください。

詳細情報

詳細情報	...See
Vertica Community Edition	https://my.vertica.com/community/
Vertica Documentation	http://my.vertica.com/docs/latest/HTML/index.htm
Big Data and Analytics Community	https://forum.vertica.com/