COPY を使用した Vertica でのデータロード

July, 2018

原文はこちら

MICRO FOCUS

目次

COPY 文によるバルクロード

COPY 文は、大量のデータを Vertica データベースにロードする最も効率的な方法です。COPY コマンドを使用して、1 つ以上のファイルをクラスターのホストにコピーできます。バルクロードの場合、最も有用な COPY コマンドは次のとおりです。

- **COPY LOCAL**: ローカルのクライアントシステムから、サーバーがファイルを処理する Vertica ホストに、データファイルまたは指定されたすべてのファイルをロードします。
- **Vertica クラスター上にソースデータを COPY**: Vertica クラスター上で、データファイルまたは指定されたすべてのファイルを、JSON や CSV などのさまざまなソースから、Vertica の内部フォーマットにロードします。
- **カスタムのソース、フィルター、パーサーを使ってユーザー定義ロード(UDL)関数で COPY**: カスタムのユーザー定義のソース、パーサー、およびフィルターから、データのロード設定を制御して、データファイルまたは指定されたファイルをロードします。

すべてのタイプの COPY 文は同じ方法論とプロセスを共有しますが、全てにおいて異なる制限事項があります。相違にかかわらず、COPY 文には常に 2 つのフェーズがあります。

- フェーズ I (イニシエーター)は、ファイルをロードして解析し、ファイルを他のノードに配布します。
- フェーズ II (エグゼキューター)は、すべてのノード上でデータを処理します。

COPY では、多くの実行エンジンオペレータをバルクロードに使用できます。1 つ以上のファイルをロードする実行エンジンオペレータの中には、Load、Parse、Load Union、Segment、Sort/Merge、および、DataTarget があります。

COPY 文は、ターゲットのテーブルが分散化されている場合、各プロジェクションごとにセグメントを作成します。*Segmentation*(セグメンテーション)は、クエリパフォーマンスと高速データ purge のためにクラスターノード間でデータがどのように分散されるかを定義します。

COPY 文を使ったデータのロード方法

1 つ以上のファイルをロードする COPY 文のワークフローは、次の 2 つのフェーズで実行されます。

フェーズ I

1. *Load*(ロード)オペレータは、データを含むソースファイルをデータベースにロードします。*Parse*(パース)オペレータは、データベース内のロードされたデータを解析します。

2. *Load Union*(ロードユニオン) オペレータは、データを分散化する前に、解析されたデータを 1 つのコンテナにマージします。オペレータは複数のファイルをロードするときにアクティブになり、1 つのファイルをロードするときには非アクティブになります。

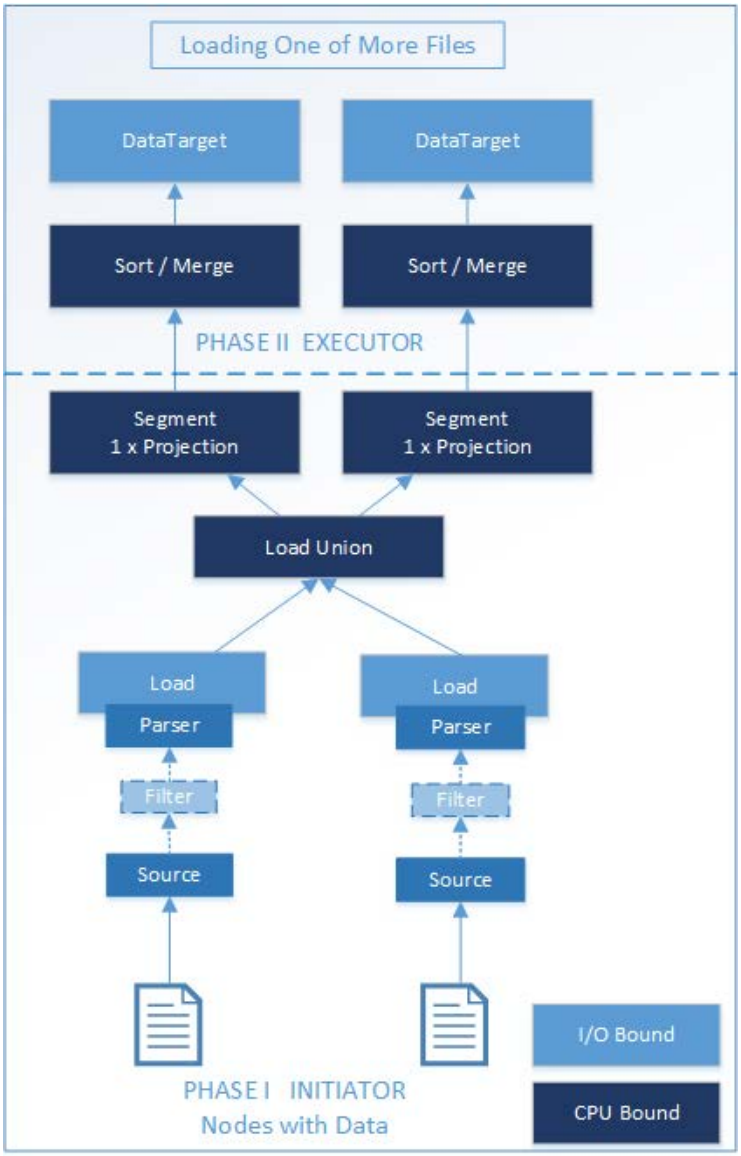
3. *Segment*(セグメント)オペレータは、解析されたデータをデータのサイズに応じて 1 つ以上のプロジェクションに分割します。加えて、テーブルのパーティション化により各ノード上のデータが分離され、複数のデータベースノード間で均等にデータが分散されます。これにより、すべてのノードがクエリの実行に関与するようになります。

フェーズ II

1. *Sort*（ソート）オペレータは、分散化されたデータとプロジェクションをソートします。*Merge*（マージ）オペレータは、ソートされたデータを適切にマージします。ソートオペレータとマージオペレータは、集計されたデータに対して機能します。

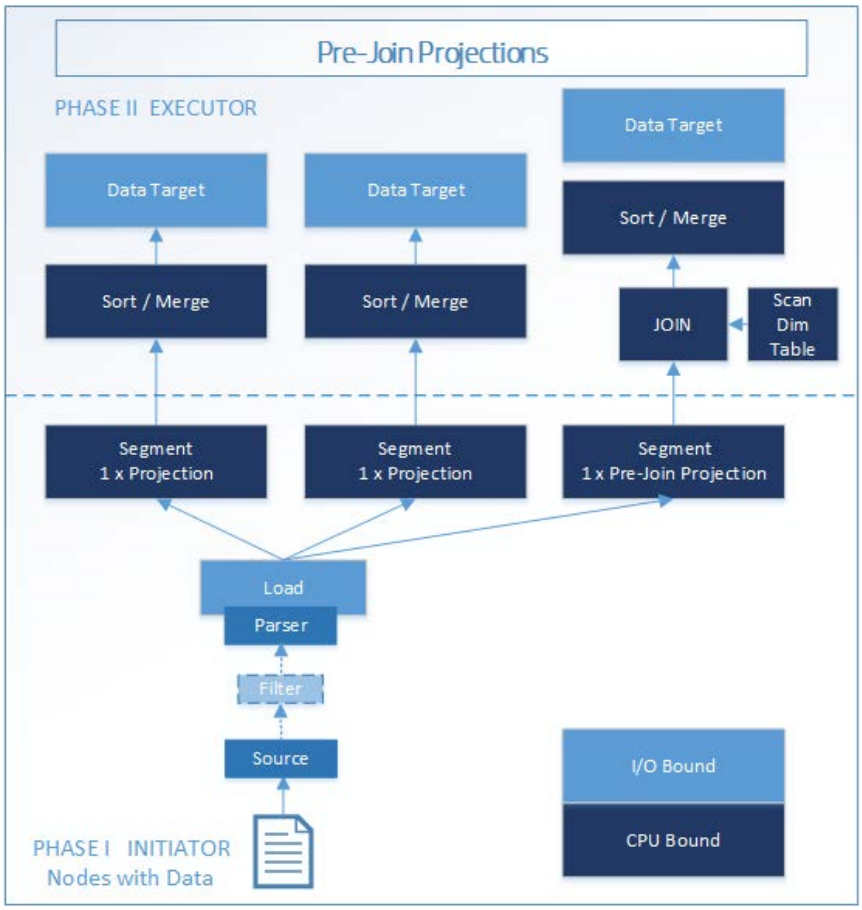
2. *DataTarget*（データターゲット）オペレータは、ディスク上のデータをコピーします。

次の図は、1 つまたは複数のファイルを 2 つのフェーズでロードするワークロードを示しています。ライトブルーとダークブルーのボックスは、実行エンジンのオペレータを表します。

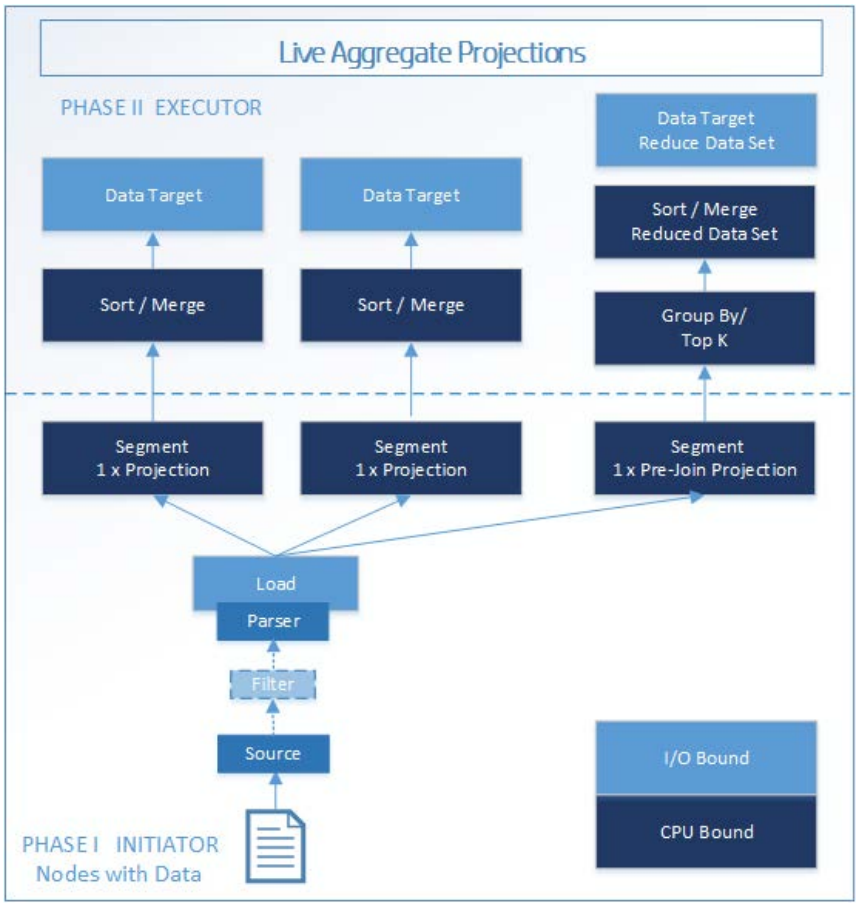


Vertica 8.0 では、Apportioned Load において、すべてのノードがソースデータにアクセスできる場合、フェーズ I は複数のノードで発生します。Apportioned Load は分割可能なロードであり、1 つのデータファイルを 1 つ以上のノード上で実行することが可能となりますビルトインの delimited パーサーは、ソースが複数のノード上で利用可能で分割可能な場合、自動的にロードを分割できます。Apportioned Load が利用できない場合、フェーズ I はファイルを読み込んだノードでのみで発生します。

フェーズ II では、追加の実行エンジンオペレータを使用して、プリジョインプロジェクションとライブアグリゲートプロジェクションに対する処理を行います。次の図のプリジョインプロジェクションでは、ディメンションテーブルに対して追加の実行エンジンオペレータである JOIN と SCAN が示されています。次の図のライブアグリゲートプロジェクションでは、追加の GROUP BY/Top-K 実行エンジンオペレータが示されています。



プリジョインプロジェクションでは、ディメンションテーブルに対して追加の実行エンジンオペレータである JOIN および SCAN が追加されます。



ライブアグリゲートプロジェクションでは、GROUP BY/Top-K 実行エンジンのオペレータが追加されます。

ロードの方法

ロードするデータによって、COPY 文にはいくつかのロードの方法があります。3 つのロード方法から選択できます。

- COPY AUTO
- COPY DIRECT
- COPY TRICKLE

COPY AUTO を使う場合

COPY は AUTO 方式を使用して WOS にデータをロードします。より小さいバルクロードに対しては、このデフォルトの AUTO ロード方式を使用してください。AUTO オプションは、ファイルのサイズを特定できない場合に最も便利です。WOS がいっぱいになると、COPY はディスク上の ROS コンテナに直接ロードし続けます。ROS データはソートされてエンコードされています。

COPY DIRECT を使う場合

COPY は DIRECT 方式を使用してデータを ROS コンテナに直接ロードします。大きなバルクロード（100 MB 以上）には、DIRECT ロード方式を使用してください。DIRECT 方式は、WOS を回避し、データを ROS コンテナにロードすることによって、大きなファイルのパフォーマンスを向上させます。多くの小さなデータセットをロードするために DIRECT を使用すると、多くの ROS コンテナが生成され、ロード後に統合される必要がでてきます。

COPY TRICKLE を使う場合

COPY では、TRICKLE 方式を使用してデータを WOS に直接ロードします。初期バルクロードを完了した後、データを追加でロードする場合に、TRICKLE ロード方式を使用してください。WOS がいっぱいになるとエラーが発生し、データのロード全体がロールバックされます。この方法は、サイトでしっかりと調整されたロードおよび moveout 処理が想定されていて、かつ、ロード中のデータを WOS が保持できると確信している場合にのみ使用してください。このオプションは、パーティション化されたテーブルにデータをロードするときに、AUTO よりも効率的です。

Vertica 上のデータロードをモニタリングするシステムテーブル

Vertica は、データベースのロードをモニタリングするためのシステムテーブルを提供します。

- **LOAD_STREAMS**: 各ノード上のロード処理の実行中のものと過去実行されたもののロードメトリックをモニタリングし、ロードおよびリジェクトされたレコードに関する統計を提供します。
- **DC_LOAD_EVENTS**: ロード解析中の重要なシステムイベントに関する情報を格納します。
  - Batchbegin
  - Sourcebegin
  - Parsebegin
  - Parsedone
  - Sourcedone
  - Batchdone

データロードのチューニング

リソースプールのパラメーターと構成パラメーターは、データロードのパフォーマンスに影響します。

リソースプールパラメーター

次のパラメーターは、データベース管理者がデータをロードするためのリソースを管理するのに役立つリソースプールの設定項目を示します。

パラメーター	説明
PLANNEDCONCURRENCY	COPY コマンドごとに割り当てられるメモリ量を定義します。リソースプール内で同時に実行されるクエリの想定される数を表します。
MAXCONCURRENCY	同時COPYジョブの数を制限し、リソースプールで使用可能な同時実行スロットの最大数を表します。
EXECUTIONPARALLELISM	このリソースプールで発行され、ロードに割り当てられた単一のクエリを処理するために使用されるスレッドの数を制限します。Verticaは、コア数、使用可能なメモリ、およびシステム内のデータ量に基づいてこの値を設定します。メモリが制限されていないか、あるいは、データ量が非常に少ない場合を除き、Verticaはこの値をノード上のコア数に設定します。

クエリバジェット

RESOURCE_POOL_STATUS システムテーブルの query_budget_kb 列には、関連するリソースプールで実行されたクエリのターゲットメモリが表示されます。

クエリの budget_kb を確認するには、次のコマンドを使用します。

```
=> SELECT pool_name, query_budget_kb FROM resource_pool_status;
```

クエリの budget_kb を変更する前に、次のメモリの考慮事項に注意してください。

- MEMORYSIZE > 0 および MAXMEMORYSIZE が設定なし、または、MEMORYSIZE に等しい場合、クエリバジェット = MEMORYSIZE / Planned Concurrency
- MEMORYSIZE = 0 および MAXMEMORYSIZE > 0 の場合、クエリバジェット =（MAXMEMORYSIZE * 0.95）/ Planned Concurrency
- MEMORYSIZE = 0 および MAXMEMORYSIZE が設定なしの場合、クエリバジェット = [（General Pool * 0.95）-（sum（他のプールの MEMORYSIZE）] / Planned Concurrency

リソースプールのパラメーターを変更する方法

リソースプールのパラメーターを変更するには、次のコマンドを使用します。

```
=> ALTER RESOURCE POOL <pool_name> <parameter> <new_value>;
```

データロードの構成パラメーター

次の構成パラメーターは、データロードのパフォーマンスを向上させるのに役立ちます。

パラメーター	説明
EnableCooperativeParse	ノード上でマルチスレッドの Cooperative パースの機能を実装します。区切られたデータのロードと固定長のデータのロードの両方にこのパラメーターを使用できます。Cooperative パースの並列化は、ソースデータのノードに対してローカルです。
SortWorkerThreads	ソートのワーカースレッドの数を制御します。0に設定すると、バックグラウンドスレッドが無効になります。ボトルネックがロードの解析／ソートフェーズにあるときのロードパフォーマンスを向上させます。
ReuseDataConnections	クエリの実行中にTCP接続を再利用しようとします。
DataBufferDepth	データ接続に割り当てるバッファを制御します。
CompressNetworkData	データトラフィックを圧縮し、データ帯域幅を縮小します。
EnableApportionLoad	ロード用の配分可能なソース／パーサーを定義し、データを適切なポーションに分割します。Vertica 8.0では、Apportioned LoadがFilePortionSourceソース関数で機能します。
MultiLevelNetworkRoutingFactor	大規模クラスターのネットワークルーティングを定義し、カウントの減少要因を調整します。

ロードシナリオにおけるトラブルシューティング

データベースにデータをロードする際に問題を解決できない場合、Vertica の製品サポートまでお問い合わせください。

大きいファイルのロード

ユースケース: 大きなファイルをロードするのに時間がかかります。

推奨事項: 2 つの方法のいずれかで各ノード上でロードを並列に実行します。

- EnableApportionLoad パラメーターを使用して、クラスター内の異なるノード間でワークロードを並列化します。Apportioned Load の場合、ファイルをロードするノード間でファイルを共有し、次のステートメントを使用して FilePortionSource Source UDx パラメーターをインストールします。

```
=> COPY copy_test.store_sales_fact with source
FilePortionSource(file='/data/test_copy/source_data5/
Store_Sales_Fact.tbl',nodes='v_vdb_node0001,v_vdb_node0002,v_vdb_node00
03')direct;
```

- すべてのノードが任意のノードのファイルにアクセスできるように、NFS マウントポイント内のファイルを分割して公開します。

どちらのオプションも同様のローディングパフォーマンスを得ることができます。ただし、2 番目のオプションでは、手動でファイルを分割する必要があります。

複数の小さなファイルを同じターゲットテーブルにロード

ユースケース: COPY DIRECT を使用して複数の小さなファイルをロードすると、パフォーマンスが低下します。小さなファイルを複数の COPY 文でロードすると、複数の ROS コンテナを生成します。多数の ROS コンテナが Vertica のパフォーマンスに影響し、ロード完了後に Tuple Mover の追加作業が必要となります。

推奨事項: Vertica は、COPY 文を使用して複数の小さなファイルをロードするために次のオプションを推奨しています。

- ロードするファイルを結合し、COPY 文の数を制御します。COPY 文を少なくすると、トランザクションの数が削減され、1 つのトランザクションでより多くのデータがロードされます。
- Linux のパイプを使用してロードするファイルを結合します。
- パフォーマンスを向上させるために、同じ COPY 文でファイルを結合します。

ワイドテーブルへのロード

ユースケース: 大きな VARCHAR 列を持つワイドテーブルは、COPY コマンドのフェーズ II でのワークフローのボトルネックとなります。

推奨事項: Vertica は、ワイドテーブルをロードするために次のオプションを推奨します。

- 特定のロードを例外として LoadMergeChunkSizeK パラメーターを変更します。
- ワイドテーブルと複数の小さなテーブルにフレックステーブルを使用する。ワイドテーブルをフレックステーブルにロードするには、多くのフィールドの代わりに 1 つのフィールドをロードする必要があります。したがって、カタログのサイズが縮小され、データベース全体のパフォーマンスが向上します。初期ロードは非常に高速で、ユーザーがデータをすばやく利用できます。ただし、クエリのパフォーマンスは、列指向のストレージに比べて低くなります。

- GROUPED 相関列を使用してワイドテーブルをロードします。GROUPED 句は、2 つ以上の列を 1 つのディスクファイルにグループ化します。1 つの列の値が他の列の値に関連する場合、2 つの列が相関します。

リソースを追加したり、ファイルを分割したり、ノード間で作業を並列化したりすることで、この問題を解決することができない場合、Vertica 製品サポートに連絡して、そのガイダンスの下で構成パラメーターを調整する必要があります。

ロードの実行ノード

ユースケース: 実行ノードは計算目的でのみ予約されており、データセグメントは含まれていません。専用ノードは全ての CPU を使用します。2 つのユースケースがこの問題に関連しています。

- 大きいファイルを 24 時間以内にロードする必要があります。
- 解析処理は、ワークフローをブロックします。

このようなユースケースでは、通常、次の条件が存在します。

- ワイドテーブルは、1 行あたりのサイズが大きいです。
- GZIP ファイルは、ネットワーク転送時間を短縮するために圧縮されています。ファイルが Vertica のローカルストレージに置かれると、COPY コマンドは CPU 使用率を増加させるデータを非圧縮します。
- ワイドなセグメンテーションキーを使用するテーブルでは、より多くの CPU が使用されます。

推奨事項: Vertica 製品サポートとの協議の後、実行ノードの使用を推奨しています。リソースプールの設定により、実行およびデータノード上のリソースが適切に使用されます。リソースプールのパラメーターはクラスター全体に適用されるため、Vertica 製品サポートにより効率的なモデルのパラメーターが提供されます。実行ノードを使ってロードする代わりに、より多くの CPU を持つサーバーを使用し、排他的に CPU をロード用のリソースプールに割り当て、ロード処理専用にリソースを制限します。