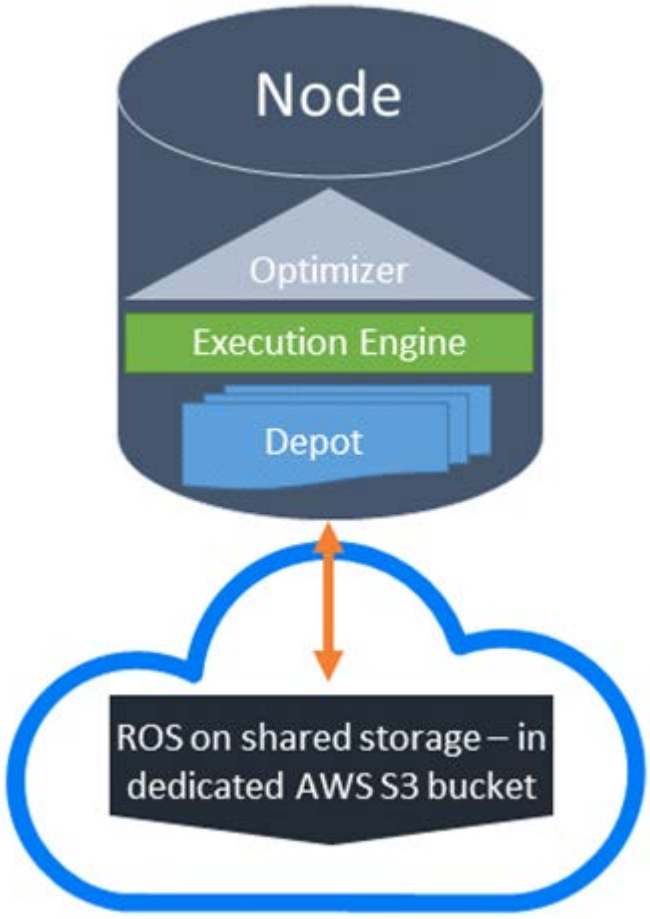
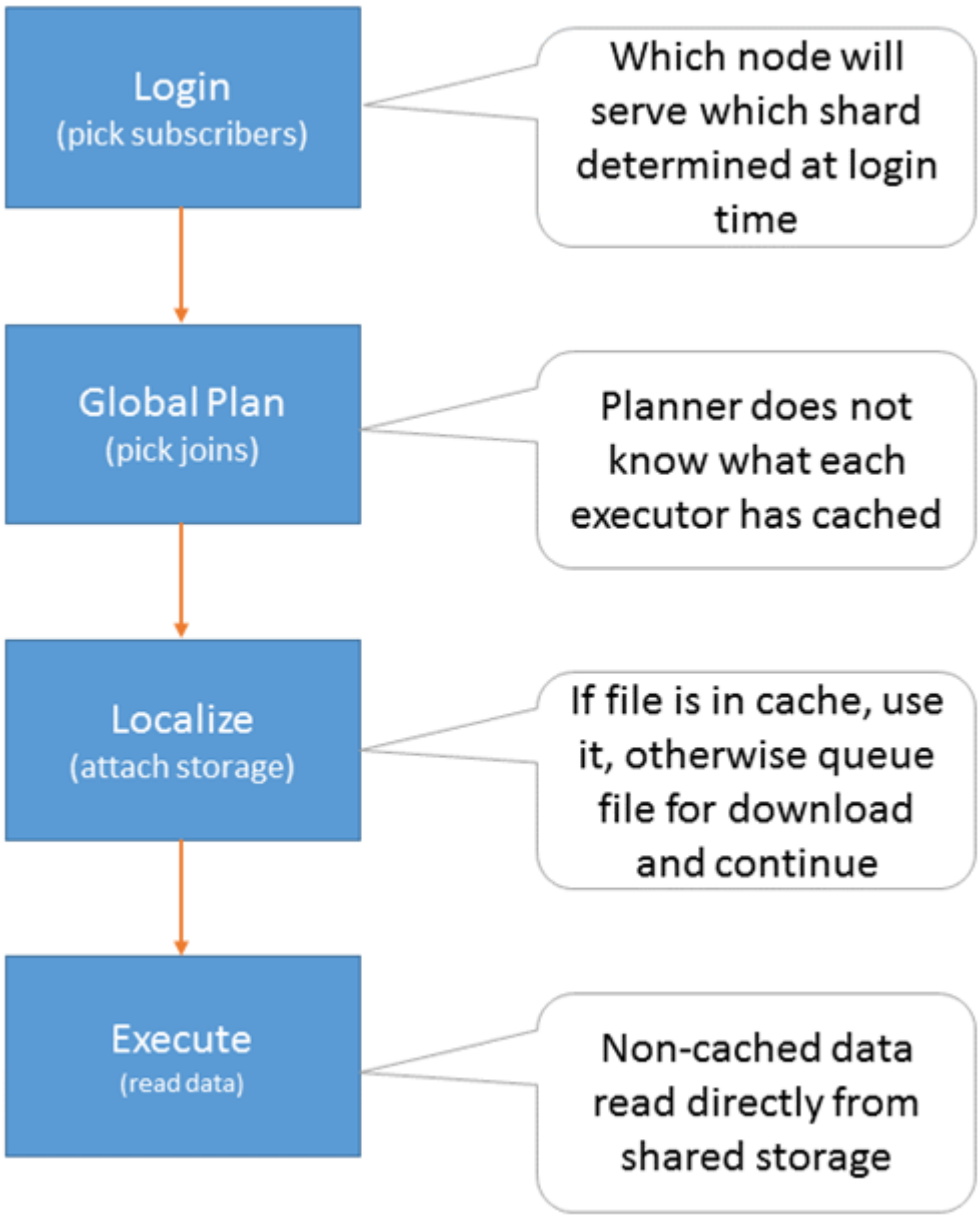
Vertica Eon モード: キャッシング

原文はこちら

共有ストレージ内のデータに対してすべてのクエリを直接実行すると、パフォーマンスが低下し、共有ストレージの負荷が大きくなります。Vertica の Eon モードでは、デポと呼ばれるキャッシュが導入され、頻繁に使用されるデータの共有ストレージからの読み込みを回避します（下図参照）。



デポは、共有ストレージからのデータファイル全体をキャッシュするためのディスクキャッシュです。Vertica は、一度書き込まれるとストレージファイルを変更することはないため、キャッシュは追加と削除を処理するだけで、決して無効化する必要はありません。キャッシュエビクションポリシーは、過去のアクセスが将来の必要性の良い予測値であると仮定して、単純な最小使用頻度（LRU）メカニズムとなっています。キャッシュはライトスルーです。新しく追加されたファイルはクエリによって参照される可能性があります。ファイル圧縮メカニズム（mergeout）は、その出力ファイルをキャッシュに入れます。



使用法

実際にキャッシュがどのように使用されているかをよりよく理解するための例を見てみましょう。4 ノード、4 シャードデータベースを考えてみましょう。データベースには、各ノード上に共有ロケーションとローカルデポがあります。storage_locations テーブルには、次のような詳細情報があります。共用ストレージはいずれのノードにも割り当てられておらず、デポのロケーションには最大サイズが割り当てられていることに注意してください。

```
=> SELECT node_name, location_path, location_usage, sharing_type, max_size
FROM storage_locations;
 node_name |              location_path              | location_usage |
sharing_type |  max_size
```

```
-----------+----------------------------------------+---------------+-------
-------+-------------
 node01    | /scratch_b/DBD/node01_data             | DATA,TEMP      | NONE
|           0
           | s3://nimbusdb/fxing/dbd                | DATA           |
COMMUNAL     |           0
 node02    | /scratch_b/DBD/node02_data             | DATA,TEMP      | NONE
|           0
 node03    | /scratch_b/DBD/node03_data             | DATA,TEMP      | NONE
|           0
 node04    | /scratch_b/DBD/node04_data             | DATA,TEMP      | NONE
|           0
 node01    | /scratch_b/qa/vertica/DBD/node01_depot | DEPOT          | NONE
| 64424509440
 node02    | /scratch_b/qa/vertica/DBD/node02_depot | DEPOT          | NONE
| 64424509440
 node03    | /scratch_b/qa/vertica/DBD/node03_depot | DEPOT          | NONE
| 64424509440
 node04    | /scratch_b/qa/vertica/DBD/node04_depot | DEPOT          | NONE
| 64424509440
(9 rows)
```

テーブルが作成され、データがロードされると、データは共用ストレージとデポの両方にコピーされます。デポ内のファイルはシステムテーブル vs_depot_lru から確認でき、S3 上のファイルは S3 バケットから見ることができます。

```
=> SELECT node_name,storage_type,size FROM vs_depot_lru WHERE
storage_type='ROS_BUNDLE';
 node_name | storage_type | size
-----------+--------------+------
 node01    | ROS_BUNDLE   |  535
 node01    | ROS_BUNDLE   |  561
 node04    | ROS_BUNDLE   |  561
 node04    | ROS_BUNDLE   |  562
 node02    | ROS_BUNDLE   |  570
 node02    | ROS_BUNDLE   |  535
 node03    | ROS_BUNDLE   |  562
 node03    | ROS_BUNDLE   |  570
(8 rows)
```

S3 バケットにはデータファイルも表示されます。

```
[14:30:55][release@ip-10-0-19-189: ~ ]$ aws s3 ls --recursive
s3://nimbusdb/fxing/dbd/
2018-02-21 13:58:03        562
fxing/dbd/4ec/0232fd1fcb77a1d9b750534797ef78f900d000000001ce11_0.gt
2018-02-21 13:58:03        561
fxing/dbd/572/02f291228ff4ee4eb27c4da4069e9e3e00a000000001ce11_0.gt
2018-02-21 13:58:03        570
fxing/dbd/a3a/0238836c1100ea08a7b87d089c6e354800c000000001ce11_0.gt
2018-02-21 13:58:03        535
fxing/dbd/d91/02bcb3da2b7d42fb5f91113100230c8200b000000001ce11_0.gt
```

デポの状態をモニターするのに役立つさまざまなシステムテーブルがあります。vs_depot_lru テーブルは、現在デポにあるファイルを示し、depot_uploads テーブルは共有ロケーションにアップロードされたファイルを示します。上記のステートメントに対して実行されたデータロードの場合、depot_uploads テーブルは次の状態を示します。

```
=> SELECT node_name, file_size_bytes, source_file,destination_file FROM
depot_uploads;
-[ RECORD 1 ]----+-----------------------------------------------------------
-------------------------------------
node_name        | node01
file_size_bytes  | 561
source_file      |
/scratch_b/qa/vertica/DBD/node01_depot/249/02f291228ff4ee4eb27c4da4069e9e3e00
a000000001ce11_0.gt
destination_file |
s3://nimbusdb/fxing/dbd/572/02f291228ff4ee4eb27c4da4069e9e3e00a000000001ce11_
0.gt
-[ RECORD 2 ]----+-----------------------------------------------------------
-------------------------------------
node_name        | node02
file_size_bytes  | 535
source_file      |
/scratch_b/qa/vertica/DBD/node02_depot/745/02bcb3da2b7d42fb5f91113100230c8200
b000000001ce11_0.gt
destination_file |
s3://nimbusdb/fxing/dbd/d91/02bcb3da2b7d42fb5f91113100230c8200b000000001ce11_
0.gt
-[ RECORD 3 ]----+-----------------------------------------------------------
-------------------------------------
node_name        | node04
file_size_bytes  | 562
```

```
source_file      |
/scratch_b/qa/vertica/DBD/node04_depot/737/0232fd1fcb77a1d9b750534797ef78f900
d000000001ce11_0.gt
destination_file |
s3://nimbusdb/fxing/dbd/4ec/0232fd1fcb77a1d9b750534797ef78f900d000000001ce11_
0.gt
-[ RECORD 4 ]----+-----------------------------------------------------------
-------------------------------------
node_name        | node03
file_size_bytes  | 570
source_file      |
/scratch_b/qa/vertica/DBD/node03_depot/241/0238836c1100ea08a7b87d089c6e354800
c000000001ce11_0.gt
destination_file |
s3://nimbusdb/fxing/dbd/a3a/0238836c1100ea08a7b87d089c6e354800c000000001ce11_
0.gt
```

全体のサイズの統計情報は、vs_depot_size テーブルに記録されます。

```
=> SELECTnode_name, max_size_in_bytes, current_size_in_bytes FROM
vs_depot_size;
 node_name | max_size_in_bytes | current_size_in_bytes
-----------+-------------------+-----------------------
 node01    |       64424509440 |                  1096
 node02    |       64424509440 |                  1105
 node03    |       64424509440 |                  1132
 node04    |       64424509440 |                  1123
```

クエリが実行されると、デポにあるデータはそこから読み取られ、残りのデータは共有のロケーションから直接読み取られます。パフォーマンスの問題を診断するために、dc_file_reads テーブルを使用して、特定のファイルの読み取り元を確認できます。

```
=> SELECT node_name, storageid, storage_path FROM dc_file_reads;
 node_name |                   storageid                    |
storage_path
-----------+------------------------------------------------+-------------
-------------------------
 node03    | 0238836c1100ea08a7b87d089c6e354800c000000001ce11 |
/scratch_b/qa/vertica/DBD/node03_depot
 node01    | 02f291228ff4ee4eb27c4da4069e9e3e00a000000001ce11 |
/scratch_b/qa/vertica/DBD/node01_depot
 node02    | 02bcb3da2b7d42fb5f91113100230c8200b000000001ce11 |
/scratch_b/qa/vertica/DBD/node02_depot
```

```
 node04    | 0232fd1fcb77a1d9b750534797ef78f900d000000001ce11 |
/scratch_b/qa/vertica/DBD/node04_depot
(4 rows)
```

データデポを消去してから再度クエリを実行すると、dc_file_reads テーブルには、ファイルが S3 から直接読み取られ、depot_fetches テーブルに示されているように、その後の読み取りのためにデポにフェッチされたことが示されます。

```
=> SELECT node_name, storageid, storage_path FROM dc_file_reads;
 node_name |                   storageid                    |
storage_path
-----------+------------------------------------------------+--------------
-------------------------
 node01    | 02f291228ff4ee4eb27c4da4069e9e3e00a000000001d05f |
s3://nimbusdb/fxing/dbd
 node03    | 0238836c1100ea08a7b87d089c6e354800c000000001d05f |
s3://nimbusdb/fxing/dbd
 node04    | 0232fd1fcb77a1d9b750534797ef78f900d000000001d05f |
s3://nimbusdb/fxing/dbd
 node02    | 02bcb3da2b7d42fb5f91113100230c8200b000000001d05f |
s3://nimbusdb/fxing/dbd
(4 rows)

=> SELECT * FROM depot_fetches;
-[ RECORD 1 ]----+-------------------------------------------------------------
-----------------------------------
node_name        | node04
storage_id       | 0232fd1fcb77a1d9b750534797ef78f900d000000001d05f
file_size_bytes  | 562
source_file      |
s3://nimbusdb/fxing/dbd/1c4/0232fd1fcb77a1d9b750534797ef78f900d000000001d05f_
0.gt
destination_file |
/scratch_b/qa/vertica/DBD/node04_depot/327/0232fd1fcb77a1d9b750534797ef78f900
d000000001d05f_0.gt
-[ RECORD 2 ]----+-------------------------------------------------------------
-----------------------------------
node_name        | node03
storage_id       | 0238836c1100ea08a7b87d089c6e354800c000000001d05f
file_size_bytes  | 570
source_file      |
s3://nimbusdb/fxing/dbd/245/0238836c1100ea08a7b87d089c6e354800c000000001d05f_
0.gt
```

```
destination_file |
/scratch_b/qa/vertica/DBD/node03_depot/831/0238836c1100ea08a7b87d089c6e354800
c000000001d05f_0.gt
reason           |
-[ RECORD 3 ]----+-------------------------------------------------------------
-----------------------------------
node_name        | node02
storage_id       | 02bcb3da2b7d42fb5f91113100230c8200b000000001d05f
file_size_bytes  | 535
source_file      |
s3://nimbusdb/fxing/dbd/ced/02bcb3da2b7d42fb5f91113100230c8200b000000001d05f_
0.gt
destination_file |
/scratch_b/qa/vertica/DBD/node02_depot/335/02bcb3da2b7d42fb5f91113100230c8200
b000000001d05f_0.gt
-[ RECORD 4 ]----+-------------------------------------------------------------
-----------------------------------
node_name        | node01
storage_id       | 02f291228ff4ee4eb27c4da4069e9e3e00a000000001d05f
file_size_bytes  | 561
source_file      |
s3://nimbusdb/fxing/dbd/572/02f291228ff4ee4eb27c4da4069e9e3e00a000000001ce11_
0.gt
destination_file |
/scratch_b/qa/vertica/DBD/node01_depot/249/02f291228ff4ee4eb27c4da4069e9e3e00
a000000001ce11_0.gt
```

メタ関数と構成パラメーター

デポには、ケースバイケースで動作を調整するためのさまざまなメタ関数と構成パラメーターがあります。

メタ関数:

1. Clear_data_depot (): されたテーブルのデポからすべてのデータを削除します。テーブル名が指定されていない場合、すべてのデータを削除します。
2. Alter_location_size(, ,) : ワークロードが変更された場合、この機能を使用して、デポのサイズを変更する必要があります。副作用として、新しいサイズが小さい場合、最近使われていないファイルが削除されます。
3. Finish_fetching_files(): クエリを実行しているときに、デポに含まれていないファイルがあると、それらは非同期サービスによってフェッチされるようにキューに入れられます。この関数は、キュー内のすべてのファイルがダウンロードされるのを待ちます。

構成パラメーター:

1. UseDepotForReads : ファイルがデポにあり、そこから読み取られているかどうか常にチェックします。
2. UseDepotForWrites : ロード中にデポに書き込み、ファイルを共有ストレージにアップロードします。
3. UsePeerToPeerDataTransfer: データロード時に、すべてのサブスクライバーにシャードのデータを送信します。

パフォーマンス

デポの目的は、共用ストレージのためにパフォーマンスが悪影響を受けないようにすることです。デポの利点を確認するためにいくつかのパフォーマンステストを行いました。この実験では、通常の TPCH クエリでは、コールドデポに対してウォームデポに対して実行すると、3.5 倍の速さで実行されることが示されています。次のハードウェア仕様のものを使用して実験を実施しました。

	カタログ	データ	一時領域	デポ
TPCH_NIMBUS	EBS – シングルボリューム	S3	インスタンスストア	インスタンスストア（60GB/ノード）