Amazon Aurora MySQL データベース管理者ハンドブック

接続管理

2019 年 3 月

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。この文書は、(a) 情報提供のみを目的としており、(b) AWS の現行製品と慣行を表したものであるため予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤー、またはライセンサーからの契約義務や確約を意味するものではありません。AWS の製品やサービスは、明示または暗示を問わず、いかなる保証、表明、条件を伴うことなく「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

© 2019 Amazon Web Services, Inc. or its affiliates.All rights reserved.

目次

注意 .....

目次 .....

はじめに 1

DNS エンドポイント 2

AURORA MYSQL での接続処理と MYSQL 3

よくある誤解 4

ベストプラクティス 5

スマートドライバーの使用 5

DNS キャッシング 7

接続管理とプーリング 8

接続スケーリング 9

トランザクション管理と自動コミット 10

接続ハンドシェイク 12

負荷分散とリーダーエンドポイント 13

耐障害性と迅速なリカバリのための設計 13

サーバー設定 14

設定変数 *max_connections* *14*

設定変数 *max_connect_errors* *15*

まとめ 17

寄稿者 17

その他の資料 17

ドキュメントの改訂 17

要約

このホワイトペーパーでは、データベース接続の管理と、サーバー接続パラメータ、クライアントプログラム、ドライバー、コネクタの設定のベストプラクティスについて概要を説明します。Amazon Aurora MySQL データベース管理者 (DBA) およびアプリケーション開発者の方々がお読みになることをお勧めします。

はじめに

Amazon Aurora MySQL (Aurora MySQL) は、MySQL 5.6 および 5.7 と強い互換性のあるマネージドリレーショナルデータベースエンジンです。現在 MySQL で使用しているドライバー、コネクタ、ツールのほとんどは、Aurora MySQL においても、ほとんど、または何も変更せずに使用できます。

Aurora MySQL データベース (DB) クラスターは、次のような高度な機能を提供します。

- 読み取り/書き込み操作をサポートする 1 つのプライマリインスタンスと、読み取り専用操作をサポートする最大 15 個の Aurora レプリカ。現在のプライマリインスタンスで障害が発生した場合、各レプリカを自動的にプライマリロールに昇格させることが可能
- フェイルオーバー時に自動的にプライマリインスタンスに追従するクラスターエンドポイント
- すべての Aurora レプリカを含み、Aurora レプリカが追加または削除されると自動的に更新されるリーダーエンドポイント
- 単一クラスター内のデータベースインスタンスをユーザーが設定したグループに含めることが可能な、カスタム DNS エンドポイント
- 内部でのサーバー接続プーリングとスレッド多重化によって向上したスケーラビリティ
- ほぼ瞬時のデータベース再起動とクラッシュからの復旧
- クラスターメタデータへほぼリアルタイムにアクセ可能であるため、アプリケーション開発者は、インスタンスの読み取り/書き込みまたは読み取り専用のロールに基づいて個別のインスタンスに直接接続可能なスマートドライバーを構築可能

準最適な構成を使用するクライアント側コンポーネント (アプリケーション、ドライバー、コネクタ、プロキシ) は、復旧アクションや DB クラスタートポロジの変更に反応できない、または反応が遅れる場合があります。これによって、予期しないダウンタイムやパフォーマンスの問題が発生する可能性があります。そのような状況を防ぎ、Aurora MySQL の機能を最大限に活用するために、データベース管理者 (DBA) およびアプリケーション開発者は、このホワイトペーパーで説明するベストプラクティスを実施することをお勧めします。

DNS エンドポイント

Aurora DB クラスターは、1 つ以上のインスタンスと、それらのインスタンスに対してデータを管理するクラスターボリュームで構成されています。2 種類のインスタンスがあります。

- プライマリインスタンス – 読み取りおよび書き込みステートメントをサポートします。現在は、DB クラスターごとに 1 つのプライマリインスタンスが存在できます
- Aurora レプリカ – 読み取り専用ステートメントをサポートします。DB クラスターあたりの Aurora レプリカの最大数は 15 個です。Aurora レプリカは読み取りスケーリングに使用でき、プライマリインスタンスで障害が発生すると自動的にフェイルオーバーターゲットとして使用されます

Aurora では、次の種類のドメインネームシステム (DNS) エンドポイントがサポートされています。

- クラスターエンドポイント – フェイルオーバー時、つまり、現在のプライマリインスタンスが降格してその代わりに Aurora レプリカのうちの 1 つが昇格したときに、プライマリインスタンに自動的に追従します
- リーダーエンドポイント – DB クラスター内のすべての Aurora レプリカを単一の DNS CNAME 下に含みます。リーダーエンドポイントを使用して、読み取り専用接続に対して DNS ラウンドロビンロードバランシングを実装できます
- インスタンスエンドポイント – DB クラスター内の各インスタンスには、独自のインスタンスエンドポイントがあります。このエンドポイントを使用して特定のインスタンスに直接接続できます
- カスタムエンドポイント – クラスターから選択された特定のインスタンスの
- グループを含むユーザー定義の DNS エンドポイント

詳細については、Amazon Aurora の概要ページを参照してください。[1]

Aurora MySQL での接続処理と MySQL

MySQL Community Edition では、接続ごとに 1 つのスレッドという方式で接続を管理します。つまり、個別のユーザー接続はそれぞれ、mysqld プロセス内で専用のオペレーティングシステムスレッドを受け取ります。このタイプの接続処理には、次のような問題があります。

- 多数のユーザー接続があると、接続が完全にアイドル状態であってもメモリ使用量が比較的多い
- 何千ものユーザー接続を処理するとき、サーバ内部の競合とコンテキストスイッチのオーバーヘッドが高い

Aurora MySQL では、これらの問題に対処するスレッドプールアプローチをサポートしています。スレッドプールアプローチの特徴は次のとおりです。

- スレッド多重化を使用し、多数のワーカースレッドがユーザーセッション (接続) を切り替えることができます。ワーカースレッドは単一のユーザーセッションに固定されず、それ専用になることもありません。ワーカースレッドは、(アイドル状態、ユーザー入力の待機中、I/O の待機中などで) 接続がアクティブに実行されていないときはいつでも、別の接続に切り替えて有益な処理を実行できます。ワーカースレッドは、マルチコアシステムにおける CPU コアのようなものと考えることができます。コアが数個しかなくても、すべてのコアが同時にアクティブになるわけではないため、何百ものプログラムを容易に同時実行できます。この非常に効率的なアプローチのおかげで、Aurora MySQL はごく少数のワーカースレッドで何千ものクライアントを同時に処理できます
- スレッドプールは自動的にそれ自体をスケールします。Aurora MySQL データベースプロセスは、そのスレッドプールの状態を継続的に監視し、必要に応じて新しいワーカーを起動あるいは既存のワーカーを破棄します。これはユーザーが明示的に行う必要はなく、手動での設定は何も必要ありません

サーバースレッドプーリングは、サーバー側で接続を維持するコストを低減します。ただし、これらの接続を最初にセットアップするときのコストをなくすことはできません。接続の開閉は、1 つの TCP パケットの送信のような単純な処理ではありません。(Key-Value やオンライントランザクション処理のように) 接続が短時間で負荷が大きいワークロードについては、アプリケーション側の接続プールの使用を検討してください。

次に示すのは、同じアベイラビリティーゾーンに位置するクライアントと MySQL 互換サーバーの間で発生する MySQL 接続ハンドシェイクのネットワークパケットトレースです。

```
04:23:29.547316 IP client.32918 > server.mysql: tcp 0
04:23:29.547478 IP server.mysql > client.32918: tcp 0
04:23:29.547496 IP client.32918 > server.mysql: tcp 0
04:23:29.547823 IP server.mysql > client.32918: tcp 78
04:23:29.547839 IP client.32918 > server.mysql: tcp 0
04:23:29.547865 IP client.32918 > server.mysql: tcp 191
04:23:29.547993 IP server.mysql > client.32918: tcp 0
04:23:29.548047 IP server.mysql > client.32918: tcp 11
04:23:29.548091 IP client.32918 > server.mysql: tcp 37
04:23:29.548361 IP server.mysql > client.32918: tcp 99
04:23:29.587272 IP client.32918 > server.mysql: tcp 0
```

これは、接続を閉じるためのパケットトレースです。

```
04:23:37.117523 IP client.32918 > server.mysql: tcp 13
04:23:37.117818 IP server.mysql > client.32918: tcp 56
04:23:37.117842 IP client.32918 > server.mysql: tcp 0
```

このように、1 つの接続の開閉という単純な処理でさえ、複数のネットワークパケットの交換を伴います。接続のセットアップ中にドライバーによって発行される SQL ステートメントを考慮に入れると、接続オーバーヘッドはより顕著になります (たとえば、セッションレベルの設定のために使用される SET variable_name = value コマンド)。サーバー側のスレッドプーリングによって、このようなオーバーヘッドがなくなるわけではありません。

よくある誤解

データベース接続管理についてのよくある誤解には、以下のものがあります。

サーバーで接続プーリングを使用していれば、アプリケーション側のプールは必要ない。
前に説明したように、これは、接続のオープンと切断が非常に頻繁なワークロードや、接続あたりのステートメント実行数が比較的少ないクライアントには当てはまりません。
接続の寿命が長い場合、接続プールが不要なこともあります。これは、接続アクティビティ時間が、接続の開閉に必要な時間よりもずっと長いことを意味します。

tcpdump でパケットトレースを実行し、接続の開閉に必要なパケット数と、それらの接続内でクエリを実行するために必要なパケット数を比較できます。接続の寿命が長い場合でも、接続プールを使用してデータベースを接続サージ (新しい接続試行の急激な増加) から保護できるメリットがあります。

アイドル状態の接続はメモリを使用しない。これは正しくありません。なぜなら、オペレーティングシステムとデータベースプロセスのどちらも、ユーザー接続ごとにメモリ内記述子を割り当てるからです。一般的に正しいのは、同じ数の接続を維持するために使用するメモリは、Aurora MySQL のほうが MySQL Community Edition より少ないということです。ただし、アイドル状態の接続のメモリ使用量は、Aurora MySQL でもゼロにはなりません。

一般的なベストプラクティスは、必要な数を大きく超えて接続を開かないことです。

ダウンタイムはデータベースの安定性とデータベースの機能だけに左右される。これは正しくありません。なぜなら、アプリケーションの設計と構成も、データベースイベント後にユーザートラフィックを復旧できる速さを決める重要な要因であるからです。詳細については、次のセクション「ベストプラクティス」を参照してください。

ベストプラクティス

以下に示すのは、データベース接続を管理し、接続ドライバーと接続プールを設定するためのベストプラクティスです。

スマートドライバーの使用

クラスターエンドポイントとリーダーエンドポイントは、DB クラスターで発生するロール変更 (プライマリインスタンスの昇格/降格) とトポロジ変更 (インスタンスの追加と削除) を抽象化します。ただし、DNS の更新は即時ではありません。また、DNS の更新のために、データベースイベントが発生してからそれがアプリケーションで認識および処理されるまでの遅延がわずかに延びる可能性があります。

Aurora MySQL では、DB インスタンスについてのほぼリアルタイムのメタデータを INFORMATION_SCHEMA.REPLICA_HOST_STATUS テーブルで公開します。

次に示すのは、メタデータテーブルに対するクエリの例です。

```
mysql> select server_id, if(session_id = 'MASTER_SESSION_ID',
'writer', 'reader') as role, replica_lag_in_milliseconds from
information_schema.replica_host_status;

+----------------------+--------+------------------------------+
| server_id            | role   | replica_lag_in_milliseconds |
+----------------------+--------+------------------------------+
| aurora-node-usw2a | writer |                            0 |
| aurora-node-usw2b | reader |      19.253999710083008 |
+----------------------+--------+------------------------------+
2 rows in set (0.00 sec)
```

テーブルにはクラスター全体のメタデータが含まれていることがわかります。DB クラスター内の任意のインスタンスに対してテーブルをクエリできます。

このホワイトペーパーでいう**スマートドライバー**とは、DB クラスタートポロジをメタデータテーブルから読み取ることができるデータベースドライバーまたはコネクタのことです。高レベルのクラスターエンドポイントに頼ることなく、個別のインスタンスエンドポイントに新しい接続をルーティングできます。スマートドライバーは通常、読み取り専用接続の負荷を、利用可能な Aurora レプリカ間でラウンドロビン方式で分散させることもできます。

MariaDB Connector/J は、Aurora MySQL DB クラスターをネイティブでサポートするサードパーティーの Java Database Connectivity (JDBC) スマートドライバーの一例です。アプリケーション開発者は MariaDB ドライバーを参考にして、Java 以外の言語用のドライバーおよびコネクタを作成できます。

詳細については、MariaDB Connector/J のページを参照してください。[2]

スマートドライバーを使用している場合でも、以降のセクションで説明している推奨事項は該当します。スマートドライバーは、データベース接続の特定のレイヤーを自動化および抽象化できます。ただし、最適な設定でそれ自体を自動的に設定したり、障害に対する弾力性をアプリケーションに自動で持たせることはできません。たとえば、スマートドライバーを使用している場合でも、接続の検証とリサイクルの機能が正しく設定されていること、基盤となるシステムとネットワークレイヤーで DNS キャッシュが過剰でないこと、トランザクションが正しく管理されていることなどを確認する必要があります。

お使いのセットアップでスマートドライバーの使用を評価することをお勧めします。サードパーティーのドライバーに Aurora MySQL 固有の機能が含まれていても、それは AWS によって正式にテスト、検証、または認定されていることを意味するものではない点ご了承ください。また、スマートドライバーには高度な機能が組み込まれており、全体的に複雑さが増しているため、多くの場合、従来の (ベアボーン) ドライバーよりも更新やバグ修正が頻繁です。ドライバーのリリースノートを定期的に確認し、可能であれば常に最新の利用可能なバージョンを使用してください。

DNS キャッシング

スマートデータベースドライバーを使用しない場合、Aurora レプリカ間でのフェイルオーバー、インスタンススケーリング、および負荷分散は DNS レコード更新と DNS 伝播に依存します。現在、Auroraが利用しているDNSの設定では5秒という短い有効期限 (TTL) が使用されています。決して、ネットワークとクライアントの設定で DNS キャッシュの TTL をそれ以上長くしないでください。

DNS キャッシングは、ネットワークレイヤーからオペレーティングシステム、さらにはアプリケーションコンテナまで、どこでも発生する可能性があることを覚えておいてください。たとえば、Java 仮想マシン (JVM) で、設定を変更しない限り DNS が無期限にキャッシュされる例があります。

以下は、DNS キャッシングのベストプラクティスに従わない場合に起きる可能性がある問題の例です。

- フェイルオーバー中に新しいプライマリインスタンスが昇格した後も、アプリケーションは古いインスタンスに書き込みトラフィックを送信し続けます。そのインスタンスはもうプライマリインスタンスではないので、データ変更ステートメントは失敗します
- DB インスタンスがスケールアップまたはスケールダウンされた後、アプリケーションは新しいインスタンスに接続できません。DNSキャッシュにより、アプリケーションは変更後のインスタンスに変更前の古い IP アドレスを使用して接続を行おうとしますが、そのアドレスはもう無効です
- ある DB インスタンスが他のインスタンスよりもはるかに多くのトラフィックを受信するなど、Aurora レプリカ間で使用率に偏りが生じる可能性があります

接続管理とプーリング

データベース接続は、開発フレームワークや言語のデストラクタに任せるのではなく、常に明示的に閉じるようにしてください。コンテナベースや code-as-a-service のシナリオでは特に、基礎となるコードコンテナがコードの完了後すぐに破棄されない状況があります。そのような場合、開いたままの接続が (メモリやロックなどの) リソースを保持し続けるという、データベース接続リークが発生する可能性があります。

クライアントアプリケーション (または対話型クライアント) がアイドル状態の接続を閉じることが確実でない場合、サーバーの wait_timeout および interactive_timeout パラメータを使用してアイドル接続タイムアウトを設定します。デフォルトのタイムアウト値は 28,800 秒 (8 時間) と非常に大きくなっています。お使いの環境で許容できる値となるように下方調整してください。詳細については、MySQL リファレンスマニュアルを参照してください。[3]

データベースを接続サージから保護するために、接続プーリングの使用を検討してください。アプリケーションが多数の接続を開く (たとえば、1 秒あたり数千回以上) が接続が短時間である、つまり、接続のセットアップと切断に必要な時間が接続の寿命全体と比べて長い場合にも、接続プーリングを検討してください。お使いの開発言語/フレームワークで接続プールがサポートされていない場合、代わりに接続プロキシを使用できます。ProxySQL、MaxScale、および ScaleArc は、MySQL プロトコルと互換性のあるサードパーティー製プロキシの例です。接続プールとプロキシに関するその他の注意事項は、接続のスケーリングを参照してください。

接続プールとプロキシの設定に関する推奨事項は以下のとおりです。

- 接続をプールから借用している場合、接続の正常性をチェックおよび検証します。**SELECT 1** のような単純な検証クエリでかまいません。ただし Aurora では、インスタンスがプライマリインスタンス (読み取り/書き込み) か、それとも Aurora レプリカ (読み取り専用) かによって異なる値を返す接続チェックも利用できます。たとえば、@@innodb_read_only 変数を使用してインスタンスのロールを判別できます。変数の値が TRUE の場合、Aurora レプリカです

- 接続を借用していない場合でも、定期的に接続をチェックおよび検証します。アプリケーションスレッドが接続を使おうとする前に、壊れているか正常でない接続を検出してクリーンアップするのに役立ちます
- 接続をいつまでもプールに置いたままにしないでください。定期的に (たとえば 15 分ごとに) 接続を閉じて再び開くことで接続をリサイクルします。これにより、これらの接続に関連付けられているリソースが解放されます。クライアントが放棄した暴走クエリやゾンビ接続などの危険な状況を防ぐためにも役立ちます。この推奨事項は、アイドル状態の接続だけでなく、すべての接続に適用されます

接続スケーリング

ウェブサービスのキャパシティーの最も一般的なスケーリング手法は、ユーザートラフィックの変化に応じてアプリケーションサーバー (インスタンス) を追加または削除することです。各アプリケーションサーバーはデータベース接続プールを使用可能です。

このアプローチでは、アプリケーションインスタンスの数に比例してデータベース接続の総数が増加します。たとえば、20 のアプリケーションサーバーでデータベース接続数がそれぞれ 200 に設定されている場合、合計 4,000 のデータベース接続が必要になります。(ピーク時などに) アプリケーションプールが 200 インスタンスまで拡張すると、合計接続数は 40,000 に達します。典型的なウェブアプリケーションのワークロードでは、おそらく、これらの接続のほとんどがアイドル状態です。極端なケースでは、これによってデータベースのスケーラビリティが制限される可能性があります。アイドル状態の接続がサーバーリソースを占有し、必要な数よりもずっと多くの接続を開いています。また、接続の総数は直接設定する類のものではなく、むしろアプリケーションサーバーの数に依存するため、制御が容易ではありません。

この状況では 2 つの選択肢があります。

- アプリケーションインスタンスで接続プールを有効にします。プール内の接続数を許容範囲内の下限にまで減らします。これは応急的な解決策にはなりますが、アプリケーションサーバーの数は増え続けるため、長期的な解決策にはならないかもしれません

- データベースとアプリケーションの間に接続プロキシを導入します。一方では、プロキシは接続数固定でデータベースに接続します。もう一方では、プロキシはアプリケーション接続を受け入れ、クエリキャッシング、接続バッファリング、クエリ書き換え/ルーティング、負荷分散などの追加機能を提供できます。ProxySQL、MaxScale、および ScaleArc は、MySQL プロトコルと互換性のあるサードパーティー製プロキシの例です。スケーラビリティと可用性をさらに高めるために、単一の DNS エンドポイントの背後で複数のプロキシインスタンスを使用可能です

トランザクション管理と自動コミット

自動コミットが有効になっていると、各 SQL ステートメントはそれ自体のトランザクションの内部で実行されます。ステートメントが終了すると、トランザクションも終了します。ステートメントとステートメントの間、クライアント接続は**トランザクション内**にありません。複数のステートメントに対してトランザクションを開いたままにしておく必要がある場合は、トランザクションを明示的に開始し、ステートメントを実行してから、トランザクションをコミットまたはロールバックします。

自動コミットが無効になっていると、接続は常に**トランザクション内**にあります。現在のトランザクションをコミットまたはロールバックでき、その時点でサーバーはすぐに新しいトランザクションを開きます。

詳細については、MySQL リファレンスマニュアルを参照してください。[4]

自動コミットを無効にして実行することは推奨されません。不要な場面で長時間にわたってトランザクションが実行されやすくなるからです。開いているトランザクションは、最適なパフォーマンスを維持するために不可欠である、サーバー内部のガベージコレクションメカニズムをブロックします。極端なケースでは、ガベージコレクションのバックログは、過剰なストレージ消費、CPU 使用率の上昇、クエリ速度につながります。

推奨事項:

- 常に自動コミットモードを有効にして実行します。autocommit パラメータを **1** に設定します。これはデータベース側ではデフォルトですが、アプリケーション側ではデフォルトでない可能性があります

- 必ず、アプリケーション側で自動コミット設定を再確認してください。たとえば、MySQLdb や PyMySQL などの Python ドライバーはデフォルトで自動コミットを無効化します
- BEGIN/START TRANSACTION および COMMIT/ROLLBACK ステートメントを使用してトランザクションを明示的に管理します。トランザクションは必要なときに開始し、トランザクション処理が完了したらすぐにコミットしてください

これらの推奨事項は Aurora MySQL に固有のものではないことに注意してください。MySQL や、InnoDB ストレージエンジンを使用する他のデータベースにも当てはまります。

長いトランザクションやガベージコレクションのバックログは簡単に監視できます。

- 現在実行中のトランザクションのメタデータは INFORMATION_SCHEMA.INNODB_TRX テーブルから取得できます。TRX_STARTED 列にはトランザクションの開始時刻が含まれており、これを使用してトランザクションの長さを計算できます。トランザクションが数分以上にわたって実行されてないかどうかを調べることには価値があります。テーブルの詳細については、MySQL リファレンスマニュアルを参照してください。5
- ガベージコレクションのバックログのサイズは、InnoDB の INFORMATION_SCHEMA.INNODB_METRICS テーブルの trx_rseg_history_len カウンターから読み取ることができます。テーブルの詳細については、MySQL リファレンスマニュアルを参照してください。6 カウンターの値が大きければ大きいほど、クエリのパフォーマンス、CPU 使用率、ストレージ消費の面で影響が深刻になる可能性があります。数万の範囲にある値は、ガベージコレクションが多少遅れていることを示します。数百万から数千万の範囲にある値は、危険性を示している可能性があり、調査が必要です。

> **注記:** Aurora では、すべての DB インスタンスが同じストレージボリュームを使用します。つまり、ガベージコレクションはクラスター全体のものであり、各インスタンスに固有のものではありません。その結果、1 つのインスタンスでトランザクションが暴走するとすべてのインスタンスに影響が及ぶ可能性があります。したがって、すべての DB インスタンスで長いトランザクションを監視する必要があります。

接続ハンドシェイク

アプリケーションコネクタまたはグラフィカルユーザーインターフェイス (GUI) ツールが新しいデータベースセッションを開くと、裏側で多くの処理が発生する可能性があります。ドライバーとクライアントツールは通常、一連のステートメントを実行してセッションの構成をセットアップします (例: SET SESSION variable = value)。これによって、新しい接続を作成するコストが増加し、アプリケーションがクエリの発行を開始できるタイミングが遅れます。

アプリケーションがレイテンシーの影響を非常に受けやすい場合、接続ハンドシェイクのコストはいっそう重要になります。接続を開くたびのコストが高いと、1 桁ミリ秒のレイテンシーが想定されている OLTP または Key-Value ワークロードでは目に見えて影響が出る可能性があります。たとえば、接続をセットアップするためにドライバーが 6 つのステートメントを実行し、各ステートメントの実行所要時間が 1 ミリ秒にすぎないとしても、アプリケーションが最初のクエリを発行するまでに 6 ミリ秒の遅延が生じます。

推奨事項:

- Aurora MySQL Advanced Audit、General Query Log、またはネットワークレベルのパケットトレース (tcpdump によるものなど) を使用して、接続ハンドシェイク中に実行されたステートメントの記録を取得します。接続またはレイテンシーの問題が実際に発生しているかどうかは別にして、管理者はデータベースドライバーの内部動作を十分に理解している必要があります
- 個々のハンドシェイクステートメントについて、その目的と、その接続で後から実行するクエリへの影響を説明できることが求められます
- 個々のハンドシェイクステートメントは、少なくとも 1 回のネットワークラウンドトリップを必要とし、セッション全体のレイテンシーを増やす要因となります。実際の作業を実行しているステートメントの数に比べてハンドシェイクステートメントの数が多いと思われる場合は、無効化できるハンドシェイクステートメントがないかどうか判断してください。接続プーリングを使用して接続ハンドシェイクの数を減らすことを検討してください

負荷分散とリーダーエンドポイント

リーダーエンドポイントにはすべての Aurora レプリカが含まれているため、新しい接続に対して DNS ベースのラウンドロビン負荷分散を利用可能です。リーダーエンドポイントに接続するたびに、ラウンドロビン方式で選択された接続可能なインスタンス IPアドレスを取得します。

DNS 負荷分散は (個別のクエリのレベルではなく) 接続レベルで機能します。解決ごとに異なるインスタンスIPアドレスを取得するには、DNS をキャッシュせずにエンドポイントを解決し続ける必要があります。エンドポイントを 1 回だけ解決してから接続をプールに保持すると、その接続でのすべてのクエリは同じインスタンスに送信されます。DNS をキャッシュする場合、エンドポイントを解決するたびに同じインスタンス IPアドレスを取得します。

ベストプラクティスに従わない場合、以下のような問題が起きる可能性があります。

- 1 つの Aurora レプリカがトラフィックの大半または全部を受信する一方、他の Aurora レプリカはアイドル状態のままというように、Aurora レプリカの使用率に偏りが生じる
- Aurora レプリカを追加またはスケーリングした後、そのレプリカがトラフィックを受信しないか、想定外の長い遅延を挟んでトラフィックを受信し始める
- Aurora レプリカを削除した後も、アプリケーションがそのインスタンスにトラフィックを送信し続ける

詳細については、DNS エンドポイントと DNS キャッシング に関する前のセクションを参照してください。

耐障害性と迅速なリカバリのための設計

大規模データベースの運用では、統計上、接続の中断やハードウェア障害などの問題が発生しやすくなります。DB インスタンスのスケーリング、追加、削除、ソフトウェアのアップグレードの実行など、運用作業をより頻繁に行う必要もあります。

この課題に対処する唯一のスケーラブルな方法は、問題や変更が発生するという前提でアプリケーションを設計することです。

例:

- Aurora MySQL では、プライマリインスタンスで障害が発生したことを検出した場合、新しいプライマリインスタンスを昇格させてそれにフェイルオーバーします。これは通常、30 秒以内に完了します。変更を迅速に、手動での介入なしに認識できるようにアプリケーションを設計する必要があります
- Aurora DB クラスターに追加の Aurora レプリカを作成した場合、アプリケーションは新しい Aurora レプリカを自動的に認識してそれらにトラフィックを送信する必要があります
- DB クラスターからインスタンスを削除した場合に、アプリケーションがそれらに接続を試みないようにする必要があります

アプリケーションを徹底的にテストし、アプリケーションがデータベースイベントにどのように反応するべきかを想定したリストを準備します。次に、実験によって想定を検証します。

ベストプラクティスに従わないと、(フェイルオーバー、スケーリング、ソフトウェアのアップグレードなどの) データベースイベントの結果として、予想よりも長いダウンタイムが発生する可能性があります。たとえば、(DB クラスターのイベント通知によると) フェイルオーバーにかかった時間は30秒にもかかわらず、それよりもずっと長い時間アプリケーションがダウンしたままという状況になることがあります。

サーバー設定

このホワイトペーパーの目的上、主なサーバー設定変数の中で言及する価値があるのは max_connections と max_connect_errors の 2 つです。

設定変数 max_connections

設定変数 max_connections は、Aurora DB インスタンスあたりのデータベース接続数を制限します。ベストプラクティスは、各インスタンスで開くと想定される最大接続数よりわずかに大きな値に設定することです。

performance_schema も有効にしている場合は、特に慎重に設定してください。パフォーマンススキーマのメモリ構造は、max_connections などにサーバー設定変数に基づいて自動的にサイズが決定されます。変数に大きな値を設定するほど、パフォーマンススキーマが使用するメモリが増加します。極端なケースでは、このことが原因で、スモールインスタンスタイプでメモリ不足の問題が発生する可能性があります。

T2 および T3 インスタンスファミリーに関する注記

メモリが 8 GB に満たない T2 および T3 Aurora DB インスタンスでパフォーマンススキーマを使用することは推奨されません。T2 および T3 インスタンスでメモリ不足の問題のリスクを減らすには:

- パフォーマンススキーマを有効にしない
- パフォーマンススキーマを使用する必要がある場合、max_connections をデフォルト値のままにする
- max_connections をデフォルト値よりかなり大きい値に増やす予定がある場合、パフォーマンススキーマを無効にする

max_connections 変数の詳細については、MySQL リファレンスマニュアルを参照してください。[7]

設定変数 max_connect_errors

設定変数 max_connect_errors は、特定のクライアントホストからの接続要求の中断を連続何回まで許容するかを決定します。クライアントホストが接続試行の連続失敗の閾値を超えた場合、サーバーはそのホストをブロックします。
そのクライアントからさらに接続を試みると、次のエラーになります。

> Host '**host_name'** is blocked because of many connection errors. (ホスト 'host_name' は接続エラーが多いためブロックされています。)Unblock with 'mysqladmin flush-hosts' (ブロックを解除するには 'mysqladmin flush-hosts' を実行してください)

一般的な (しかし間違った) 対策は、パラメータを非常に大きな値に設定してクライアント接続の問題を回避するというものです。以下の理由から、この対策は推奨されません。

- 根本原因を特定して解決するのではなく、接続の問題をアプリケーション所有者が許容するように仕向けている。接続の問題はアプリケーションの正常性に影響する可能性があるため、無視せずに解決する必要があります。
- 誰かが積極的にサーバーへの侵入を試みているといった本当の脅威を見えなくしてしまう可能性がある。

「ホストがブロックされている」というエラーが発生した場合、max_connect_errors 変数の値を大きくすることは正しい対応ではありません。代わりに、aborted_connects 状態変数と host_cache テーブルでサーバーの診断カウンターを調べます。次に、その情報を使用して、接続の問題が発生しているクライアントを特定し、修正します。skip_name_resolve が 1 (デフォルト) に設定されている場合、このパラメータは効果がないことにも注意してください。

以下の詳細については、MySQL リファレンスマニュアルを参照してください。

- Max_connect_errors 変数[8]
- "Host is blocked" エラー[9]
- Aborted_connects 状態変数[10]
- Host_cache テーブル[11]

まとめ

接続管理のベストプラクティスを理解して実施することは、スケーラビリティを向上させ、ダウンタイムを短縮し、アプリケーションレイヤーとデータベースレイヤーの円滑な統合を確実にするために不可欠です。このホワイトペーパーで提示している推奨事項の多くは、エンジニアリング的な作業をほとんど、または一切伴わずに適用できます。

このホワイトペーパーで提供しているガイダンスは、Aurora MySQL DB クラスターを使用して現在デプロイしている、また将来デプロイするアプリケーションの改善に役立つはずです。

寄稿者

この文書の寄稿者は次のとおりです。

- Szymon Komendera、データベースエンジニア、Amazon Aurora

その他の資料

詳細については、以下を参照してください。

- Aurora on Amazon RDS ユーザーガイド[12]
- MySQL リファレンスマニュアルの「Communication Errors and Aborted Connections」。[13]

ドキュメントの改訂

日付	説明
2019 年 3 月	「はじめに」「DNS エンドポイント」「サーバー設定」の各トピックの内容の軽微な更新。
2018 年 1 月	初版発行

注記

[1] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Overview.html

[2] https://mariadb.com/kb/en/the-mariadb-library/failover-and-high-availability-with- mariadb-connector-j/#specifics-for-amazon-aurora

[3] https://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html#sysvar_wait_timeout

[4] https://dev.mysql.com/doc/refman/5.6/en/innodb-autocommit-commit-rollback.html

[5] https://dev.mysql.com/doc/refman/5.6/en/innodb-trx-table.html

[6] https://dev.mysql.com/doc/refman/5.6/en/innodb-metrics-table.html

[7] https://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html#sysvar_max_connections

[8] https://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html#sysvar_max_connect_errors

[9] https://dev.mysql.com/doc/refman/5.6/en/blocked-host.html

[10] https://dev.mysql.com/doc/refman/5.6/en/server-status-variables.html#statvar_Aborted_connects

[11] https://dev.mysql.com/doc/refman/5.6/en/host-cache-table.html

[12] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Aurora.html

[13] https://dev.mysql.com/doc/refman/5.6/en/communication-errors.html