PostgreSQL

Enterprise applications

NetApp
May 09, 2024

目次

PostgreSQL . . . 1
ONTAP上のPostgreSQLデータベース . . . 1
データベース設定 . . . 1
ストレージ構成 . . . 5
データ保護 . . . 9

PostgreSQL

ONTAP上のPostgreSQLデータベース

PostgreSQLには、PostgreSQL、PostgreSQL Plus、EDB Postgres Advanced Server（EPAS）などのバリアントが付属しています。PostgreSQLは通常、多層アプリケーションのバックエンドデータベースとして導入されます。一般的なミドルウェアパッケージ(PHP、Java、Python、Tcl/Tk、ODBCなど)でサポートされています。 とJDBC）は、オープンソースのデータベース管理システムでは、歴史的に人気のある選択肢でした。ONTAPは、信頼性、パフォーマンス、効率に優れたデータ管理機能を備えたPostgreSQLデータベースを実行するための優れた選択肢です。

ONTAPおよびPostgreSQLデータベースに関するこのドキュメントは、以前に公開されていた_TR-4770：『PostgreSQL database on ONTAP best practices』に代わるものです。_

データが指数関数的に増加するにつれて、企業にとってデータ管理はより複雑になります。この複雑さにより、ライセンス、運用、サポート、メンテナンスのコストが増大します。全体的なTCOを削減するには、信頼性とパフォーマンスに優れたバックエンドストレージを使用して、商用データベースからオープンソースデータベースに切り替えることを検討してください。

ONTAPは理想的なプラットフォームです。ONTAPは文字通りデータベース向けに設計されているからです。データベースワークロードのニーズに対応するために、高度なQuality of Service（QoS；サービス品質）機能や基本的なFlexClone機能に対するランダムI/Oレイテンシの最適化など、多数の機能が特別に開発されました。

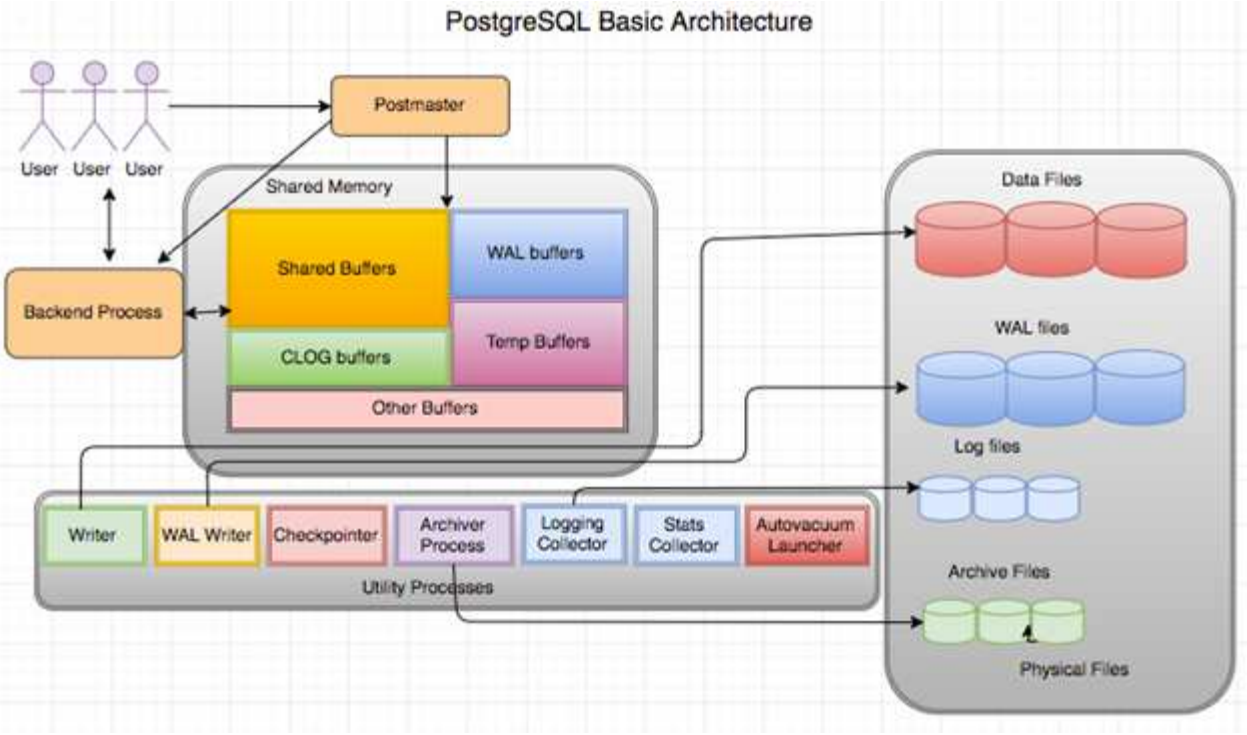
無停止アップグレード（ストレージの交換など）などの追加機能により、重要なデータベースの可用性を維持できます。また、MetroClusterを使用して大規模な環境で瞬時にディザスタリカバリを実行したり、SnapMirrorアクティブ同期を使用してデータベースを選択したりすることもできます。

最も重要なことは、ONTAPが卓越したパフォーマンスを提供し、お客様固有のニーズに合わせて解決策をサイジングできることです。ネットアップのハイエンドシステムは100万超のIOPSをマイクロ秒単位のレイテンシで提供できますが、必要なIOPSが10万だけの場合は、同じストレージオペレーティングシステムを実行する小型のコントローラでストレージ解決策を適切にサイジングできます。

データベース設定

PostgreSQLアーキテクチャ

PostgreSQLは、クライアントとサーバのアーキテクチャに基づいたRDBMSです。PostgreSQLインスタンスはデータベースクラスタと呼ばれ、サーバの集合ではなくデータベースの集合です。



PostgreSQLデータベースには、postmaster、フロントエンド(クライアント)、バックエンドの3つの要素があります。クライアントは、IPプロトコルや接続先データベースなどの情報を含む要求をポストマスターに送信します。postmasterは接続を認証し、さらに通信するためにバックエンドプロセスに渡します。バックエンドプロセスはクエリを実行し、結果を直接フロントエンド（クライアント）に送信します。

PostgreSQLインスタンスは、マルチスレッドモデルではなく、マルチプロセスモデルに基づいています。ジョブごとに複数のプロセスが生成され、各プロセスには独自の機能があります。主なプロセスには、クライアントプロセス、WALライタプロセス、バックグラウンドライタプロセス、およびcheckpointerプロセスが含まれます。

- クライアント（フォアグラウンド）プロセスがPostgreSQLインスタンスに読み取りまたは書き込み要求を送信しても、データを直接ディスクに読み書きすることはありません。最初に、共有バッファとWAL(Write-Ahead Logging)バッファにデータをバッファします。
- WALライタプロセスは、共有バッファとWALバッファの内容を操作してWALログに書き込みます。WALログは通常PostgreSQLのトランザクションログであり、シーケンシャルに書き込まれます。したがって、データベースからの応答時間を短縮するために、PostgreSQLはまずトランザクションログに書き込み、クライアントに確認応答します。
- データベースを整合性のある状態にするために、バックグラウンドライタープロセスは共有バッファにダーティページがないか定期的にチェックします。次に、NetAppボリュームまたはLUNに格納されているデータファイルにデータをフラッシュします。
- checkpointerプロセスも定期的に（バックグラウンドプロセスよりも少ない頻度で）実行され、バッファへの変更を防ぎます。WALライタプロセスに、NetAppディスクに保存されているWALログの末尾にチェックポイントレコードを書き込み、フラッシュするように指示します。また、すべてのダーティページをディスクに書き込み、フラッシュするようにバックグラウンドライタープロセスに通知します。

PostgreSQL初期化パラメータ

新しいデータベースクラスタを作成するには、 `initdb` プログラム。A `initdb` スクリプトは、クラスタを定義するデータファイル、システムテーブル、およびテンプレート

データベース（template0およびtemplate1）を作成します。

テンプレートデータベースはストックデータベースを表します。システムテーブル、標準ビュー、関数、およびデータ型の定義が含まれています。 `pgdata` の引数として機能します。 `initdb` データベースクラスタの場所を指定するスクリプト。

PostgreSQLのすべてのデータベースオブジェクトは、それぞれのOIDによって内部的に管理されます。テーブルとインデックスは、個 々 のOIDによっても管理されます。データベースオブジェクトとそれぞれのOIDとの関係は、オブジェクトのタイプに応じて適切なシステムカタログテーブルに格納されます。たとえば、データベースとヒープテーブルのOIDは、 `pg_database` それぞれ`pg_classと`pg_classです。OIDを確認するには、PostgreSQLクライアントでクエリを発行します。

各データベースには、1GBに制限された個別のテーブルとインデックスファイルがあります。各テーブルには、それぞれサフィックス付きの2つのファイルが関連付けられています。 `_fsm` および `_vm`。これらは、フリースペースマップおよび可視性マップと呼ばれます。これらのファイルには空きスペース容量に関する情報が格納され、テーブルファイルの各ページに表示されます。インデックスには個 々 の空き領域マップのみがあり、可視性マップはありません。

。 `pg_xlog/pg_wal` ディレクトリには、先行書き込みログが格納されます。先行書き込みログは、データベースの信頼性とパフォーマンスを向上させるために使用されます。テーブル内の行を更新するたびに、PostgreSQLは先読みログに変更内容を書き込み、その後実際のデータページに変更内容をディスクに書き込みます。。 `pg_xlog` ディレクトリには通常複数のファイルが含まれていますが、initdbは最初のファイルだけを作成します。必要に応じて追加のファイルが追加されます。各xlogファイルの長さは16MBです。

ONTAPを使用したPostgreSQLデータベースの設定

パフォーマンスを向上させるPostgreSQLのチューニング設定がいくつかあります。

最も一般的に使用されるパラメータは次のとおりです。

- `max_connections = <num>`:一度に持つデータベース接続の最大数。このパラメータを使用して、ディスクへのスワップを制限し、パフォーマンスを強制終了します。アプリケーションの要件に応じて、このパラメータを接続プールの設定に合わせて調整することもできます。
- `shared_buffers = <num>`：データベース・サーバのパフォーマンスを向上させる最も簡単な方法最新のほとんどのハードウェアでは、デフォルトはlowです。導入時に、システム上の使用可能なRAMの約25%に設定されます。このパラメータ設定は、特定のデータベースインスタンスでの動作によって異なります。試行錯誤して値を増減しなければならない場合があります。ただし、この値をHighに設定すると、パフォーマンスが低下する可能性があります。
- `effective_cache_size = <num>`:この値は、PostgreSQLのオプティマイザに、PostgreSQLがデータをキャッシュするために使用できるメモリ量を伝え、インデックスを使用するかどうかを判断するのに役立ちます。値を大きくすると、インデックスを使用する可能性が高くなります。このパラメータは、に割り当てられているメモリの量に設定する必要があります。 `shared_buffers` さらに、使用可能なOSキャッシュの容量も表示されます。多くの場合、この値はシステムメモリ全体の50%を超えています。
- `work_mem = <num>`:このパラメータは、ソート操作およびハッシュテーブルで使用するメモリの量を制御します。アプリケーションで大量のソートを行う場合は、メモリの量を増やす必要があるかもしれませんが、注意が必要です。これはシステム全体のパラメータではなく、操作ごとのパラメータです。複雑なクエリに複数のソート操作が含まれている場合、複数のwork_mem単位のメモリを使用し、複数のバックエンドが同時にこれを実行する可能性があります。このクエリを実行すると'値が大きすぎると'データベース・サーバがスワップされることがよくありますこのオプションは、以前のバージョンのPostgreSQLではsort_memと呼ばれていました。

- `fsync = <boolean> (on or off)`:このパラメータは、トランザクションがコミットされる前にfsync()を使用して、すべてのWALページをディスクに同期するかどうかを決定します。電源をオフにすると、書き込みパフォーマンスが向上し、オンにすると、システムクラッシュ時の破損のリスクからの保護が強化されます。
- `checkpoint_timeout`:チェックポイント・プロセスは'コミットされたデータをディスクにフラッシュしますこれには、ディスク上で多くの読み取り/書き込み処理が含まれます。値は秒単位で設定され、値を小さくするとクラッシュリカバリ時間が短縮されます。値を大きくすると、チェックポイントコールが削減されるため、システムリソースの負荷が軽減されます。アプリケーションの重要度、使用状況、データベースの可用性に応じて、checkpoint_timeoutの値を設定します。
- `commit_delay = <num>` および `commit_siblings = <num>`:これらのオプションを組み合わせて使用すると、一度にコミットする複数のトランザクションを書き出すことで、パフォーマンスを向上させることができます。トランザクションがコミットされた瞬間に複数のcommit_siblingsオブジェクトがアクティブになっている場合、サーバはcommit_delayマイクロ秒を待って、一度に複数のトランザクションをコミットしようとします。
- `max_worker_processes / max_parallel_workers`:プロセスに最適なワーカー数を設定します。max_parallel_workersは、使用可能なCPUの数に対応します。アプリケーションの設計によっては、クエリで並列処理に必要なワーカーの数が少なくて済みます。両方のパラメータの値は同じにし、テスト後に値を調整することをお勧めします。
- `random_page_cost = <num>`:この値は、PostgreSQLが非シーケンシャルディスク読み取りを表示する方法を制御します。値を大きくすると、PostgreSQLはインデックススキャンではなくシーケンシャルスキャンを使用する可能性が高くなります。これは、サーバーに高速ディスクがあることを示します。計画ベースの最適化、真空化、クエリやスキーマの変更に対するインデックス付けなど、他のオプションを評価した後で、この設定を変更してください。
- `effective_io_concurrency = <num>`:このパラメータは、PostgreSQLが同時に実行を試みる同時ディスクI/O処理の数を設定します。この値を大きくすると、個 々 のPostgreSQLセッションが並行して開始しようとするI/O処理の数が増加します。指定できる範囲は1～1、000です。非同期I/O要求の発行を無効にする場合は0にします。現在、この設定はビットマップヒープスキャンにのみ影響します。ソリッドステートドライブ（SSD）やその他のメモリベースストレージ（NVMe）は、多数の同時要求を処理できることが多いため、数百の数を推奨します。

PostgreSQL設定パラメータの完全なリストについては、PostgreSQLのドキュメントを参照してください。

トースト

TOASTは、特大属性ストレージテクニックを表しています。PostgreSQLは固定のページサイズ（通常は8KB）を使用しており、タプルを複数のページにまたがることはできません。したがって、大きなフィールド値を直接保存することはできません。このサイズを超える行を格納しようとすると、トーストは大きな列のデータを小さな「ピース」に分割してトーストテーブルに格納します。

トーストされた属性の大きな値は'結果セットがクライアントに送信されるときにのみ(選択されている場合)プルアウトされますテーブル自体は非常に小さく、アウトオブラインストレージ（TOAST）を使用しない場合よりも多くの行を共有バッファキャッシュに格納できます。

バキューム

通常のPostgreSQL操作では、更新によって削除または廃止されたタプルはテーブルから物理的に削除されず、VACUUMが実行されるまで存在したままになります。したがって、特に頻繁に更新されるテーブルでは、VACUMを定期的に実行する必要があります。ディスクスペースが使用されているスペースは、ディスクスペースが停止しないように、新しい行で再利用できるように再利用する必要があります。ただし、スペースはオペレーティングシステムに返されません。

ページ内の空き領域は断片化されません。VACUUMはブロック全体を書き換え、残りの行を効率的にパッキングし、1つの連続した空き領域ブロックをページに残します。

一方、VACUUM FULLは、デッドスペースのないまったく新しいバージョンのテーブルファイルを作成することで、テーブルを積極的に圧縮します。この操作により、テーブルのサイズは最小限に抑えられますが、時間がかかることがあります。また、処理が完了するまで、テーブルの新しいコピー用に追加のディスクスペースが必要になります。ルーチンバキュームの目的は、バキュームフルアクティビティを回避することです。このプロセスでは、テーブルが最小サイズに維持されるだけでなく、ディスクスペースの安定した使用量も維持されます。

PostgreSQLテーブルスペース

データベース・クラスタが初期化されると、2つの表領域が自動的に作成されます。

。 `pg_global` 表領域は共有システムカタログに使用されます。。 `pg_default` tablespaceは、template1およびtemplate0データベースのデフォルトのテーブルスペースです。クラスタが初期化されたパーティションまたはボリュームの容量が不足し、拡張できない場合は、別のパーティションに表領域を作成して、システムを再構成できるようになるまで使用できます。

頻繁に使用されるインデックスは、ソリッドステートデバイスのような高速で可用性の高いディスクに配置できます。また、ほとんど使用されない、またはパフォーマンスが重要でないアーカイブデータを格納するテーブルは、SASドライブやSATAドライブなどの低コストで低速なディスクシステムに格納できます。

表領域はデータベースクラスタの一部であり、データファイルの自律的なコレクションとして扱うことはできません。これらは、メインデータディレクトリに含まれるメタデータに依存するため、別のデータベースクラスタに接続したり、個別にバックアップしたりすることはできません。同様に、（ファイル削除やディスク障害などによって）テーブルスペースが失われると、データベースクラスタが読み取り不能になったり、起動できなくなったりすることがあります。RAMディスクのような一時ファイルシステムに表領域を配置すると、クラスタ全体の信頼性が低下します。

作成後、要求元ユーザに十分な権限があれば、任意のデータベースから表領域を使用できます。PostgreSQLは、テーブルスペースの実装を簡素化するためにシンボリックリンクを使用します。PostgreSQLは、 `pg_tablespace` Table（クラスタ全体のテーブル）を作成し、その行に新しいオブジェクト識別子（OID）を割り当てます。最後に、サーバはOIDを使用して、クラスタと指定されたディレクトリの間にシンボリックリンクを作成します。ディレクトリ `$PGDATA/pg_tblspc` クラスタで定義されている組み込み以外の各表領域を参照するシンボリックリンクが含まれます。

ストレージ構成

NFSファイルシステムを使用したPostgreSQLデータベース

PostgreSQLデータベースは、NFSv3またはNFSv4ファイルシステムでホストできます。最適なオプションは、データベース外の要因によって異なります。

たとえば、特定のクラスタ環境ではNFSv4のロック動作が推奨されます。（参照： "こちらをご覧ください" 詳細はこちら）

それ以外の場合は、パフォーマンスも含めて、データベース機能はほぼ同一である必要があります。唯一の要件は、 `hard` マウントオプション。これは、ソフトタイムアウトによって回復不能なIOエラーが発生しないようにするために必要です。

NFSv4がプロトコルとして選択されている場合、NetAppではNFSv4.1の使用を推奨します。NFSv4.1では、NFSv4.0よりも耐障害性が向上するように、NFSv4プロトコルの機能がいくつか拡張されています。

一般的なデータベースワークロードには、次のマウントオプションを使用します。

```
rw,hard,nointr,bg,vers=[3|4],proto=tcp,rsize=65536,wsize=65536
```

大量のシーケンシャルIOが予想される場合は、次のセクションの説明に従ってNFS転送サイズを増やすことができます。

NFSテンソウサイス

ONTAPでは、デフォルトでNFS I/Oサイズが64Kに制限されています。

ほとんどのアプリケーションとデータベースでランダムI/Oを実行すると、ブロックサイズがはるかに小さくなり、最大64Kよりもはるかに小さくなります。ラージブロックI/Oは通常並列処理されるため、最大64Kも最大帯域幅の確保に制限されるわけではありません。

一部のワークロードでは、最大64Kに制限があります。特に、バックアップ/リカバリ処理やデータベースのフルテーブルスキャンなどのシングルスレッド処理は、実行回数が少なくても大容量のI/Oを実行できるのであれば、より高速かつ効率的に実行できます。ONTAPに最適なI/O処理サイズは256Kです。

特定のONTAP SVMの最大転送サイズは、次のように変更できます。

```
Cluster01::> set advanced
Warning: These advanced commands are potentially dangerous; use them only
when directed to do so by NetApp personnel.
Do you want to continue? {y|n}: y
Cluster01::*> nfs server modify -vserver vserver1 -tcp-max-xfer-size
262144
Cluster01::*>
```

注意
ONTAPで許容される最大転送サイズを、現在マウントされているNFSファイルシステムのrsize/wsizeの値より小さくしないでください。これにより、一部のオペレーティングシステムでハングしたり、データが破損したりする可能性があります。たとえば、NFSクライアントのrsize / wsizeが65536に設定されている場合は、クライアント自体が制限されているため、ONTAPの最大転送サイズを65536~1048576の間で調整しても効果はありません。最大転送サイズを65536未満に縮小すると、可用性やデータが損傷する可能性があります。

転送サイズをONTAPレベルで拡張すると、次のマウントオプションが使用されます。

```
rw,hard,nointr,bg,vers=[3|4],proto=tcp,rsize=262144,wsize=262144
```

NFSv3 TCPスロットテーブル

LinuxでNFSv3を使用する場合は、TCPスロットテーブルを適切に設定することが重要です。

TCPスロットテーブルは、NFSv3でホストバスアダプタ（HBA）のキュー深度に相当します。一度に未処理となることのできるNFS処理の数を制御します。デフォルト値は通常16ですが、最適なパフォーマンスを得るには小さすぎます。逆に、新しいLinuxカーネルでTCPスロットテーブルの上限をNFSサーバが要求でいっぱいになるレベルに自動的に引き上げることができるため、問題が発生します。

パフォーマンスを最適化し、パフォーマンスの問題を回避するには、TCPスロットテーブルを制御するカーネルパラメータを調整します。

を実行します `sysctl -a | grep tcp.*.slot_table` コマンドを実行し、次のパラメータを確認します。

```
# sysctl -a | grep tcp.*.slot_table
sunrpc.tcp_max_slot_table_entries = 128
sunrpc.tcp_slot_table_entries = 128
```

すべてのLinuxシステムに `sunrpc.tcp_slot_table_entries`ただし、次のようなものがあります。 `sunrpc.tcp_max_slot_table_entries`。どちらも128に設定する必要があります。

注意
これらのパラメータを設定しないと、パフォーマンスに大きく影響する可能性があります。Linux OSが十分なI/Oを発行していないためにパフォーマンスが制限される場合もあります。一方では、Linux OSが問題で処理できる以上のI/Oを試行すると、I/Oレイテンシが増加します。

SANファイルシステムを使用するPostgreSQL

SANを使用したPostgreSQLデータベースは、通常xfsファイルシステムでホストされますが、OSベンダーがサポートしていれば他のデータベースも使用できます。

1つのLUNで最大10万IOPSをサポートできますが、I/O負荷の高いデータベースでは、一般にLVMとストライピングを使用する必要があります。

LVMストライピング

フラッシュドライブが登場する以前は、回転式ドライブのパフォーマンス上の制限を克服するためにストライピングが使用されていました。たとえば、OSが1MBの読み取り操作を実行する必要がある場合、1つのドライブからその1MBのデータを読み取るには、1MBがゆっくり転送されるため、多くのドライブヘッドのシークと読み取りが必要になります。この1MBのデータが8つのLUNにストライピングされている場合、OSは8つの128K読み取り処理を並行して問題できるため、1MB転送の完了に必要な時間が短縮されます。

回転式ドライブを使用したストライピングは、I/Oパターンを事前に把握しておく必要があったため、より困難でした。ストライピングが実際のI/Oパターンに合わせて正しく調整されていない場合、ストライピングされた構成ではパフォーマンスが低下する可能性があります。Oracleデータベース、特にオールフラッシュ構成では、ストライピングは設定がはるかに簡単で、パフォーマンスが劇的に向上することが実証されています。

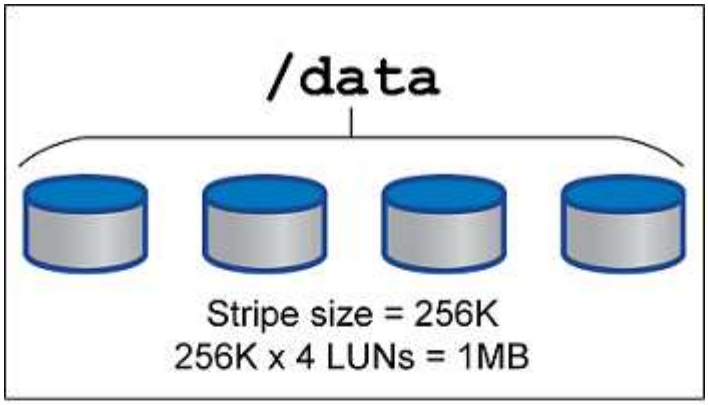
デフォルトではOracle ASMなどの論理ボリュームマネージャがストライプされますが、ネイティブOS LVMはストライプされません。その中には、複数のLUNを連結されたデバイスとして結合するものもあります。その

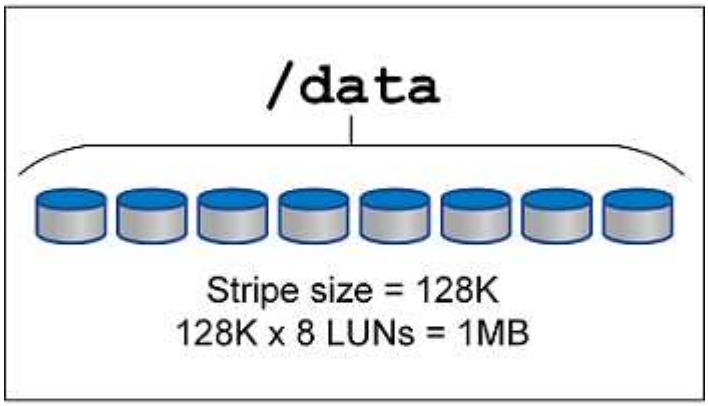
ため、データファイルは1つのLUNデバイスにしか存在しません。これにより、ホットスポットが発生します。他のLVM実装では、デフォルトで分散エクステントが使用されます。これはストライピングに似ていますが、粗いです。ボリュームグループ内のLUNはエクステントと呼ばれる大きな部分にスライスされ、通常は数メガバイト単位で測定され、論理ボリュームがそれらのエクステントに分散されます。その結果、ファイルに対するランダムI/OはLUN間で適切に分散されますが、シーケンシャルI/O処理はそれほど効率的ではありません。

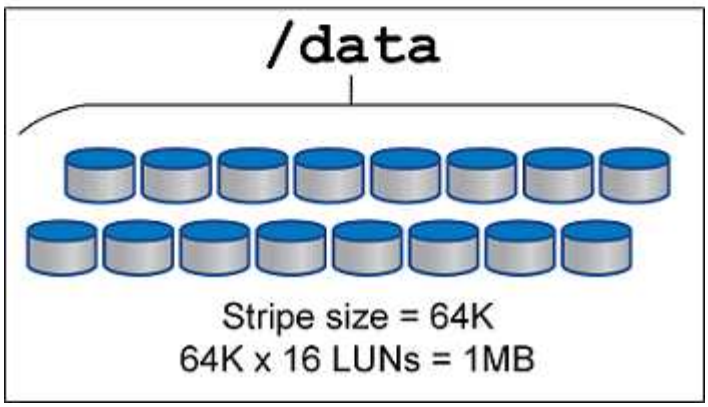
高いパフォーマンスを必要とするアプリケーションI/Oは、ほとんどの場合（a）基本ブロックサイズの単位または（b）1メガバイトのいずれかです。

ストライピング構成の主な目的は、シングルファイルI/Oを1つのユニットとして実行し、マルチブロックI/O（サイズは1MB）をストライピングされたボリューム内のすべてのLUNで均等に並列化できるようにすることです。つまり、ストライプ・サイズはデータベース・ブロック・サイズより小さくすることはできず、ストライプ・サイズにLUN数を掛けたサイズは1MBにする必要があります。

次の図に、ストライプサイズと幅の調整に使用できる3つのオプションを示します。LUNの数は、前述のパフォーマンス要件を満たすように選択されますが、いずれの場合も、1つのストライプ内の総データ量は1MBです。







データ保護

PostgreSQLのデータ保護

ストレージ設計の主な側面の1つは、PostgreSQLボリュームの保護を有効にすることです。お客様は、ダンプアプローチを使用するか、ファイルシステムバックアップを使用して、PostgreSQLデータベースを保護できます。このセクションでは、個 々 のデータベースまたはクラスタ全体をバックアップするさまざまな方法について説明します。

PostgreSQLデータをバックアップするには、次の3つの方法があります。

- SQL Serverダンプ
- ファイルシステムレベルのバックアップ
- 継続的アーカイブ

SQL Serverダンプ方式の背後にある考え方は、SQL Serverコマンドを使用してファイルを生成することです。このコマンドをサーバに返すと、ダンプ時と同じようにデータベースを再作成できます。PostgreSQLはユーティリティプログラムを提供します。 `pg_dump` および `pg_dump_all` 個 々 のバックアップとクラスタレベルのバックアップの作成に使用します。これらのダンプは論理的であり、WAL再生で使用するのに十分な情報が含まれていません。

別のバックアップ戦略として、PostgreSQLがデータベースにデータを保存するために使用するファイルを管理者が直接コピーするファイルシステムレベルのバックアップを使用する方法があります。この方法はオフラインモードで実行されます。データベースまたはクラスタをシャットダウンする必要があります。もう1つの選択肢は、 `pg_basebackup` PostgreSQLデータベースのホットストリーミングバックアップを実行します。

PostgreSQLデータベースとストレージスナップショット

PostgreSQLを使用したスナップショットベースのバックアップでは、フルリカバリまたはポイントインタイムリカバリを提供するために、データファイル、WALファイル、およびアーカイブされたWALファイルのスナップショットを構成する必要があります。

PostgreSQLデータベースの場合、Snapshotを使用した平均バックアップ時間は数秒~数分です。このバックアップ速度は、 `pg_basebackup` その他のファイル・システム・ベースのバックアップ・アプローチ

NetAppストレージ上のSnapshotは、crash-consistentとアプリケーション整合性の両方が可能です。crash-consistent Snapshotはデータベースを休止せずにストレージ上に作成されますが、アプリケーション整合性Snapshotはデータベースがバックアップモードの間に作成されます。NetAppでは、後続のスナップショットが永久増分バックアップとなるため、ストレージの節約とネットワークの効率化が促進されます。

スナップショットは高速で、システムのパフォーマンスに影響を与えないため、他のストリーミングバックアップテクノロジのように1日1回のバックアップを作成するのではなく、1日に複数のスナップショットをスケジュールできます。リストアとリカバリの処理が必要な場合は、次の2つの主な機能によってシステムのダウンタイムが短縮されます。

- NetApp SnapRestoreのデータリカバリテクノロジにより、リストア処理が数秒で実行されます。
- Recovery Point Objective（RPO；目標復旧時点）が頻繁に発生するため、適用するデータベースログの数が減り、フォワードリカバリも高速化されます。

PostgreSQLをバックアップするには、データボリュームが（コンシステンシグループ）WALとアーカイブログと同時に保護されていることを確認する必要があります。Snapshotテクノロジを使用してWALファイルをコピーする場合は、次のコマンドを実行してください： `pg_stop` アーカイブする必要があるすべてのWALエントリをフラッシュします。リストア中にWALエントリをフラッシュする場合は、データベースを停止するか、既存のデータディレクトリをアンマウントまたは削除し、ストレージでSnapRestore操作を実行するだけで済みます。リストアが完了したら、システムをマウントして現在の状態に戻すことができます。ポイントインタイムリカバリの場合は、WALとアーカイブログをリストアすることもできます。次に、PostgreSQLは最も整合性のあるポイントを決定し、自動的にリカバリします。

整合グループはONTAPの機能であり、1つのインスタンスまたは複数の表領域を含むデータベースに複数のボリュームがマウントされている場合に推奨されます。整合性グループSnapshotを使用すると、すべてのボリュームがグループ化されて保護されます。整合グループはONTAP System Managerから効率的に管理できます。また、整合グループをクローニングして、テストや開発用にデータベースのインスタンスコピーを作成することもできます。

コンシステンシグループの詳細については、を参照してください。 "NetApp整合性グループの概要"。

PostgreSQLデータ保護ソフトウェア

PostgreSQLデータベース用のNetApp SnapCenterプラグインをSnapshotおよびNetApp FlexCloneテクノロジと組み合わせると、次のようなメリットがあります。

- 高速なバックアップとリストア：
- スペース効率に優れたクローン：
- 迅速で効果的なディザスタリカバリシステムを構築する能力。

> 次のような状況では、Veeam SoftwareやCommvaultなど、ネットアップのプレミアムバックアップパートナーを選択することもできます。
>
> - 異機種混在環境全体でのワークロードの管理
> - バックアップをクラウドまたはテープに保存して長期保持
> - さまざまなバージョンと種類のOSをサポート

PostgreSQL用のSnapCenterプラグインはコミュニティでサポートされているプラグインであり、セットアップとドキュメントはNetAppオートメーションストアで入手できます。SnapCenterを使用すると、データベースのバックアップ、データのクローニング、リストアをリモートで実行できます。

著作権に関する情報

商標に関する情報