Mobile Link 12 シンクロナイゼーションのパフォーマンスとチューニング

Principal Author

Graham Hurst

March 2012

last modified by Jeff Albion on Feb 24, 2015

この文書は、日本訳版です。オリジナルはこちらをご参照ください。

https://wiki.scn.sap.com/wiki/display/SQLANY/Performance+and+Tuning+-+MobiLink

概要

Mobile Link は、クライアントに配備されたデータベースと中央のデータベース（統合データベースと呼びます）とのスケーラブルかつ高速・ハイパフォーマンスなデータシンクロナイゼーションを実現する製品です。Mobile Link シンクロナイゼーションのパフォーマンスにおいてキーとなる要素は以下のとおりです。

- 統合データベース側の同時（concurrent）シンクロナイゼーションスクリプト実行時のパフォーマンス
- ネットワークの帯域幅
- クライアントの処理速度
- Mobile Link サーバーが稼働するコンピューターの処理速度
- 十分な数の Mobile Link データベースワーカースレッド （それぞれが統合データベースと接続）
- Mobile Link サーバーによる過大なスワップを避けるための十分なキャシュとコンピューターメモリー

Mobile Link シンクロナイゼーションのパフォーマンスについてまとめると以下のとおりです。

1. Mobile Link サーバーのハードウェア要件は小さく、複数のプロセッサーを効果的に使用します。例えば、SAP SQL Anywhere を統合データベースとした当社のテストにおいては、Mobile Link が必要とするハードウェアの処理能力は通常 SAP SQL Anywhere 統合データベースが必要とするハードウェアの処理能力の半分未満でした。
2. Mobile Link サーバーは、シンクロナイゼーション対象のリモートデータベースが増加しても非常に効果的にスケールします。
3. シンクロナイゼーションスクリプトの同時（concurrent）パフォーマンスをチューニングするには、負荷の高いシンクロナイゼーション条件でテストする必要があります（例えば、インデックスが貼られたクエリーを使用したり、ロックを最小限にしたりするなど）。これは、Mobile Link データベースワーカースレッド数の自動調整の範囲を選択するのに役に立ちます。
4. シンクロナイゼーション毎のオーバーヘッドもあるため、小規模のシンクロナイゼーションではスループットは低めになります。
5. ネットワークプロトコルの選択が、パフォーマンス、特にダウンロードで影響する場合があります。実施したテストでは、HTTP は TCPIP とほぼ同速でしたが、圧縮が使用されている時に暗号化が発生する場合には HTTP がベストなパフォーマンスを示しました。
6. Mobile Link の状態情報は、Mobile Link サーバーではなく、統合データベース側でもそれ以外のデータベースでもメンテナンスすることが可能なため、複数の Mobile Link サーバーを使用することができます。高可用性やスケーラビリティーには、Relay Server やサードパーティー製のロードバランサーとともに使用し、複数の Mobile Link サーバーを単一のサーバーのように使用することもできます。

パフォーマンスの例として、SAP SQL Anywhere 12 統合データベースで、クライアント数 10,000、各 92 バイトの行を 1,000 シンクロナイゼーションした場合の（計 1,000 万行シンクロナイゼーション）総サーバータイムとシンクロナイゼーションレートの最高結果を下に示します。Mobile Link サーバーと統合データベースの稼働環境は以下のとおりです。

Dell PowerEdge R510 サーバー（どちらも）、Xeon X5650 2.6 GHz プロセッサー（6コア x 2）、RAM 48 GB （どちらも専用サーバー）

統合 DB からのダウンロード	21.3 秒 または 469 シンクロナイゼーション/秒 (41 MB/秒)
統合 DB への挿入のアップロード	33.2秒 または 301シンクロナイゼーション/秒 (26 MB/秒)
統合 DBへの削除のアップロード	57.8秒 または 173シンクロナイゼーション/秒 (15 MB/秒)
更新のアップロード（競合検出）	88.1秒 または 114シンクロナイゼーション/秒 (10 MB/秒)

上記の情報の詳細については、この文書の後半で解説します。

はじめに

複数のクライアント側データベースと中央の一つのデータベース（統合データベースと呼びます）との効率的でスケーラブルなシンクロナイゼーションを実現するために Mobile Link シンクロナイゼーションテクノロジは開発されました。Mobile Link は SAP SQL Anywhere のパッケージに含まれています。クライアント側データベースは、SAP SQL Anywhere データベースでも Ultra Light でも利用可能です。サポートしている統合データベースは、SAP SQL Anywhere、SAP HANA、SAP Adaptive Server Enterprise、SAP IQ、Microsoft SQL Server、Oracle、MySQL、IBM DB2 LUWなどです。

この文書のゴールは、Mobile Link サーバーパフォーマンスの理解と、データシンクロナイゼーションのニーズへの適用性の評価支援です。最初に、Mobile Link シンクロナイゼーションのステップについて説明します。

テスト方法を簡単に説明した後、以下の条件を変更した場合のパフォーマンス結果と分析について解説します。

- Mobile Link データベースワーカースレッドの数と Mobile Link データベースワーカースレッド数の自動調整
- 非常に多くの数のクライアントデータベースをシンクロナイゼーションする場合のリトライ間隔と接続制限
- 同時（simultaneously）にシンクロナイゼーションするクライアントデータベース数
- 各シンクロナイゼーションで送信されるデータ量
- Mobile Link サーバーへクライアントデータベースを接続するために使用されるネットワークプロトコルのタイプ

その後、Mobile Link シンクロナイゼーションでベストなパフォーマンスを得るための推奨方法について解説します。

以前の Mobile Link パフォーマンスホワイトペーパーとの違い

今回 Mobile Link と SAP SQL Anywhere のバージョン12 を使用しただけでなく、Mobile Link 11 パフォーマンスホワイトペーパーで実施したベンチマークテストからいくつか変更しました。変更点の一部は以下のとおりです。

- 一般的な使用方法を反映させ、Mobile Link サーバーと統合データベースを同一のコンピューターではなく異なるコンピューター上で稼働させました。
- より新しいサーバーコンピューターを使用しました。
- Windows Server 2003 ではなく Windows 2008 R2 を使用したため、以前の OS で可能だった多くの同時ネットワーク接続を可能にするレジストリ設定が、この OS ではサポート対象外でした。
- upload_fetch スクリプトと競合解決スクリプトを使用せず、アップロードされた更新の競合検出と解決にストアドプロシージャーを使用しました。これは upload_update スクリプト内のアップロードされた更新の競合検出と対処における我々の現在の推奨方法に従ったものです。これにより、競合検出において upload_fetch スクリプトを使用するよりもより高速なパフォーマンスを実現することが可能です。
- Mobile Link サーバーの新機能であるデータベースワーカースレッド数の自動調整機能と手動チューニングとのパフォーマンスを比較するためのテストを実施しました。
- クライアントデータベースと Mobile Link サーバー間の異なるネットワークプロトコルオプション毎のパフォーマンステストを実施しました。
- キャッシュサイズを明示的に設定するのではなく、Mobile Link サーバーの新機能であるキャッシュサイズの自動調整機能を使用しました。また、全体を通してデフォルトの物理メモリーの 70% までを使用する設定を使用しました。
- 現在の Mobile Link サーバーでは、リモートスキーマを自動的にキャッシュするため、シンクロナイゼーション毎のオーバーヘッドは低減されています。

Mobile Link シンクロナイゼーションの処理ステップの解説

Mobile Link サーバーパフォーマンスの理解を支援するため、まず最初に Mobile Link サーバーアーキテクチャーを簡単に説明し、単一の Mobile Link シンクロナイゼーションの詳細ステップについて解説します。ここでは、同期にかかるタイムにそれぞれがどう貢献しているのかということにフォーカスします。その後、より多くのクライアント数にスケールさせ、潜在的なボトルネックを特定します。

Mobile Link シンクロナイゼーションの主なステップは以下のとおりです。

1. クライアントがシンクロナイゼーションを開始し、アップロードストリームを Mobile Link サーバーに送る
2. Mobile Link サーバーがアップロードストリームを処理し、統合データベースで実行するコマンドを作成
3. データベースワーカースレッドとデータベース接続にシンクロナイゼーションをアサイン
4. 統合データベースにアップロードデータを適用
5. ダウンロードするローを統合データベースからフェッチ
6. データベースワーカースレッドとデータベース接続を解放
7. Mobile Link サーバーがダウンロードストリームを準備し、Mobile Link クライアントへ送信。Mobile Link クライアントがこれをクライアントデータベースへ適用。デフォルトのダウンロード確認 (Acknowledgement) なしを使用したシンクロナイゼーションのステップの場合はこれで終了
8. ダウンロード確認 (Acknowledgement) が有効な場合、Mobile Link クライアントはダウンロード確認 (Acknowledgement) を Mobile Link サーバーに送信。再度データベースワーカースレッドとデータベース接続にシンクロナイゼーションがアサインされ、確認 (Acknowledgement) が処理された後、データベースワーカースレッドとデータベース接続を解放

Mobile Link サーバーのアーキテクチャー

Mobile Link サーバーのアーキテクチャーは、SEDA (staged event-driven architecture) を拡張したものです。シンクロナイゼーションは、各ステージで少なくとも一つのスレッドを実行し、異なるステージにパススルーします。このアーキテクチャーは、データベースアクティビティーからネットワークアクティビティーを独立させることでシンクロナイゼーションスループットを最大化し、各シンクロナイゼーションが統合データベースへの接続に必要とするタイムを最短化するよう設計されています。

スタート

シンクロナイゼーションは、通常 Mobile Link サーバーとネットワーク接続を確立するクライアント側から開始します。Mobile Link サーバー側で開始するシンクロナイゼーションもオプションとして用意されていますが、クライアントに対してシンクロナイゼーションをスタートさせる通知メカニズムが異なるだけです。イベントの順序はクライアントの種類に依存します。Ultra Light クライアントは、まず Mobile Link とネットワーク接続を確立し、それを送信する時にアップロードストリームを構築します。一方 SAP SQL Anywhere クライアントはネットワーク接続を確立する前にアップロードストリームを構築します。このアップロードストリーム構築の違いにより、Ultra Light クライアントはより少ないメモリーで機能します 。

クライアントはまず Mobile Link サーバーとのネットワーク接続をリクエストします。Mobile Link サーバーは、ネットワークステージスレッドを使用して即時にネットワーク接続を受けつけます。あるいは、接続制限に達している場合にはリクエストを拒否します。

デフォルトでは、Mobile Link サーバーは 1,024 までのネットワーク接続（-nc オプション）を受けつけます。しかし、この制限に達する前に、OS の設定によってはネットワークエラーが発生する可能性があります。Mobile Link サーバーの制限に達している場合には、追加のシンクロナイゼーションリクエストは拒否され、クライアントはネットワークエラーになります。ネットワークエラーになってしまったクライアントは、しばらくし

た後リトライする必要があります。Mobile Link サーバーがビジーな場合で、たとえスループットが低減される可能性があっても、キューになるよりはシンクロナイゼーションが拒否される方が良い場合にはネットワーク接続制限を低くすることも可能です。ただし、クライアントは、Ultra Light クライアントの場合には最初の部分を、SAP SQL Anywhere クライアントの場合にはすべてのアップロードストリームを再構築して、シンクロナイゼーションをリトライする必要があることに注意してください。

アップロード処理

ネットワークステージスレッドは、ネットワークから Mobile Link サーバーのインメモリーキャッシュへのアップロードストリームのバイトを読むために使用されています。アップロードストリームの送信タイムに影響する要因は、主にクライアントから Mobile Link ネットワークへの接続スピードと、アップロードストリームのサイズです。

Ultra Light クライアントのアップロードフェーズには、どのデータをアップロードする必要があるのか（新規のデータや前回のシンクロナイゼーションから変更があったデータなど）を判断するクライアントの処理タイムが含まれますが、通常、ダウンロードフェーズで必要とされるクライアント処理よりもかなり少ないものです。

SAP SQL Anywhere のクライアントの場合は、クライアントが Mobile Link に接続する前にアップロードストリームが作成されます。そのため、アップロードストリームの作成にかかるタイムは、Mobile Link への接続前に発生します。

Mobile Link サーバーでは、アップロードされたバイトを Mobile Link プロトコルコマンドにデコードするためにストリーム・ステージスレッドを使用します。このステージには、暗号化が有効になっている場合のストリームの暗号化解除と、圧縮が有効になっている場合のストリームの圧縮解除の処理も含まれます。暗号化と圧縮は必要な処理タイムに追加されますが、打合せストリーム圧縮は送信されるバイト数を低減するためネットワーク送信タイムを低減します。このステージからの Mobile Link プロトコルコマンドは、コマンドキューに追加されます。

キュー状態のコマンドはコマンドプロセッサーステージ内のスレッドのプールによって実行されます。多くのコマンドは直接実行されますが、統合データベースが必要なものは、リストに収集されます。シンクロナイゼーションのためのリストが完了すると、シンクロナイゼーションがコマンドプロセッサースレッドにアサインされます。コマンドプロセッサースレッドはデータベース接続を得ることでデータベースワーカースレッドになり、そのシンクロナイゼーションのデータベースコマンドのリストが完了するまでデータベースワーカースレッドを継続します。

同時にデータベースワーカースレッドとして機能できるのは、最大でもコマンドプロセッサースレッドの半分です。コマンドプロセッサースレッドプールのサイズは、Mobile Link サーバーの -wm や -w オプションで指定されたデータベースワーカースレッドの最大数の 2 倍です。そのため、Mobile Link サーバーはデータベースのコマンドとデータベース以外のコマンドを同時に実行することができます。Mobile Link サーバーの負荷によっては、データベースワーカースレッドとして機能するコマンドプロセススレッドの半分よりも少ない可能性もあります。

データベース接続

シンクロナイゼーションのためのデータベース接続は、そのシンクロナイゼーションにデータベースアクセスが必要になる段階に達してはじめて作成、割り当てられます。

Mobile Link は、Mobile Link システムテーブルからのシンクロナイゼーションスクリプトの読み込みや他の管理タスクのための 1 以上のシェアード接続だけでなく、データベースワーカースレッドで使用するために、統合データベースへの接続のプールを維持します。残りのデータベース接続は、必要に応じてデータベースワーカースレッドによって作成されます。データベースコマンドリストが完了するまでは、データベースワーカースレッド専用に 1 接続使用され、その後データベース接続プールに返されます。データベースワーカースレッド数よりもかなり大きい数のデータベース接続数をこのプールに指定することができますが、一度に使用される最大数は、ワーカースレッドごとに 1 接続です。

Mobile Link が、接続プールから接続をクローズし、新しい接続をオープンするケースが 2 つあります。1 つ目は、エラーが発生した場合です。2 つ目は、クライアントがシンクロナイゼーションスクリプトバージョンをリクエストしたにも関わらず、利用可能な接続がそのシンクロナイゼーションスクリプトバージョンを使用しなかった場合です。シンクロナイゼーションスクリプトのバージョンが 2 つ以上ある場合には、異なるシンクロナイゼーショ

ンバージョンがリクエストされるたびに毎回 Mobile Link がデータベース接続をクローズして新たにオープンしなくてもよいよう、プールされた接続の最大数は十分な数で設定した方が良いかもしれません。

新規データベース接続の作成にかかるタイムを最短化するには、以下を行うことが可能です。

- データベースワーカースレッドの数を制限する (-wm と -w オプション)。(スループットの最適なデータベースワーカースレッド数の選択の詳細については、後で説明します)
- プールされた Mobile Link データベース接続の最大数をシンクロナイゼーションスクリプトバージョン数 × Mobile Link データベースサーバースレッド数に設定（-cn オプション）する。これにより新しいデータベース接続を作成する必要性を避けることができます。なぜならば、各データベースサーバースレッドで各バージョン 1 接続利用可能になるからです。

アップロードの適用

1 シンクロナイゼーションが 1 データベースワーカースレッドに割りふられると、そのスレッドはコマンドキューからデータベースコマンドを実行します。最初に、単純なクエリーを使用して、データベース接続がまだアクティブなのかチェックし、認証を行います。

その後、Mobile Link データベースワーカースレッドは、コマンドキュー内のアップロードデータベースコマンドを実行します。Mobile Link ワーカースレッドは、データベース接続を使用し、アップロードスクリプトを実行してアップロードされたローを統合データベースに適用します。デフォルトでは、アップロードされたテーブルスクリプトはローがアップロードされた場合にのみ実行されます。Mobile Link データベースワーカースレッドには、例えば同時（simultaneous）接続や、トランザクションのサイズ、同時性（concurrency）の問題、そして、統合データベースが Mobile Link と別のコンピューター上にある場合には、ネットワークのスピードなど、統合データベースとトランザクション処理を実行するクライアントと同様のパフォーマンスに関する問題が存在します。さらに、アップロードスクリプトが upload_fetch スクリプト経由の競合検出を包含する場合、Mobile Link サーバーは競合をチェックするため、更新を適用する前に統合データベースから更新すべきそれぞれのローをフェッチします。より高速なパフォーマンスを実現するには、aupload_fetch スクリプトと競合解決スクリプトを定義するのではなく、upload_update スクリプト内に競合検出と解決を実装することを推奨します。

コマンドキュー内のアップロードデータベースコマンドのリストが完成した時点で、データベースワーカースレッドはダウンロードデータベースコマンドをスタートさせます。

ダウンロード

データベースワーカースレッドは、ダウンロードデータベースコマンドを実行し、シンクロナイゼーションします。これらのコマンドは Mobile Link クライアントにダウンロードされるローをフェッチすると同時に統合データベース内のダウンロードスクリプトを実行します。フェッチされたローは、Mobile Link サーバーのキャッシュに格納されます。文字コードセットの翻訳などのデータコンバージョンも全て実行されます。アップロードに関しては、ダウンロードスクリプトを実行する際の Mobile Link データベースワーカースレッドには、他のデータベースクライアントアプリケーションと同様のパフォーマンス問題が存在します。**シンクロナイゼーションのたびに実行されるため、ダウンロードスクリプトクエリーにはインデックスを使用し、ロックを避けることが非常に重要です！**

フェッチングやデータコンバージョンコマンドの完成後、データベースコマンドのリストが完成します。ダウンロードトランザクションはコミットされ、終了シンクロナイゼーションスクリプトを実行、コミットし、データベース接続はデータベース接続プールにまた解放され、スレッド（データベースワーカースレッドだったもの）は、コマンドプロセッサースレッドのプールに返されます。

コマンドプロセッサースレッドは、その後継続しているダウンロードコマンドを実行してシンクロナイゼーションし、同時にダウンロードストリームフォーマットにダウンロードするローをパッキングします。ダウンロードストリームがメモリー上に作られると、ネットワークスレッドを使用して Mobile Link クライアントに送信されます。

ダウンロードクエリーがチューニングされていると仮定すると、通常ダウンロードフェーズにおける Mobile Link サーバーおよび統合データベースでの処理は、アップロードフェーズよりも少なめになります。

ダウンロードストリームをクライアントに送信するのにかかるタイムはダウンロードのサイズ、ネットワーク帯域、そしてクライアントの処理スピードに依存します（Ultra Lightクライアントの場合。SAP SQL Anywhereクライアントの場合はダウンロードがダウンロードバッファーに合わない場合）。

Mobile Link サーバーとは異なり、クライアントはアップロードフェーズよりもダウンロードフェーズの方がより多くの処理を行います。Ultra Light クライアントはダウンロードストリームを受信するとそれを処理します。一方、SAP SQL Anywhere Mobile Link クライアントはダウンロードストリームをバッファーし、それを処理します。ダウンロードストリームを処理すると、除くべきローを削除、新しいローを挿入、変更されたローを更新、そしてインデックスを更新し、参照整合性をチェックします。この処理には、低速のプロセッサーやフラッシュメモリーのような低速のストレージメディアのクライアントの場合、長タイムを要する可能性があります。

クライアント側のダウンロード確認（acknowledgement）が有効になっている場合、クライアントがダウンロードストリームを受信、処理、コミットすると、Mobile Link サーバーにダウンロード確認（acknowledgement）を送信します。そして、データベースワーカースレッドを取得し、non-blocking ダウンロード確認（acknowledgement）スクリプトを実行します。

ベストのスループットとスケーラビリティーが得られるのは、デフォルトの設定で、ダウンロード確認（acknowledgement）を有効にしていない場合です。

より多くの数のクライアントにスケールアップ

単一のシンクロナイゼーションにおけるステップについてタイムをかけて解説したので、次は複数のクライアントを同時（simultaneously）にシンクロナイゼーションするケースについて考察します。

Mobile Link サーバーが同時 (simultaneous) にシンクロナイゼーションできる最大数は、ネットワーク接続の制限で設定します (-nc オプション) 。なぜならば、各シンクロナイゼーションは全てのシンクロナイゼーション期間のネットワーク接続を必要とするからです。サーバーのネットワーク接続が最大数に達している場合、追加のシンクロナイゼーション接続リクエストは全て拒否されます。クライアント側が低速なコンピューターまたはネットワークの場合、ネットワーク送信ステージにおいてより多くのシンクロナイゼーションを同時（simultaneously）に行うため、ベストのスループットを得るには、デフォルトの 1,024 よりも多くの接続制限を使用する必要があるかもしれません。しかしながら、接続リクエストは、特に制限数を増やした場合、同時 (simultaneous) シンクロナイゼーション数が Mobile Link 接続制限に達する前に、OS から拒否される可能性があります。

データベースステージで同時（simultaneously）に存在できるアクティブなシンクロナイゼーションの最大数は、データベースワーカースレッドの最大数で設定します (-wm または -w オプション)。 最適な値は統合データベースに依存します。つまり、これはコンスタントにシンクロナイゼーションスクリプトを実行することのできる最適なデータベース接続数に相当します。実際には、この数は他の共通のデータベースアプリケーションのための数よりも少くなります。なぜならば、Mobile Link シンクロナイゼーションは、多くのクライアントデータベース、凝縮した、同時 (simultaneous) シンクロナイゼーションに対して長い時間をかけて行われるワークに注力するからです。

データベースが多くの同時接続をサポートできる一方で、ビジーな接続の最適数は通常それよりもかなり小さくなります。例えば、CPU-bound なデータベースロードでは、その負荷を課している接続の最適数は CPU コア数に密接に関係しています。

お客様の現金の入出庫にわざわざ出納係が金庫に行かなければならない旧式の銀行を例に類似点を考えてみます。お客様はシンクロナイゼーション、出納係は Mobile Link サーバーのデータベースワーカースレッド、そして、金庫は統合データベースのようなものです。出納係が増えれば、一度に対応できるお客様数は増えますが、金庫にいる出納係が多すぎるとお互いの邪魔になってしまいます。つまり、出納係が多すぎると、金庫にいる出納係の人数が少ない場合よりもトランザクション率は小さくなります。

銀行は金庫をより大きなものにすることも可能ですが、それぞれの金庫の出納係数には最適な数があります。それぞれの金庫に最適な出納係数よりもあまりにも多すぎるあるいは少なすぎる場合、混雑のしすぎによりトランザクション率を下げてしまい、金庫を十分に活用できないこと

になります。

さらに広げて考えると、この例の銀行で対応できるお客様の数は、Mobile Link の接続制限のようなものです。なぜならば、それぞれのお客様は、払出伝票や預金伝票を記入したり、トランザクションを記録したり、引き出した現金をしまったりする時間を必要とするため、銀行の出納係の人数よりも多くのお客様に対応できるのは、理にかなっていると言えるでしょう。

潜在的なボトルネック

Mobile Link シンクロナイゼーションのスループットを含め、あらゆるシステムの全体パフォーマンスは、通常そのシステム内のある一つのポイントのボトルネックで制限されます。Mobile Link サーバーのシンクロナイゼーションスループットの制限には、以下のボトルネックが考えられます。

統合データベースのパフォーマンス - Mobile Link で特に重要なのは、Mobile Link データベースワーカースレッドの数と同程度の接続数で、他のデータベースロードと、Mobile Link のスクリプトを同時 (simultaneously) に実行できる統合データベースのスピードです。ベストなスループットを得るには、シンクロナイゼーションスクリプトのデータベースの競合を削減するため多くのテストと熟練した チューニングが必要となる可能性がありますが、これは、Mobile Link のパフォーマンスで最もよくみられるボトルネックではありません。なぜならば、統合データベースに集中する負荷を生成するアプリケーションはほとんどないからです。

クライアントから Mobile Link サーバーへの通信帯域 - 旧世代の携帯電話ネットワークやダイアルアップネットワークなどの低速の接続では、Mobile Link クライアントと Mobile Link サーバーにおいてデータの送信待機が発生する可能性あります。

Mobile Link から統合データベースへの通信帯域 - これは、Mobile Link サーバーと統合データベースが同じコンピューター上で稼働している場合、あるいは高速ネットワークで接続されている別々のコンピューターで稼働している場合にはボトルネックになることはほとんどありません。

Mobile Link サーバーが稼働するコンピューターの処理速度 - .NET あるいは Java のシンクロナイゼーションスクリプトを使用している場合、これがより重要になります。なぜならば、SQL シンクロナイゼーションスクリプトが統合データベースコンピューター上で実行されるのに対し、これらは Mobile Link サーバーコンピューター上で実行されるからです。

Mobile Link データベースワーカースレッド数 - データベースワーカースレッド数が少ない場合、統合データベースにおける競合発生の可能性は低くなりますが、データベースワーカースレッド数が少なすぎる場合には、クライアント側がフリーのデータベースワーカーを待ち続けることになる、あるいは統合データベースを十分に活用できないことになります。逆に、データベースワーカースレッド数が大きすぎる場合には、競合が発生し、スループットは低下します。Version 12 では、Mobile Link サーバーが自動的にデータベースワーカースレッド数を調整するオプションがあります。とはいえ、自動調整の最適範囲を選択する上では、パフォーマンステストが役に立ちます。

クライアントの処理速度 - これは、アップロードよりもダウンロードでボトルネックになりがちです。なぜならば、ダウンロードでは、より多くのクライアント処理と永続ストレージへの書き込み処理が行われるからです。Mobile Link サーバーは、ダウンロードを送信した後は、クライアントを待つ必要はありません。

パフォーマンステスト

Mobile Link 特有のものではない潜在的なボトルネックはこのホワイトペーパーのカバー範囲ではありません。帯域幅の影響や一般的なクライアントサーバー型の統合データベースのパフォーマンスは考慮していません。

- 同時（simultaneously）に大多数のクライアントをシンクロナイゼーションした時の Mobile Link のパフォーマンス
- 特定のワークロードのスループットを最大化するための Mobile Link データベースワーカースレッドの最適数
- シンクロナイゼーションリクエストが Mobile Link サーバーネットワーク接続制限を超えた時の最適なリトライ間隔と制限増の影響
- 同時（simultaneously）にシンクロナイゼーションするクライアント数の変化の影響
- 各シンクロナイゼーションで送信されるデータ量の変化の影響
- Mobile Link サーバーのハードウェア要件と統合データベースとの相関

これらの結果の解釈を容易にするため、以下の方針でテストを行いました。

1 度に 1 つのパラメーターを変更 – Mobile Link シンクロナイゼーションのスループットに影響する変数は複数あるため、できる限り一度に変える変数を 1 つだけにするようにしました。最適なスループットに達するまでは、1 つのパラメータのみ変更し、そのパラメーターを固定して、別のパラメーターを変更しました。

Mobile Link サーバーに対してストレス – ここでは Mobile Link サーバーのパフォーマンスに注力しているため、Mobile Link サーバーに対して確実にストレスがかかり、統合データベースの負荷は最小限にする選択をする必要がありました。Mobile Link サーバーと統合データベースサーバーを理想的なコンピューター上で稼働させ、クライアントは別のコンピューターで稼働させました。十分な同時 (simultaneous) シンクロナイゼーションを実行することで、サーバーコンピューターの CPU 利用率を 100% に近い状態に保つようにしました。Mobile Link サーバーと統合データベースサーバーを理想的なコンピューターで稼働させることで、それぞれの CPU 利用状況から相対ハードウェア要件を推測するできるようにしました。

シンプルに保つ – Mobile Link サーバー固有のパフォーマンスに注力するため、シンクロナイゼーションスクリプト、クライアントアプリケーション、そしてデータベーススキーマは全てシンプルなものしました。スキーマとシンクロナイゼーションスクリプトが複雑になると統合データベースの負荷が増すことになります。そのため、これらをシンプルに保つことで、Mobile Link サーバーの相関ワークロードを最大にしました。

以下のセクションでは、パフォーマンステストの特定の側面を解説します。

データベーススキーマ

テストは全て、単一のテーブルのデータのシンクロナイゼーションに基づいて行いました。また、カラムの型は、よく使用されるデータ型を使用する選択をしました。以下のSQL 文は、テーブル定義を示します。

```
CREATE TABLE Purchase (
    emp_id      INT           NOT NULL,
    purch_id    INT           NOT NULL,
    cust_id     INT           NOT NULL,
    cost        NUMERIC(30,6) NOT NULL,
    order_date  TIMESTAMP     NOT NULL,
    notes       VARCHAR(64),
    PRIMARY KEY ( emp_id, purch_id )
);
```

複数クライアント間のデータのパーティショニングをシンプルにするため、合成プライマリーキーを使用しました。最初のカラムはそのローがどのクライアントと関連しているかを示します。複数クライアント間のローを共有しないことでデータベースの競合を防ぎました。これはデータベースサー

バーの負荷を最小限にし、Mobile Link サーバーの相関ワークロードを増やすのに役立ちます。

値

Purchase テーブルを埋めるデータの値は、クライアントまたは統合データベースのどちらかで生成します。この値を生成するアルゴリズムによって、integer データには大きな値を使用することで、クライアントへまたはクライアントからデータが送信された時に、パッキングスキームがデータを縮めないようにしました。コストの最初の値は、全てのローで、123456789.12です。 それぞれの更新ではコスト値を 1 つずつ増やします。 notes カラムに使用されている全ての値は、正確に 64 文字長としました。これは、それぞれのローで一定のバイト数で確実に送信されるようにするためです。このケースでは、送信されるそれぞれのローで使用されたのは 92 バイトです。

タイミングフレームワーク

パフォーマンステストを行うため、Mobile Link シンクロナイゼーションタイミングのフレームワークを開発しました。フレームワークのコンポーネントは以下のとおりです。

統合データベースへの追加テーブル – 統合データベースへの追加テーブルとして、実行数、実行ごとのクライアント数、各クライアントがすべきことなど、テストを実行するための情報を保持するためのテーブルが 1 つ、各シンクロナイゼーションで 4 つのタイムスタンプ、つまりクライアントと統合データベース双方で記録される開始と終了タイムを格納するタイミング情報のテーブルが 1 つ、それぞれのクライアントにダウンロードする次のデータをトラッキングするためのテーブルが 1 つ、さらに実行情報とタイミングに関係するテンポラリーテーブルとビューなどが存在します。

Mobile Link シンクロナイゼーションスクリプト – シンクロナイゼーションバージョン、あるいはスクリプトのセットをいくつか採用しました。それぞれのバージョンにはクライアントデータベースに定義される相当パブリケーションがあります。 2 つのバージョンがセットアップをコントロールします。1 つはタイム記録の（timed） シンクロナイゼーションのコントロール、もう 1 つは実行後にクライアントが統合データベースに送るクライアントのタイミングです。タイム記録の（timed） シンクロナイゼーションでは、以下のスクリプトがタイム記録（timed）と定義されています。

- begin_connection – 他のスクリプトで使用された接続レベル変数を定義します
- begin_synchronization – サーバーの開始時間を記録します
- upload_delete – アップロードされた削除の適用方法を定義します
- upload_insert – アップロードされた挿入の適用方法を定義します
- upload_update – アップロードされた更新の適用方法を定義します
- download_cursor – ダウンロードするローの数をコントロールします
- end_synchronization – サーバー終了時間を記録します

クライアントアプリケーション – 多数のクライアントが同時（simultaneously）にシンクロナイゼーションすることで Mobile Link サーバーにストレスを与えるため、2 台以上のコンピューター上で複数のインスタンスを同時（simultaneously）に実行できる小規模で効率的なクライアントプログラムが必要でした。それぞれのクライアントのインスタンスが小さなメモリーフットプリントであるよう、クライアントデータベースには Ultra Light を選択しました。効率性を考慮し、Ultra Light へのインターフェースには embedded SQL を使用した C ベースのクライアントを実装しました。ディスクアクセスによってクライアントが低速になるのを避けるため、通常の Ultra Light の動作である最初のシンクロナイゼーションのディスクへのフラッシングは無効にしました。マルチプロセッシングの簡易化のため、それ自身の複数のインスタンスを spawn できる Windows コンソールプログラムを作成しました。最初のインスタンスは、マスタープロセスとして使用します。これはクライアントとして機能する子プロセスを spawn します。このアプリケーションには、特定の間隔後、通信エラーによって失敗するあらゆるシンクロナイゼーションをリトライするオプションを用意しました。

スーパーバイザー アプリケーション – これは、異なるコンピューターで稼働するクライアントをコーディネートするものです。

ゲートの使用を通じて全てのクライアントの歩調が合うようにします。ゲートでは、レースのスタート地点のように、処理前に全てのクライアントが同じ地点に達するのを待ちます。効率性を考慮し、ゲートの実装は OS primitives を使用しました（同じコンピューター上のクライアントの待機には Windows イベントオブジェクトを使用し、コンピューター間のクライアントの待機には Windows named pipes を使用しました）。ゲートの実装は非常に効率的です。全てのクライアントがゲートの後は数秒でスタートします。

クライアントはタイム記録の（timed）シンクロナイゼーションそれぞれの前後でゲートを使用します。シンクロナイゼーション前のゲートでは、全てのクライアントが確実にシンクロナイゼーションの同時開始を試みるようにし、シンクロナイゼーション後のゲートでは、確実に全クライアントシンクロナイゼーションが完了するまでその他のクライアントシンクロナイゼーション処理が発生しないようにします。

タイム記録（timed）されたシンクロナイゼーションは、以下のステップで行われます。

		クライアント	Mobile Link サーバー
	1	シンクロナイゼーションの準備	
⌛	2	他のすべてのクライアントを待機（「ゲート」経由）	
🕓	3	クライアントの開始タイムを記録	
	4	シンクロナイゼーションを開始（Ultra Light シンクロナイゼーション経由）	
🕓	5		（begin_synchronizationスクリプト内で）サーバー開始タイムを記録
	6	シンクロナイゼーションの実行	シンクロナイゼーションの実行
🕓	7		（end_synchronization スクリプト内で）サーバーの終了タイムを記録
🕓	8	クライアント終了タイムの記録	
⌛	9	他の全クライアントを待機（「ゲート」経由）	

テーブル 1：他の全クライアントを待機（「ゲート」経由）

上の表のアイコンは、タイミングに関連した追加のステップを示します。これらの追加ステップは、タイムスタンプの記録や（🕓）、ゲートで他の全てのクライアントを待機すること（⌛）も含まれます。

クライアントタイムはクライアントのメモリー内に記録されます。全てのタイム記録の（timed）シンクロナイゼーションが実行された後で、統合データベースにタイムが送られます。サーバータイムは統合データベースに直接記録されます。

タイミングの結果を議論するにあたり、記録されたタイムスタンプ値からもたらされた以下の数量を参照します。

クライアントシンクロナイゼーションタイム ＝ クライアント終了タイム – クライアント開始タイム

サーバーシンクロナイゼーションタイム ＝ サーバー終了タイム – サーバー開始タイム

総サーバータイム ＝ 最終サーバー終了タイム – 最初のサーバー開始タイム

各クライアントが複数の同じタイプのシンクロナイゼーションを行う場合、同時（simultaneous）シンクロナイゼーションのセットにグルーピングされます。セットになった全てのシンクロナイゼーションは同時に開始され、全てが完了するまで次のセットは開始しません。このケースでは、総サーバータイムの計算に以下の式を使用しました。

$$\text{総サーバータイム} = \sum_{i=1}^{\text{セット数}} \text{セットiの総サーバータイム}$$

言い方を変えれば、同時（simultaneous）シンクロナイゼーションのセットが複数ある場合でも、シンクロナイゼーションに費やされたタイムだけをカウントします（例: ゲート間）。

テスト結果では、総サーバータイム分のシンクロナイゼーション数、のような総サーバータイムが直接のスループット測定のかわりに報告されます。長い値がグラフ内で最も顕著であるようにレートよりもタイムが報告されます。なぜならば、最長のオペレーションを最小化することは、ベストのスループットを得るうえで最も重要だからです。この議論は、スループットを最大化する、または総サーバータイムを最短化するのどちらかも参照することになるかもしれません。なぜならば、これらは同等だからです。

テスト環境

タイミングテストは、以下のソフトウェアとハードウェアを使用して行いました。

テストしたソフトウェアは SAP SQL Anywhere 12.0.1.3554 です。統合データベースは、SAP SQL Anywhere ネットワークデータベースサーバーを、同期サーバーには、Mobile Link サーバーをシンクロナイゼーションに使用しました。クライアントは（すでに述べたように）Ultra Light embedded SQL インターフェースを Ultra Light データベースに使用しました。通常 Mobile Link と SAP SQL Anywhere には、デフォルトのオプションを使用しました。指定がない限りは、暗号化や圧縮なしの TPC/IP 通信プロトコルを使用しました。テストの再現性を改善し、Mobile Link サーバーの負荷を最大化するため、SAP SQL Anywhere データベースは（データベースのあらゆるディスクボトルネックをなくすため）オプショナルのインメモリーモードを使用しました。この結果では、インメモリーオプションによって変化は少ないことが示されました。同様に、変化を低減するため SAP SQL Anywhere のマルチプログラミングレベルの自動調整は無効にしました。 事前のテストでは、デフォルトの最小の 24 (論理 CPU 数) がベスト結果を示したため、これを使用しました。

Mobile Link サーバーと SAP SQL Anywhere は、Dell PowerEdge R510 サーバーコンピューターでそれぞれ別に稼働させました。それぞれのサーバーコンピューターには、Intel Xeon X5650 2.6 GHz プロセッサー 6 コアx 2、RAM 48 GB、Intel Gigabit ET Quad Port ネットワークアダプター、シングル SAS 15K RPM ハードドライブを搭載、ハイパースレッドを有効化し、どちらのサーバーも 24 コアとレポートされた状態にしました。OS は、Windows Server 2008 R2 64bit でしたので、Mobile Link サーバーと SAP SQL Anywhere データベースサーバーとも 64 bit 版を使用しました。

クライアントは 32 bit Windows Server 2003 が稼働する Dell PowerEdge 750 コンピューター最大 12 のラックで稼働。それぞれのコンピューターには 3.2 GHz Pentium 4 プロセッサー、ハイパースレッドを有効化、RAM 4 GB 搭載。全テストにおいて、PowerEdge 750 コンピューター 10 台を使用しました。

これらのクライアントコンピューターは Mobile Link コンピューターにギガビットネットワークスイッチを使用してネットワーク接続し、Mobile Link コンピューターは、異なるポートとギガビットネットワークスイッチ経由で統合データベースサーバーコンピューターにネットワーク接続しました。タイミング実行のため、どちらのスイッチも他のネットワークからは独立させました。

実施したテスト

Mobile Link のパフォーマンスとスケーラビリティーの特徴を評価するため、以下のテストを行うにあたりタイミングフレームワークを使用しました。

1. データベースワーカースレッド数を変更しスループットを最大化するための最適な数を決定。そして自動的に数を変更させた場合と比較。これを 1,000 クライアント同時（simultaneous）のシンクロナイゼーションを使用して実行
2. クライアント数が Mobile Link サーバーネットワーク接続制限を超えた場合と、クライアント数に合うよう制限が増加した場合とで、クライアントのリトライの間隔を変更
3. 同時（simultaneously）にシンクロナイゼーションするクライアント数を変更し、クライアント数が増えるとスループットが落ちるか確認
4. シンクロナイゼーションのサイズを変更し、スループットへの影響を確認
5. クライアントと Mobile Link サーバー間で使用されているネットワークプロトコルを変更し、異なるプロトコル、暗号化、圧縮

を使用する影響を確認

それぞれのケースで、4 つのタイプのシンクロナイゼーションを測定しました。

- 統合データベースからダウンロードされた新しいデータ（挿入）
- 競合検出を有効化した状態でのクライアントデータベースからアップロードされた更新（upload_update スクリプト内）
- クライアントデータベースからアップロードされた削除
- クライアントデータベースからアップロードされた挿入

全てのタイミング実行は同じステップで行いました。まず、新しい統合データベースを作成し、ダウンロードに十分なデータで埋めます。その後、それぞれのクライアントは空のシンクロナイゼーションを 1 回行い、Mobile Link ユーザーテーブルにクライアント名を設定して、リモートスキーマをキャッシュし、確実に正しいシンクロナイゼーションスクリプトのバージョンで全てのデータベース接続が確立されるようにしました。その後タイム記録の（timed） シンクロナイゼーションを以下の順に実行しました。ダウンロード、更新、削除、そして挿入です。これは、4 種のシンクロナイゼーションを行います。この順番により、統合データベースは確実に開始時と同数のローで終了し、Ultra Light データベースはダウンロードされたローより多いローは持ちません。

更新のダウンロードと、削除のダウンロードはテストしていません。ダウンロードでは、挿入と更新のダウンロード間の違いはないからです。Mobile Link サーバーは、download_cursor スクリプトで選択されたローをダウンロードします。そして、クライアントデータベースは内部の効率的な「upsert」を使用して、ダウンロードされたローを適用します。download_delete_cursor スクリプトがプライマリーキーカラムのみを含むと仮定すると、削除のダウンロードの方が高速になる可能性が高くなります。

結果や分析などこれらのテストの詳細は、以降のセクションで解説します。

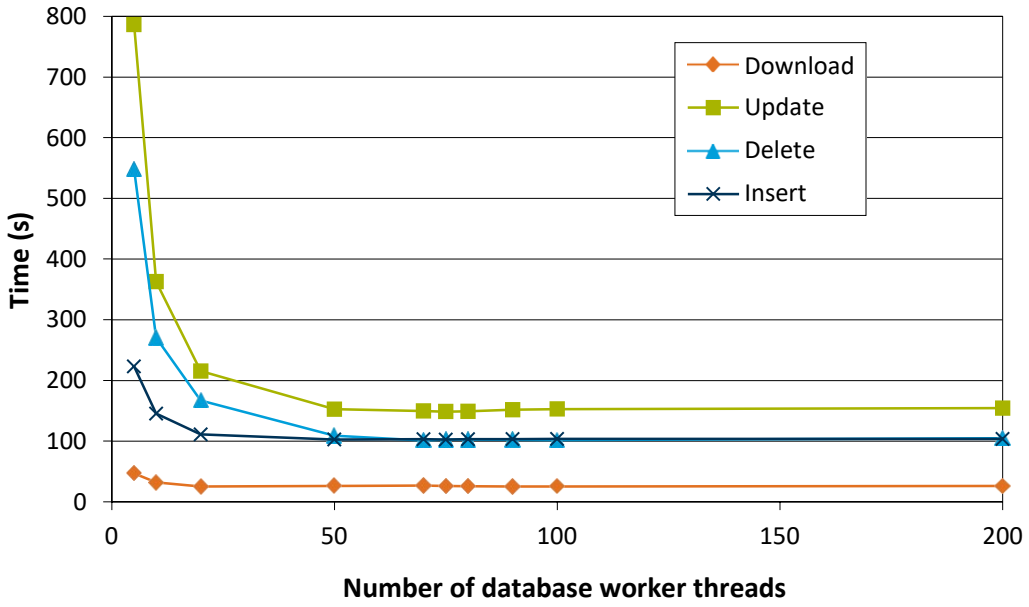
テスト 1：最適な Mobile Link データベースワーカースレッド数

このテストでは、Mobile Link データベースワーカースレッド数を変更しつつ、1,000 の同時 (simultaneous) シンクロナイゼーション クライアントを使用することを選択しました。最初は、自動調整なしに数を変更し、自動調整の範囲を変更しました。Mobile Link サーバーにストレスを与えるため、クライアントは 1,000 使用することを選択し、クライアントがなくなる前にスループットが減少するポイントにヒットするようにしました。また、Mobile Link のデフォルトのネットワーク接続数制限の 1,024 以下にしたかったため、以下の条件を保ちました。

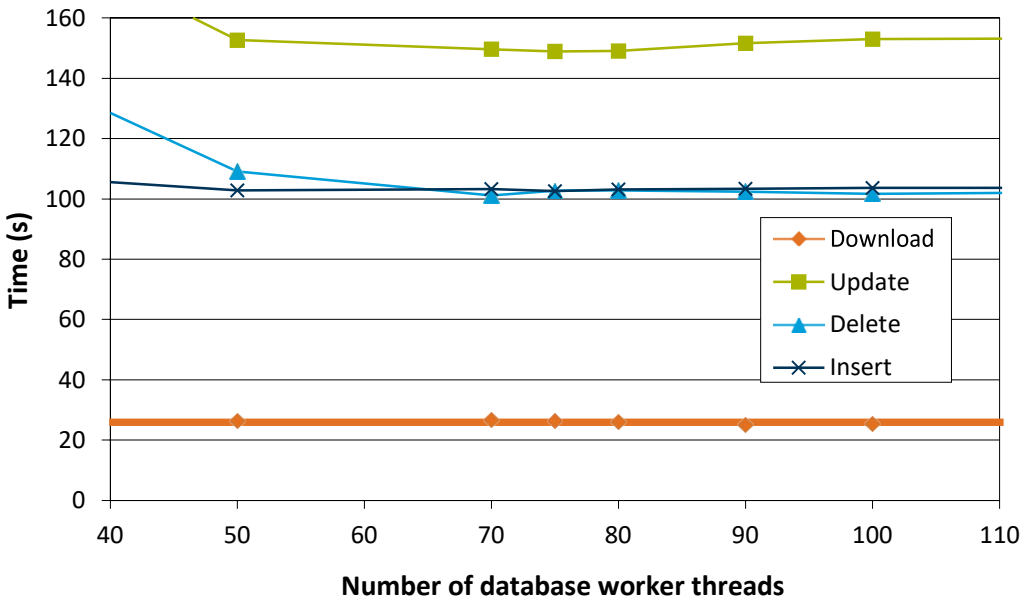
- Pentium 4 3.2GHzプロセッサーのコンピューター10 台上に 1,000 クライアント
- 1 クライアントシンクロナイゼーション毎に 1,000ロー（ロー毎に 92 バイト）
- 各クライアントに 10 セットのシンクロナイゼーション
- 計 10,000 のシンクロナイゼーションと 1,000 万ロー

データベースワーカースレッドの固定数を使用する

自動調整機能を使用しない場合の、異なるデータベースワーカースレッド数の結果は以下のグラフ（概要と詳細）と表のとおりです。



グラフ1：総サーバータイム vs Mobile Link データベースワーカースレッドの固定数、各ポイントは1,000 クライアントシンクロナイゼーションを 10 回



グラフ 2：グラフ 1 のズームインビュー

DB ワーカー	ダウンロード	アップデート	削除	挿入	合計
5	47.0	786.5	547.9	223.2	1604.6
10	31.9	362.9	270.4	145.4	810.5
20	**25.1**	215.5	167.3	111.0	518.9
50	26.5	152.7	109.1	102.8	391.0
70	26.7	149.7	**101.1**	103.3	380.8
75	26.3	**148.9**	102.7	**102.6**	**380.5**
80	26.0	149.1	102.8	103.1	381.0
90	25.1	151.6	102.4	103.4	382.5
100	25.4	153.0	101.6	103.6	383.6
200	26.2	154.4	105.1	103.8	389.4

テーブル 2：総サーバータイム (秒) vs Mobile Link データベースワーカースレッドの固定数

このテストのベストタイムは 50 と 100 のデータベースワーカースレッド間を使用して得たものです。実施したテストでは、データベースワーカースレッド数が 75 のものが 4 つのタイプのシンクロナイゼーションの合計で最短タイムを記録しました。そのため、以降のテストでは、データベースワーカースレッド数は 75 で実施しました。データベースワーカースレッド数が 50 より上の場合、カーブは比較的フラットでした。タイミングにはばらつきがあることに注意してください。同一条件でも数秒の違いはよくあるため、テストは繰り返し実行しました。例えば、70 から 100 データベースワーカースレッドで実行したテストを 5 回繰り返し、総タイムでベストを記録しました。

データベースワーカースレッド数の最適な値は、シンクロナイゼーションスクリプトの同時（concurrent）実行による負荷がかかった統合データベースの処理能力の影響を最も受けます。これらのテストでは、どちらのサーバーコンピューターも 24 の論理 CPU を持ち、SAP SQL Anywhere のマルチプログラミングレベルはその値で固定しましたが、ベストなスループットは、Mobile Link データベースワーカースレッドの 2～4 倍で得られました。

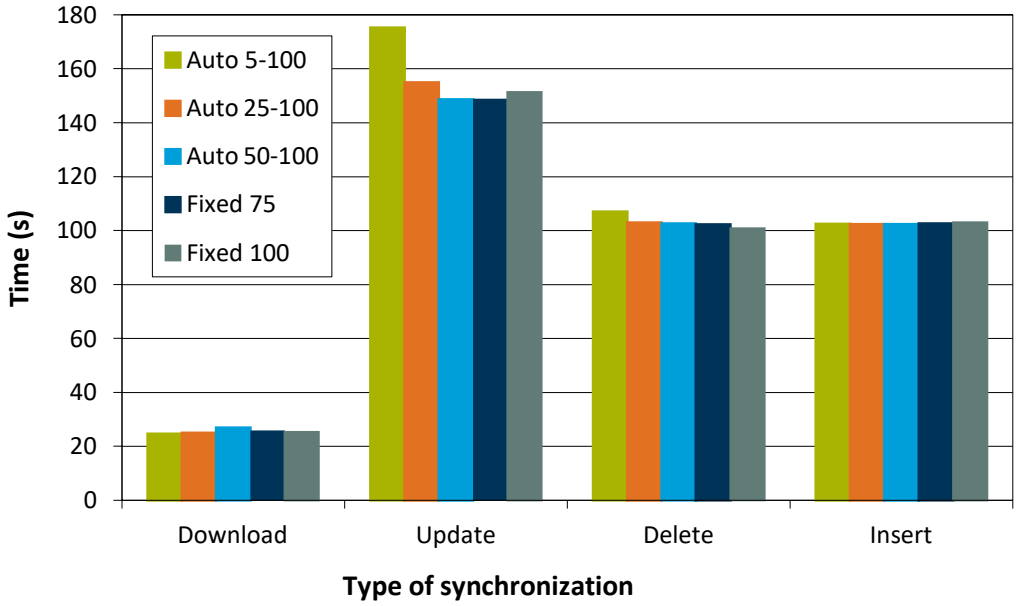
データベースワーカースレッドの自動調整機能を使用

Version 12 から実装された Mobile Link サーバーの新機能に、データベースワーカースレッド数を自動調整するオプションがあります。最小数 (–w オプション経由) よりも大きい最大数（–wm オプション経由）を指定した場合、Mobile Link サーバーは、パフォーマンスをモニターし、定期的にデータベースワーカースレッド数を調節します。これは、継続する負荷が存在する場合、あるいは少なくとも負荷の継続タイムが、調整ポイント間で 30 秒よりも長く続く場合最もうまく機能します。しかしながら、ここでのベンチマークテストは、調整ポイント間に負荷がない場合の高負荷のバーストに焦点をあてているため、負荷の継続タイムは通常 30 秒よりもかなり少なく、これらのテストに自動調整機能は理想的なものとは言えません。

データベースワーカースレッドの自動調整機能をテストし、固定数の場合と比較するため、以下の Mobile Link サーバーオプションを指定し、3 種の自動調整範囲と 2 つのワーカースレッド数固定値を使用しました。

- -wm 100 : デフォルトの 5 と 100 の間で自動的に調整
- -w 25 –wm 100 : 25 と 100 の間で自動的に調整
- -w 50 –wm 100 : 50 と 100 の間で自動的に調整
- -w 75 : 固定値 75 を使用（固定数を使用した実行からの最適値）
- -w 100 : 固定値 100 を使用（自動調整に選択された最大値と同じ）

上記でそれぞれ 5 回繰り返したベスト結果を表したのが下のグラフと表です。

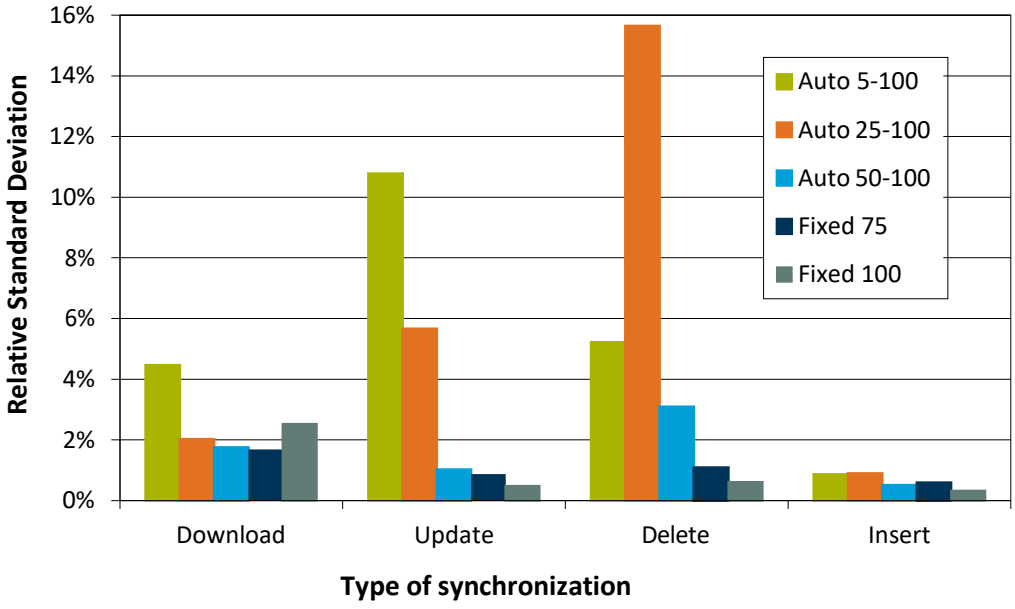


グラフ 3：異なるデータベースワーカースレッドで5回実行したうちのベスト総サーバータイム

DB ワーカーオプション	ダウンロード	更新	削除	挿入
自動 5-100	**24.6**	175.3	107.0	102.5
自動 25-100	25.0	154.9	103.0	**102.4**
自動 50-100	26.9	**148.7**	102.7	102.4
固定 75	25.8	148.9	102.3	102.6
固定 100	25.4	151.4	**100.8**	103.1

表 3：異なるデータベースワーカースレッドオプション毎5回実行したうちのベスト総サーバータイム

これらのテストにおけるパルス負荷は自動調整にとって理想的なものではないものの、自動調整をおこなった場合のベストは、（上の）最適に近い固定数または自動調整範囲の最大のものと同程度、またはそれより多少良い程度でした。しかしながら、理想的なテスト間のばらつきは、通常、下のグラフや表で示される相対標準偏差のように、より大きなものになります。



グラフ 4：異なるデータベースワーカースレッドオプションで5回実行した場合の相対標準偏差

DB ワーカーオプション	ダウンロード	更新	削除	挿入
自動 5-100	4.5%	10.8%	5.2%	0.9%
自動 25-100	2.0%	5.7%	15.6%	0.9%
自動 50-100	1.7%	1.0%	3.1%	0.5%
固定 75	**1.7%**	0.9%	1.1%	0.6%
固定 100	2.5%	**0.5%**	**0.6%**	**0.3%**

表 4：異なるデータベースワーカースレッドオプションで5回実行した場合の相対標準偏差

データベースワーカースレッド数が固定の場合が最もばらつきが少なく、自動調整を行った場合には、指定した範囲が自動変化なしのテストから判断した最適範囲に近い場合、ばらつきは、通常低めになります。

データベースワーカースレッドを自動調節する Mobile Link の新機能は、これらの難しいテストにおいてもうまく機能しました。とはいえ、自動調整における最適範囲を判断することには価値があります。統合データベースがシンクロナイゼーションを行いながらより多くのことを実行しなければならない現実のデータベーススキーマでは、データベースワーカースレッドが多すぎることで発生する競合のマイナスの影響がより顕著になることに注意してください。

このベンチマークのセットアップでも、1,000 クライアント同時（simultaneously）シンクロナイゼーションの試行に 80 以上のデータベースワーカースレッドを使用するメリットはありません。データベースワーカースレッドが多くなると、結果としてパフォーマンスの低下を招きます。データベースワーカースレッドを増加した場合のパフォーマンス低下の可能性の高い原因として下の 2 つがあります。

統合データベースにおける競合 - 他の接続によって接続がブロックされる機会は、データベースワーカースレッド数とともに増加し、ブロックされた接続は、パフォーマンスを低下させます。このテストでは、データベースの競合がパフォーマンスを制限しないよう、それぞれのクライアントは異なるデータにアクセスさせることにしましたが、通常は、ほとんどのアプリケーションでこれは不可能なため、統合データベースの競合がパフォーマンスのボトルネックになります。

サーバープロセッサーまたはディスクリソースのサチュレーション - それ以上利用可能な処理パワーがない場合、データベースワーカースレッドを追加すると、マルチタスクのための OS のオーバーヘッドが増加し、ハードウェアの競合が増加します。ベストなパフォーマンスが発揮される条件のテストでは、Mobile Link サーバーの CPU はダウンロードシンクロナイゼーションとアップロードされた更新シンクロナイゼーションの最初のところでサチュレーションしており、一方、統合データベースサーバーの CPU は更新シンクロナイゼーションのリマインダーや、削除・挿入のシンクロナイゼーションでサチュレーションしています。

データベースワーカースレッド数 75 では、統合データベースを一度に使用できたのは、同時シンクロナイゼーション 1,000 のうち 7.5% のみでした。残りのクライアントは、更新の送信中、データベースワーカースレッドのキューイング中、ダウンロード受信中、あるいはシンクロナイゼーション完了済みかのいずれかでした。クライアントの観点では、待機タイムの増加により、クライアントがキューの後ろに近いとシンクロナイゼーションタイムはより長くなります。これは、最悪のシナリオである、全てのシンクロナイゼーションをほとんど同時に開始した結果です。全シンクロナイゼーションリクエストが受けつけられるケースでは、スループットが最大になると平均クライアントタイムは最短になります（例：総タイムが最短化）。これによりどのキューも最速で移動します。

異なるシンクロナイゼーションのタイプによる相対タイムに気がつかれたでしょうか。ダウンロードは通常アップロードよりも高速です。アップロードでは、挿入が最速で、更新は通常最も低速です。更新をアップロードする場合、更新されたローのそれぞれ 2 つのバージョンがアップロードされるため（競合検出を可能にする新しいものと古いものの両方）、より多くのメモリーとネットワーク帯域幅が使用されます。

使用するデータベースワーカースレッド数を選択する際に、以下の点を念頭に入れてください。

- 現実の条件で予想されるシンクロナイゼーションのタイプを反映するテストを実行し、ベストなスループットを実現する自動調整の範囲を選択します。
- ベストなスループットには、十分な数のデータベースワーカースレッドを必要とします。しかし、多すぎてもスループットを低下させます。サーバー CPU またはディスクがすでにサチュレーション状態の場合、あるいは統合データベースの競合がボトルネックの場合、データベースワーカースレッド数を追加しても、逆効果です。
- 我々のテストでは、サーバーの CPU をサチュレーションさせるに十分なシンクロナイゼーション負荷では、およそ 75 のデータベーススレッドを使用することで、ベストなスループットを出すことができました。これは、Mobile Link サーバーと統合データベースサーバーの CPU コアの約 3 倍です。

テスト 2：最適なクライアントのリトライ間隔

最初のセクションで説明したように、Mobile link サーバーには Mobile Link クライアントのネットワーク接続、そしてそれによる同時（simultaneous）シンクロナイゼーション数に制限があります。この制限に達すると、追加のシンクロナイゼーションリクエストはネットワーク接続が拒否されたことを示すネットワークエラーとして返されます。OS のネットワークサポートのようなネットワークインフラの制限でもまた、ネットワーク接続が拒否されることが起こりえます。

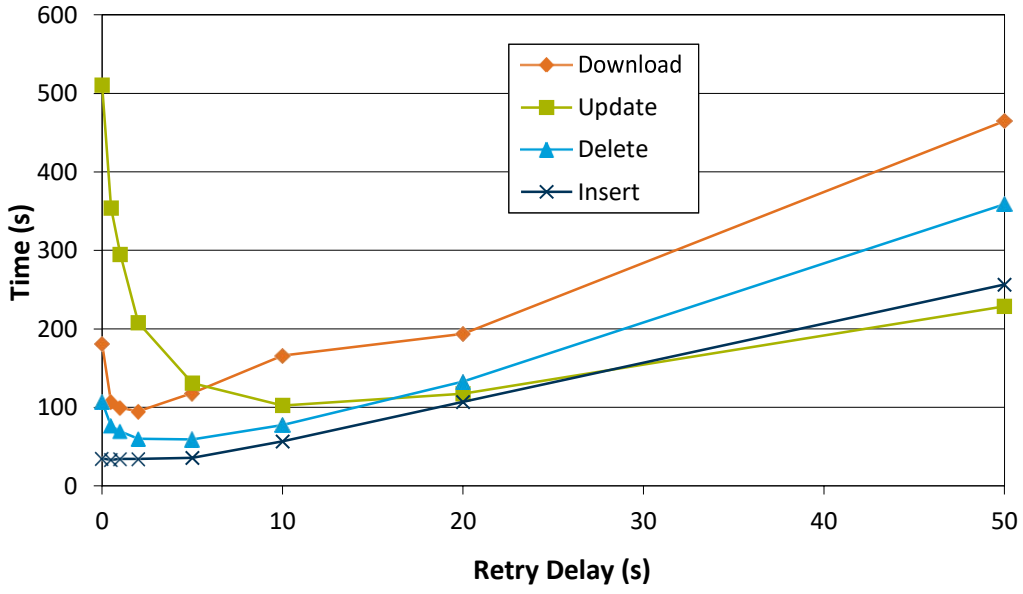
Mobile Link サーバー制限またはネットワーク制限によってシンクロナイゼーション接続リクエストが拒否された場合、クライアントアプリケーションまたはユーザーは、成功するまでシンクロナイゼーションを定期的にリトライする必要があります。Mobile Link クライアントはシンクロナイゼーションを自動的にリトライするわけではありません。シンクロナイゼーションリクエストの拒否とリトライの間隔は、シンクロナイゼーションスループットに影響を与える可能性があります。リトライ前の間隔が長すぎる場合、Mobile Link サーバーはアイドル状態になる可能性があります。間隔が短すぎる場合、Mobile Link サーバーはビジーに保たれ、アクティブなシンクロナイゼーションを処理せず、シンクロナイゼーションを拒否します。

このテストの目的は最適なリトライ間隔を決定することです。最初は、デフォルトの Mobile Link サーバー接続制限よりも多くの同時（simultaneous）シンクロナイゼーションがある場合、次に Mobile Link サーバー制限はシンクロナイゼーション数に設定し、それゆえ OS のみが接続を拒否する場合についてテストしました。このテストでは、クライアントアプリケーションによって使用されているリトライ間隔を変更しながら、10,000 のクライアントがそれぞれ 1,000 のローのシンクロナイゼーションを試みました。常に以下の条件で行いました。

- Pentium 4 3.2GHz プロセッサーのコンピューター 10 台、10,000クライアント
- 1 クライアントシンクロナイゼーションあたりのロー数 1,000（ローごとに 92 バイト）
- 総シンクロナイゼーション数 10,000、1,000 万ロー
- Mobile Link データベースワーカースレッド数75

Mobile Link サーバー接続制限の超過

デフォルトの Mobile Link 接続制限の 1,024 を使用した場合のタイミング結果を以下のグラフと表に示します。



グラフ 5：総サーバータイム（秒）vs デフォルト接続制限 1,024 の 10,000 クライアントのリトライ間隔

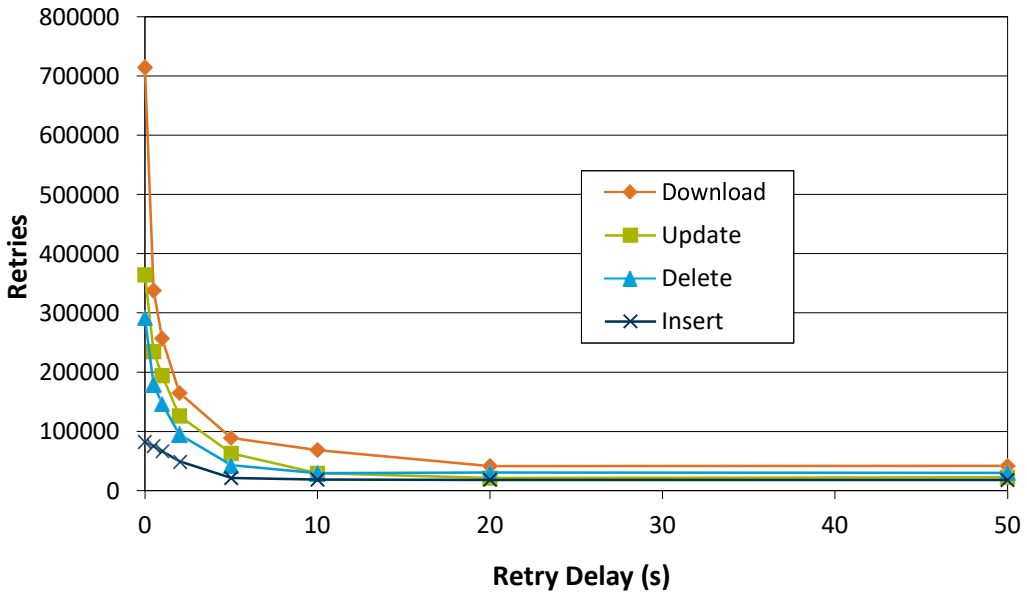
リトライ間隔	ダウンロード	更新	削除	挿入	計
0	180.7	510.7	107.0	34.7	833.0
0.5	106.7	354.2	76.4	**33.3**	570.7
1	99.7	295.0	69.6	34.3	498.6
2	**95.1**	208.0	60.3	34.3	397.6
5	118.2	131.1	**59.4**	35.8	**344.6**
10	165.9	**102.7**	77.7	56.7	403.0
20	194.0	117.6	132.9	107.1	551.6
50	465.0	229.0	359.0	256.5	1309.4

表 5：総サーバー時間（秒）vs デフォルト接続制限 1,024 の 10,000 クライアントのリトライ間隔

リトライ間隔はスループットに影響します。最適なリトライ間隔はシンクロナイゼーションのタイプに依存します。例えば、挿入のアップロードのベストなタイムはリトライの間隔が 0.5 秒の時です。しかしながら、更新のアップロードでは、ベストなリトライ間隔は 10 秒です。 通常、ベストの間隔は異なるタイプのシンクロナイゼーションのスピードに対応します。より高速なシンクロナイゼーションのタイプはより短いリトライ間隔でベストなスループットになります。間隔が短すぎる場合、Mobile Link サーバーは接続リクエストを拒否するのに多くのタイムをかけすぎることになります。間隔が長すぎる場合、Mobile Link サーバーには使用されていない容量が存在することになります。

実施したテストでは、リトライ間隔が 5 秒のものが 4 種のシンクロナイゼーションの合計タイムが最短だったので、デフォルトの接続制限を使用した次のテストではその値を使用しました。

（デフォルトの Mobile Link 接続制限を使用した）同じ条件下で実行した結果拒否されたシンクロナイゼーションリクエスト数を下のグラフと表に示します。



グラフ 6：クライアントのリトライ数 vs デフォルト接続制限 1,024 で行った 10,000 クライアントリトライ間隔

リトライ間隔	ダウンロード	更新	削除	挿入
0	714,616	364,457	291,695	81,744
0.5	338,340	234,931	179,385	75,210
1	257,282	194,344	146,663	66,810
2	165,317	126,472	95,042	49,594
5	88,830	63,106	42,950	21,434
10	68,530	29,433	**29,558**	18,663
20	**41,289**	**21,418**	30,475	18,522
50	42,236	22,664	30,134	**18,007**

表 6：クライアントのリトライ数 vs デフォルト接続制限 1,024 で行った 10,000 クライアントリトライ間隔

Mobile Link 接続制限を増加

Mobile Link 接続制限を、同時（simultaneous）シンクロナイゼーション数に上げた後、リトライ間隔も変更されました。この設定では、Mobile Link はどの接続も拒否しませんでした。しかし、OS が接続を拒否するためにクライアントがリトライを必要とする可能性はあります。

Mobile Link 接続制限の 10,0000 を使用したタイミング結果を以下のグラフと表に示します。

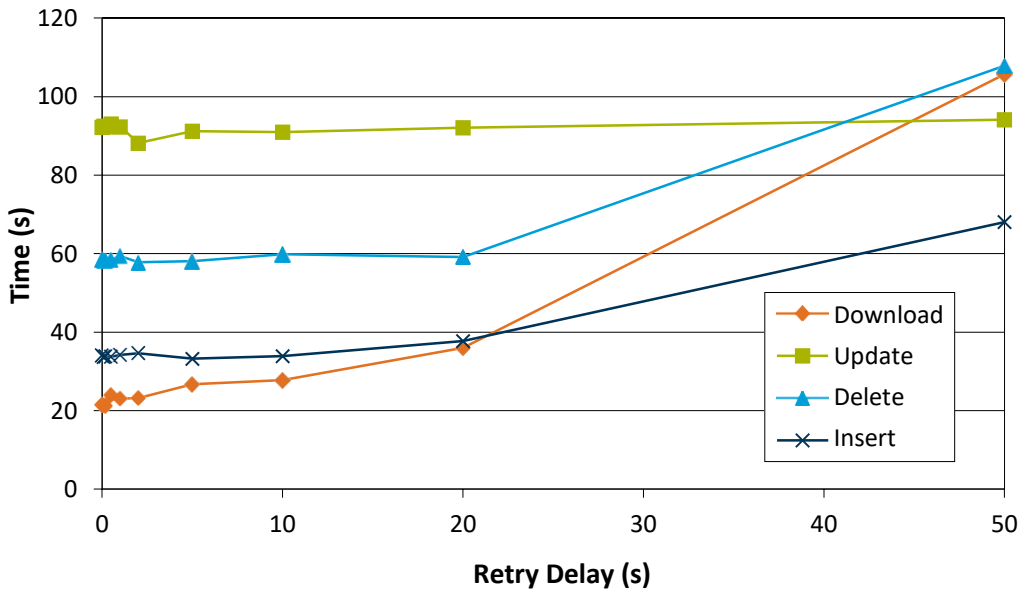


表 7：総サーバータイム vs 接続制限 10,000、10,000 クライアントのリトライ間隔

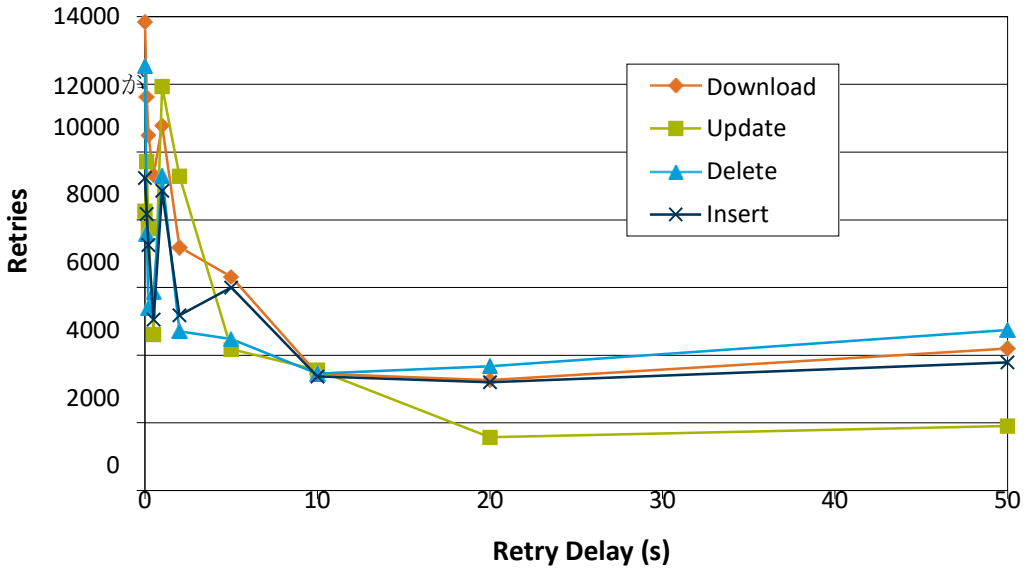
リトライ間隔	ダウンロード	更新	削除	挿入	計
0	21.5	92.2	58.5	34.0	206.2
0.1	21.4	92.5	58.4	33.7	205.9
0.2	**21.3**	92.3	58.3	33.8	205.7
0.5	23.9	93.0	58.5	33.7	209.1
1	23.0	92.3	59.4	34.2	208.9
2	23.2	**88.1**	**57.8**	34.6	**203.7**
5	26.7	91.2	58.0	**33.2**	209.2
10	27.7	91.0	59.8	33.9	212.4
20	36.0	92.1	59.1	37.7	224.9
50	105.7	94.1	107.9	68.1	375.8

表 7: 総サーバー時間（秒） vs 接続制限 10,000、10,000 クライアントのリトライ間隔

テストでは、リトライ間隔が 2 秒のものが、4 種のシンクロナイゼーションのタイムの合計が最短だったため、次の接続制限 10,000 のテストでは、この値を使用しました。

デフォルトの接続制限での結果と比較すると、タイミングはリトライ間隔にあまり依存していません。特に、Mobile Link サーバーが接続を拒否することがないと、2 秒未満のリトライ間隔を使用することによるペナルティはほとんどありません。タイミングは、特にダウンロードにおいてより良く、（下に示されるように）リトライ数はより少なくなります。

（Mobile Link 接続制限の 10,000 ）と同じ条件で実行し、拒否されたシンクロナイゼーションリクエスト数を下のグラフと表に示します。



グラフ 8：クライアントリトライ数 vs 接続制限 10,000、10,000 クライアントでのリトライ間隔

リトライ間隔	ダウンロード	更新	削除	挿入
0	13,842	8,260	12,543	9,238
0.1	11,637	9,719	7,597	8,173
0.2	10,500	7,744	5,403	7,256
0.5	9,322	4,614	5,857	5,052
1	10,787	11,940	9,306	8,858
2	7,190	9,290	4,709	5,180
5	6,315	4,173	4,481	5,999
10	3,451	3,564	**3,457**	3,373
20	**3,262**	**1,575**	3,673	**3,202**
50	4,200	1,907	4,744	3,789

表 8：リトライ数 vs 接続制限 10,000、10,000 クライアントでのリトライ間隔

Mobile Link 接続制限の影響は、以下のテストでより詳細に解説します。

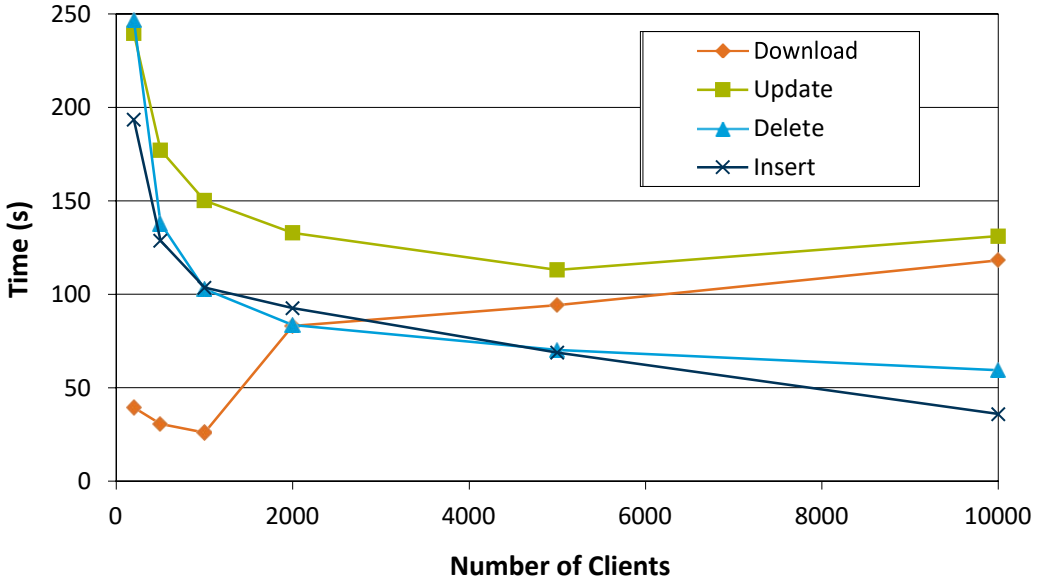
テスト 3：クライアント数に関するスケーラビリティー

このテストでは、データベースワーカースレッド数をテスト 1 の結果から得たベストな値に保ち、クライアントの数を変更することで、Mobile Link がクライアント数の増加に対してどのようにスケールするかをみました。クライアント毎のシンクロナイゼーション数は、全体として同量のデータが送信されるように、そして、各シンクロナイゼーションサイズも一定に保つよう調整しました。例えば、クライアント数 10,000では、（それぞれのタイプで）1 つのシンクロナイゼーションを使用しました。500 クライアントでは、クライアント毎に20シンクロナイゼーションを使用しました。さらに、200 クライアントでは、クライアント毎に 50 シンクロナイゼーションを使用しました。また、以下の条件を保ちました。

- Pentium 4 3.2 GHz プロセッサーのコンピューター 10 台上に 10,000 クライアント
- クライアントシンクロナイゼーションあたりのロー数 1,000（ローごとに 92 バイト）
- 総シンクロナイゼーション数 10,000、1,000 万ロー
- Mobile Link データベースワーカースレッド数 75

1,000クライアントより多い場合のテストは2回行いました。1 回目はデフォルトのネットワーク接続 1,024 で 5 秒後にリトライし、2 回目はネットワーク接続制限を10,000 に上げてリトライは 2 秒後に行いました。

デフォルトの接続制限（およびリトライ間隔 5 秒）の結果を、下のグラフと表に示します。

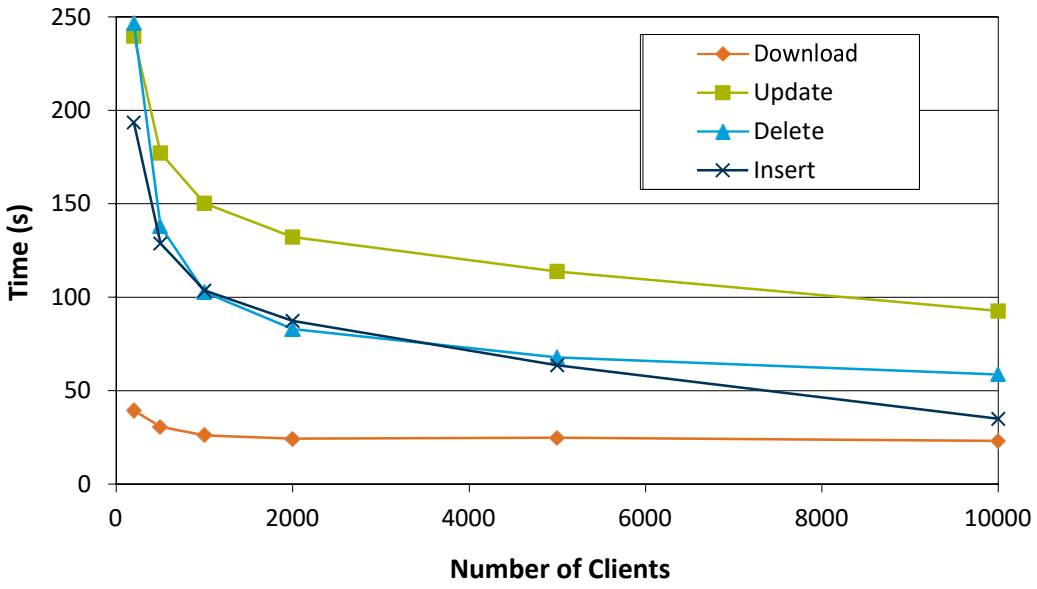


グラフ 9：総サーバータイム vs 各ポイントで 10,000 シンクロナイゼーションするクライアント数。接続制限はデフォルトの 1024、リトライ間隔は 5 秒

クライアント数	ダウンロード	更新	削除	挿入
200	39.3	239.7	246.7	193.5
500	30.6	177.1	137.9	128.7
1,000	**26.0**	150.2	102.7	103.5
2,000	83.0	132.9	83.5	92.6
5,000	94.2	**113.1**	70.2	68.8
10,000	118.2	131.1	**59.4**	**35.8**

グラフ 9：総サーバータイム vs 各ポイントで10,000シンクロナイゼーションするクライアント数。接続制限はデフォルトの1024、リトライ間隔は5秒

接続制限を上げた結果を下のグラフと表に示します。このグラフは上の表の 200, 500、1,000 クライアントと同じデータを含みます。なぜならば、これらの実行結果は Mobile Link 接続制限を超えず、リトライはなかったからです。



グラフ10：総サーバータイム vs 各ポイントで 10,000 シンクロナイゼーションするクライアント数。接続制限は 10,000、リトライ間隔は 2 秒

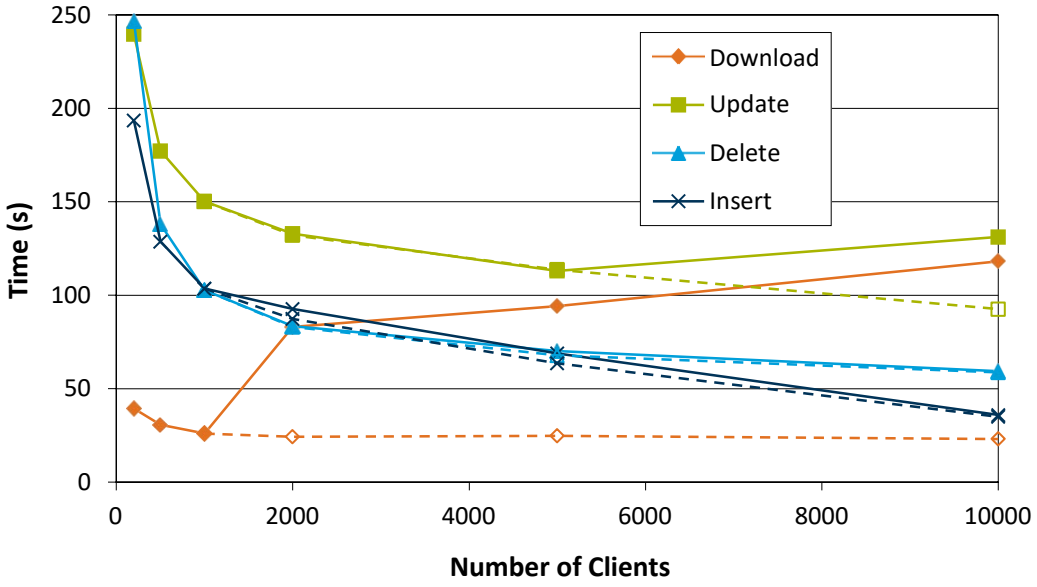
クライアント数	ダウンロード	更新	削除	挿入
2,000	24.2	132.2	82.9	87.3
5,000	24.7	113.7	67.8	63.5
10,000	**23.1**	**92.6**	**58.7**	**34.9**

表10：総サーバータイム(秒) vs クライアント数。接続制限は 10,000、リトライ間隔は 2 秒

グラフが示すように、Mobile Link サーバーは、全てのシンクロナイゼーションのタイプにおいて、より多くの接続制限数、ダウンロードと更新を除いた全タイプのデフォルト制限で、シンクロナイゼーションするクライアント数の増加とともに、理想的にスケールします。Mobile Link サーバーが接続を拒否している場合、Mobile Link 接続数を超えるとダウンロードのスループットは劇的に落ちます

観察されたスケーラビリティーから、少なめの数のクライアントで実施したテスト結果を、より大規模なクライアント数の結果予測に使用することが可能だということが示唆されました。

接続制限の影響を説明するため、以前の 2 つのグラフを合体させたグラフを下に示します。デフォルトの接続制限での結果には実線を、接続制限を上げた場合の結果は点線で表しています。実践と点線の偏差は、Mobile Link 接続制限を超えたことが原因です。



グラフ 11：総サーバータイム (秒) vs 異なる接続制限でのクライアント数。実線はデフォルトの接続制限数の 1,024、点線は 10,000

Mobile Link 接続制限を超える場合のコストは、Mobile Link 11 のホワイトペーパーで行ったベンチマークで見られたものよりも大きくなりました。おそらく、Windows OS の変更によるものと考えれます。以前のベンチマークで使用したのは、Windows Server 2003 で、ネットワーク接続を受けつけるための大きな動的バックログを有効にするためレジストリ設定を変更していました。OS は、Mobile Link 接続制限が超過した場合でも、より多くの接続リクエストをバッファーするため、リトライがより少ない結果になります。Windows Server 2000 と2003 では、動的バックログの機能は、"TCP/IP SYN flooding" アタックに対してオプショナルなプロテクションを提供していました。Windows Server 2008 では代替のプロテクションが提供されていますが、動的バックログの機能は、利用できません。そのため、これらのテストでは、より多くのリトライがあります。

Mobile Link をかなりの規模の数のクライアントに使用することを検討する場合、以下のポイントに注意してください。

- 総サーバータイムを最小限にするため、現実世界の使用を正確にシミュレートしたテストを行うことが重要です
- Mobile Link 接続制限を超過すると、特に OS 側に接続リクエストのための大きなバックログを使用する設定ができない場合など、スループットに悪影響を与える可能性があります

テスト 4: 各シンクロナイゼーションのサイズ

このテストでは、シンクロナイゼーションサイズによるパフォーマンスの影響を見るため各シンクロナイゼーションで送信されるロー数を変更しました。クライアント数は 1,000 を使用し、シンクロナイゼーションされる総ロー数を変更せずに、シンクロナイゼーション毎のロー数を広く変更できるようにしました。

このテストでは、クライアント毎のシンクロナイゼーション数を変更することで、送信される総データ量を変更せずにシンクロナイゼーション毎のロー数を変更しました。例えば、各クライアントは 10,000 ローを一度、5,000 ローを 2 回、2,000 ローを 5 回、等シンクロナイゼーションできました。以下の条件を、常に保ちました。

- 1 Pentium 4 3.2 GHz プロセッサーのコンピューター 10 台上で 10,000 クライアント
- 総シンクロナイゼーション数 10,000、1,000 万ロー
- Mobile Link データベースワーカースレッド数 75

結果を以下のグラフと表に示します。

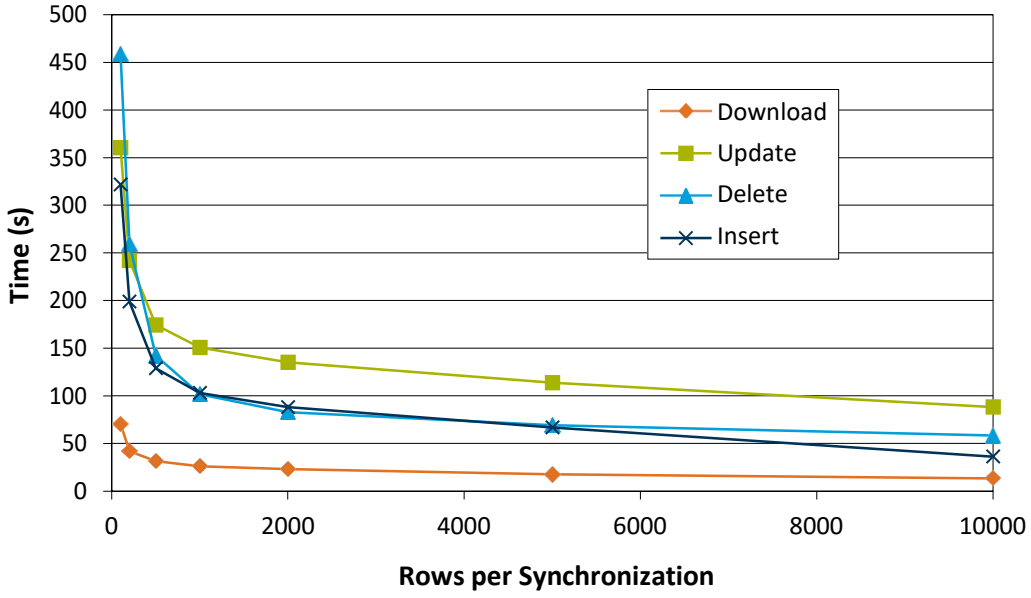


表12：総サーバータイム vs シンクロナイゼーション毎のロー数。各ポイントで1,000万ローシンクロナイゼーションを表す

ロー数	ダウンロード	更新	削除	挿入
100	70.7	360.5	458.7	321.8
200	41.9	242.2	259.5	198.9
500	31.4	174.5	142.4	128.9
1,000	26.1	150.8	101.9	103.1
2,000	23.0	135.3	82.8	88.2
5,000	17.7	113.9	69.2	66.9
10,000	13.6	88.4	58.3	36.4

表 11：総サーバータイム（秒）vs シンクロナイゼーション毎のロー数

シンクロナイゼーションのサイズはスループットに影響します。小規模なシンクロナイゼーションの数が多い時が最も低速で、大規模なシンクロナイゼーションが少数である方が総サーバータイムは高速になりました。これは、各シンクロナイゼーションのオーバーヘッドが固定の場合でも一貫していましたが、これはシンクロナイゼーションのサイズがより大きい場合に低減しました。

同時 (simultaneous) シンクロナイゼーション数 1,000 のセットごとの平均総サーバータイムを得るには、上記からの総タイムを各クライアントが行うシンクロナイゼーション数で割ると、より明確になります。以下のグラフは、タイム vs シンクロナイゼーションしたロー数を示します。

80

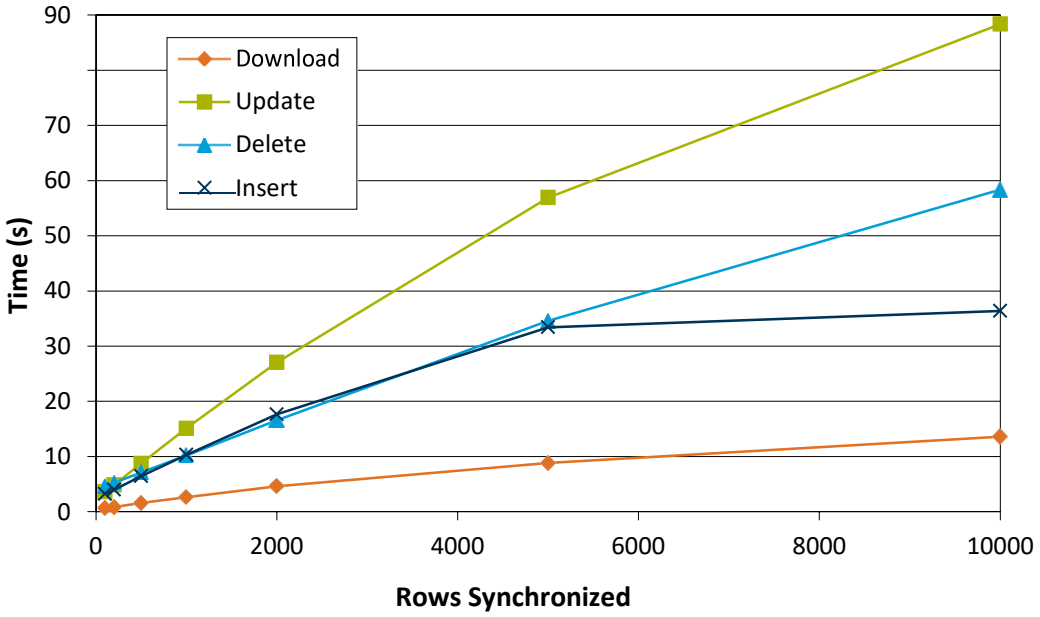


表 13：1,000シンクロナイゼーション（秒）のセットにおける総サーバータイムの平均 vs シンクロナイゼーションロー数
このグラフでは、シンクロナイゼーションされるローの総数は、各ポイントのそれぞれと同じではない

ご覧のとおり、1,000 シンクロナイゼーションのセットの平均タイムとシンクロナイゼーションされたロー数の間にはほぼリネアな関係があります。シンクロナイゼーションされたローがない場合（それゆえアップロードスクリプトが関係しない場合）、全てのラインは1,000 の空のシンクロナイゼーションのタイムに集約・集中コンバージェします。

- シンクロナイゼーションのサイズを検討する際には、シンクロナイゼーションごとのオーバーヘッドがあることを念頭に置いてください。そのため、スループットのレートは、小規模シンクロナイゼーションではより遅くなります

テスト 5：ネットワークプロトコル

Mobile Link クライアントと Mobile Link サーバー間の接続に使用可能なネットワークプロトコルにはいくつかのオプションがあります。テストでは、暗号化や圧縮なしの TCP/IP 接続を使用しました。TCP/IP または HTTP のどちらかを選択して使用することができます。暗号化では、TLS または HTTPS になります。さらに暗号化には 2 つのオプションがあります。RSA と FIPS で承認された RSA（FIPSとします）です。FIPS は別オプションになっていることに注意してください（Editionによっては含まれているものもあります）。Mobile Link の以前のバージョンでは、ECC も利用可能でした。これらのオプションでは、Mobile Link クライアントとサーバー間のデータ送信サイズを削減するために zlib 圧縮を有効にすることができます。これらのオプションのいずれでも、zlib 圧縮を有効にし、Mobile Link クライアントとサーバー間のデータサイズを低減することが可能です。

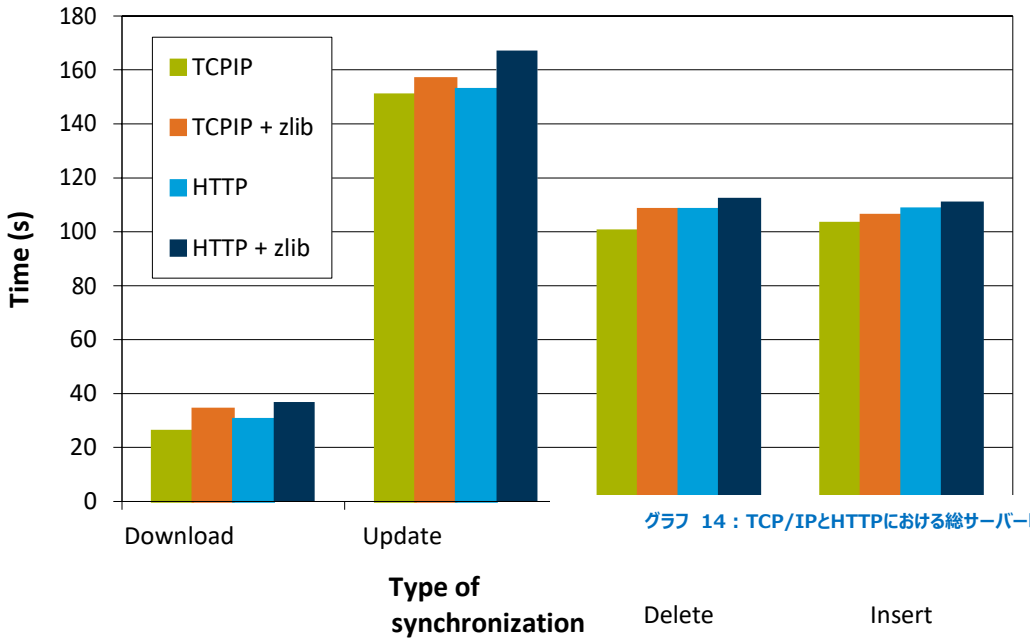
このテストでは、異なるネットワークプロトコルのオプションを Mobile Link クライアントとサーバー間で使用し、パフォーマンスへの影響をみました。Mobile Link のデフォルトのネットワーク接続制限の1,024 より下になるよう 1,000 クライアントを使用しました。以下の条件を常に保ちました。

- Pentium 4 3.2GHzプロセッサーのコンピューター 10 台上の 10,000 クライアント
- 1 クライアントシンクロナイゼーションあたりのロー数 1,000（ローごとに 92 バイト）
- 各クライアントのシンクロナイゼーションのセット数 10
- 総シンクロナイゼーション数 10,000、1,000 万ロー
- Mobile Link データベースワーカースレッド数 75

他のテストのように、クライアントと Mobile Link サーバー間の接続には、専用のギガビットネットワークを使用しました。

TCP/IP または HTTP

暗号化なしの実行結果を以下のグラフとテーブルに示します。



グラフ 14：TCP/IPとHTTPにおける総サーバー時間、zlib 圧縮ありとなし

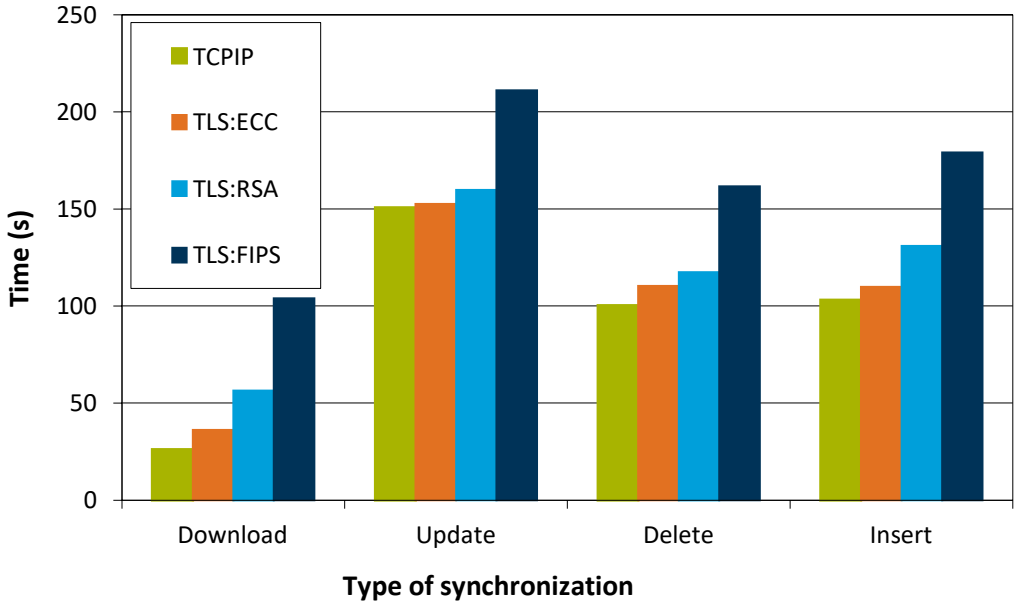
プロトコル	ダウンロード	更新	削除	挿入
TCPIP	**26.1**	**150.8**	**100.4**	**103.2**
TCPIP + zlib	34.3	156.9	108.3	106.2
HTTP	30.5	152.9	108.3	108.6
HTTP + zlib	36.9	167.2	112.1	110.7

表 12：TCP/IPとHTTPにおける総サーバータイム、zlib 圧縮ありとなし

これらのテストでは、TCP/IP のタイムがベストでした。また、HTTP では、17% 長かったダウンロードタイムをのぞけば、10% 内でした。圧縮は TCP/IP も HTTP もタイムが増加しましたが、アップロードでは 10% 内、ダウンロードシンクロナイゼーションでは長くなりました。TCP/IPでは 31%、HTTP では 21% でした。低速のネットワークでは、圧縮はスループットを改善することができました。しかし、暗号化なしのテストでは役にたっているようにはみえませんでした。

暗号化された TCP/IP (TLS)

圧縮なし、暗号化なしの TCP/IP と、暗号化された TCP/IP (TLS) の実行結果を以下のグラフと表に示します。



グラフ 15：TCP/IP での総サーバータイム、異なる暗号化オプション、圧縮なし

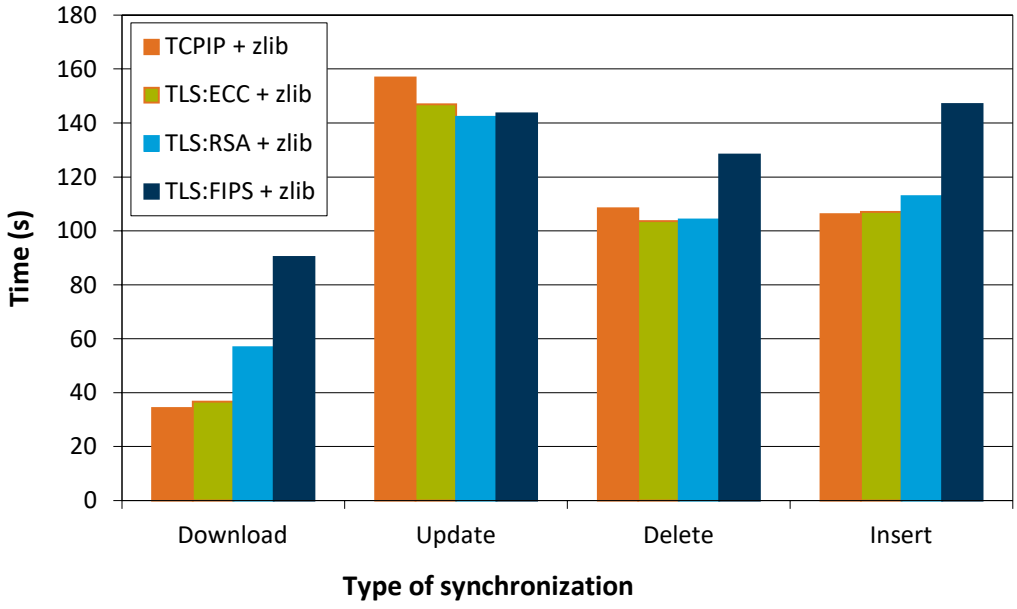
プロトコル	ダウンロード	更新	削除	挿入
TCPIP	**26.1**	**150.8**	**100.4**	**103.2**
TLS:ECC	36.2	152.6	110.4	109.9
TLS:RSA	56.4	159.9	117.4	131.0
TLS:FIPS	103.9	211.1	161.7	179.1

表 13：TCP/IP での総サーバータイム、異なる暗号化オプション、圧縮なし

暗号化オプションの間の大きな違いはダウンロードタイムに現れます。全てのシンクロナイゼーションの種類で、相対的な順序は以下のとおりです。

1. 暗号化なし
2. ECC 暗号化
3. RSA 暗号化
4. FIPS RSA 暗号化

TCP/IP で暗号化と圧縮のどちらも使用した結果を以下のグラフと表に示します。



グラフ 16：TPC/C での異なる暗号化オプション、zlib 圧縮毎の総サーバータイム

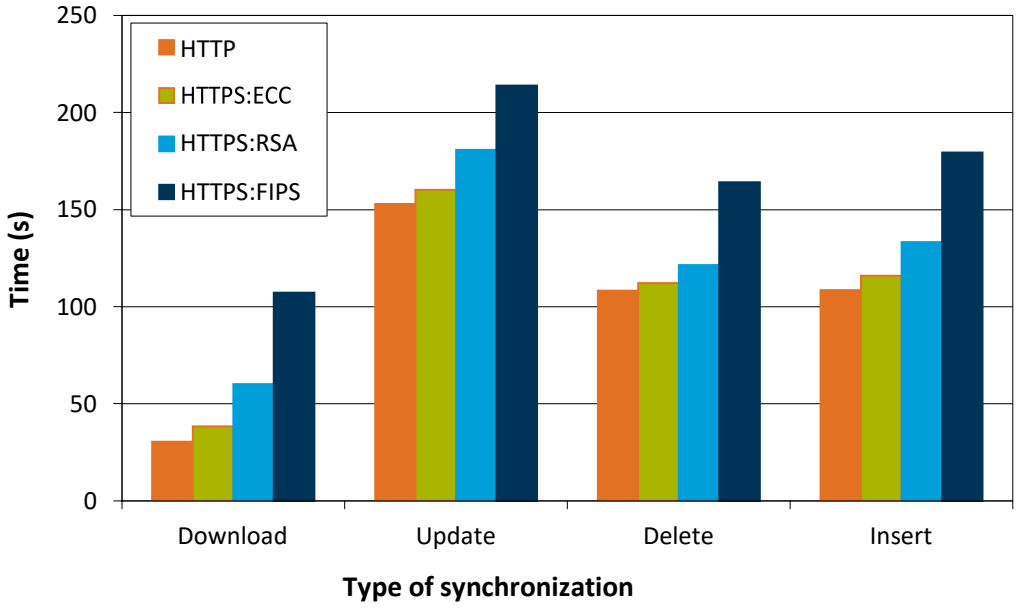
プロトコル	ダウンロード	更新	削除	挿入
TCPIP + zlib	**34.3**	156.9	108.3	**106.2**
TLS : ECC + zlib	36.7	146.9	**103.6**	107.0
TLS : RSA + zlib	56.8	**142.2**	104.2	112.8
TLS : FIPS + zlib	90.3	143.6	128.3	147.0

表 14 : TPC/C での異なる暗号化オプション、zlib 圧縮毎の総サーバータイム

暗号化した状態では、ECC と RSA を使用したダウンロード同様、圧縮した場合でもタイム増はありませんでした。あるいは FIPS 暗号化を使用した場合には特にタイムが大幅に短縮されました。更新と削除（FIPS 以外）では、暗号化の結果は、暗号化なしよりも高速でした。圧縮した場合には、暗号化の種類による違いはほとんどありませんでした。高速ネットワークの場合でも、暗号化を使用する場合には、常に圧縮も使用する方が価値が高いようです。

暗号化された HTTP (HTTPS)

圧縮なし、暗号化なしの HTTP と、暗号化ありの HTTPS の実行結果を以下のグラフとテーブルに示します。



グラフ 17： HTTP における異なる暗号化オプション、圧縮なし、の総サーバータイム

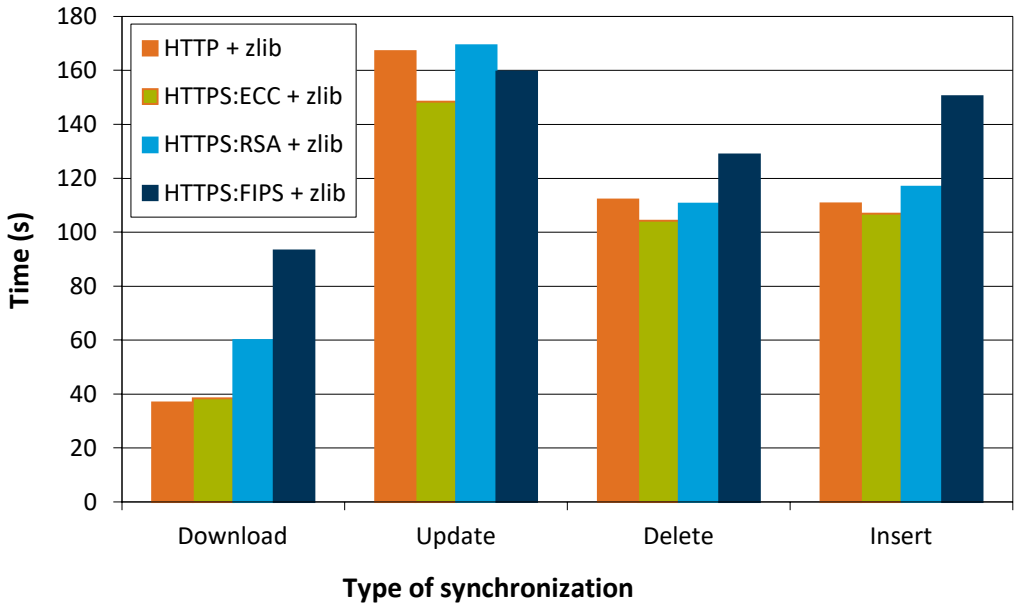
プロトコル	ダウンロード	アップデート	削除	挿入
HTTP	**30.5**	**152.9**	**108.3**	**108.6**
HTTPS : ECC	38.4	160.2	112.2	116.0
HTTPS : RSA	60.1	180.8	121.4	133.4
HTTPS : FIPS	107.3	213.9	164.1	179.4

表 15：HTTPにおける異なる暗号化オプション、圧縮なし、の総サーバータイム

TCP/IPで、暗号化オプション間の最大の違いは、ダウンロードタイムに現れました。全てのシンクロナイゼーションの種類において、相対順序は以下のとおりです。

1. 暗号化なし
2. ECC 暗号化
3. RSA 暗号化
4. FIPS RSA 暗号化

HTTP で暗号化と圧縮の双方を使用した結果を以下のグラフと表に示します。



グラフ 18：異なる暗号化オプションと zlib 圧縮を使用した HTTP における総サーバータイム

プロトコル	ダウンロード	更新	削除	挿入
HTTP + zlib	**36.9**	167.2	112.1	110.7
HTTPS : ECC + zlib	38.5	**148.4**	**104.2**	**106.9**
HTTPS : RSA + zlib	60.0	169.3	110.5	116.9
HTTPS : FIPS + zlib	93.2	159.6	128.8	150.4

グラフ 18：異なる暗号化オプションと zlib 圧縮を使用した HTTP における総サーバータイム

暗号化オプションでは、圧縮も、ECC と RSA を使用したダウンロード同様、どちらもタイムの追加はありませんでした。特に FIPS 暗号化では、TLS 同様、タイムを大幅に短縮しました。アップロードでは、ECC 暗号化での結果は、暗号化なしよりも高速でした。圧縮では、暗号化タイプ間の違いはほとんどありませんでした。TLS、HTTPSでは、同様に圧縮を使用した場合がベストでした。

ハードウェア要件

Mobile Link サーバーの相対的なハードウェア要件を評価するために、統合データベースにとって理想的なコンピューターで実行し、2 台のコンピューターにおける CPU の利用率を観察しました。CPU の利用率は、Microsoft の Process Explorer から得ました。

２台のサーバーコンピューターのうちの 1 台の、タイム記録の（timed） シンクロナイゼーション中の全体的な CPU 利用率は、通常 90% から 100% でした。例外は、クライアント数が少ない場合のダウンロードとデータベースワーカースレッド数が少ない場合です。CPU 利用率がどちらかのサーバーで 100% に達した場合、ボトルネックはそのサーバーコンピューターの処理スピードにあります。Mobile Link サーバーコンピューターの CPU 利用率が、ダウンロードのボトルネックでした。Mobile Link サーバーがアップロードされたデータを受信し、それを統合データベースに適用する前にそれを処理する際の、アップロードをシンクロナイゼーションするごく始めの、特に更新です。アップロードが統合データベースに適用されると、CPU の利用率は、90% から 100% になります。一方で、Mobile Link サーバーのCPU 利用率は、50% 未満で、通常 35% 前後です。

ダウンロードシンクロナイゼーションでは、Mobile Link サーバーの CPU 利用率は 100% に近く、統合データベースサーバーの CPU 利用率は通常 50% 未満です。これらのテストが、単純なスキーマとシンクロナイゼーションスクリプトを使用し、データベースを完全にインメモリーで稼働させることで統合データベースの負荷を最小限にするために設計されていることを考えると、Mobilink Server は、ダウンロードにだけボトルネックがあるように見えます。

十分な RAM がある限りは、Mobile Link サーバーはディスクアクセスを少ししか使用しません。そのため、ほとんどのデータベースサーバーとは異なり、高速なディスクアクセスや大規模ストレージ容量を必要としません。そのため、これらのテスト期間中、統合データベースはディスクバウンドがないよう完全にインメモリーで稼働させました。しかし、統合データベースは通常ディスクバウンドなため、異なるパフォーマンス特徴を表す結果になる可能性があります。

ネットワークがボトルネックではなかったことを示すために、Mobile Link クライアントと Mobile Link サーバー間のネットワークトラフィックは、Mobile Link サーバーと統合データベース間のトラフィックのものとは異なるネットワークと Mobile Link サーバーイーサネットポートを使用しました。これにより、全トラフィックが同じネットワークと Mobile Link サーバーイーサネットポートを使用した場合より良いパフォーマンスを得ることができました。

Mobile Link のハードウェア要件を検討する場合、以下のポイントを思い出してください。

- Mobile Link は、統合データベースと比較して処理パワーをほとんど必要とせず、ディスクの容量やパフォーマンスもほとんど必要としません。

推奨

このセクションでは、すでに述べた Mobile Link パフォーマンス Tips についてまとめ、大規模なディプロイメントにおいてどう Mobile Link が処理するか予測できるよう小規模なテストの実行方法をアドバイスします。また、Mobile Link に必要なハードウェアリソースについてもアドバイスします。

パフォーマンス向上のためのTips

以下の Tips に従ってください。Mobile Link のベストなパフォーマンスを引き出すのに役立つはずです。

- 高負荷下のシンクロナイゼーションスクリプトの同時（concurrent）パフォーマンスをテストおよびチューニングしてください。我々の経験上、シンクロナイゼーションスクリプトの競合が Mobile Link システムにおいて最もよく見られるパフォーマンス問題です。
- 高負荷下のシンクロナイゼーションスクリプトの同時（concurrent）パフォーマンスをテストおよびチューニングしてください。我々の経験上、シンクロナイゼーションスクリプトの競合が Mobile Link システムにおいて最もよく見られるパフォーマンス問題です。
- 最適なスループットを提供する Mobile Link データベースワーカースレッド数の自動調整範囲を使用してください。例えば、我々のテストではおよそ 75 のデータベースサーカースレッドまたは 50 から 100 の間の自動調整で最適なスループットを得ました。これらのテストでは、より大きな数でもパフォーマンスに大きな影響はありませんでした。しかしながら、現実世界のデータベーススキーマの競合は、最適な数よりもずっと大きくなりがちです。そのため、高めに固定した値よりも、自動調整の範囲を使用する方が良いことになります。
- スループットを最大化するネットワーク接続制限か、受けつける同時（simultaneous）シンクロナイゼーション数の最大数を強化するネットワーク接続制限を選択してください（Mobile Link -nc オプション経由で）。デフォルトを超える数にすることで、スループットを改善することは可能ですが、OS の制限によって接続を拒否される可能性があります。
- Mobile Link サーバーが少なくとも 64 bit OS のコンピューターで稼働しており、Mobile Link サーバーのメモリー要件やその他のメモリー要件を満たすよう十分な物理メモリーがあることを確認してください。
- Mobile Link キャッシュをディスクにスワッピングしないよう利用可能な十分なメモリーがあることを確認してください。以下のような警告がないか、Mobile Link サーバーメッセージログやコンソールをチェックしてください。

  ```
  MobiLink server has swapped memory pages out:9000 in:8000 concurrent
  pages:10000
  ```

  このようなメッセージを目にした場合、小さすぎるので、 –cm または –cmax オプションを指定されていないか（デフォルトは物理メモリーの 70%）、利用可能な十分な物理メモリーがあるかチェクしてください。
- ２つ以上のスクリプトバージョンを使用している場合には、Mobile Link がデータベース接続をクローズして作成する必要性を削減するため、（Mobile Link -cn オプション経由で）Mobile Link データベース接続数の最大数が、通常のシンクロナイゼーションバージョンx Mobile Link データベースワーカースレッド数になるよう設定することを検討してください。
- 業務要件とコンパチブルな最小ログ取得 verbosity を使用してください。デフォルトでは、Verbose ログ取得はオフになっており、Mobile Link はログをディスクに書き込みません。異なるログ取得オプションでテストし、そのログ取得 verbosity がシステムのパフォーマンスを低下させていないか確認してください。

大規模ディプロイメント

単一の Mobile Link サーバーで、数万、数十万ものクライアントデータベースを扱うことができます。これまでのテストでは、最大 10,000 クライアントの同時 (simultaneous) シンクロナイゼーションについてみてきました。シンクロナイゼーションのタイプにより、これは、21 から 88 秒かかりました。これは、1 時間に 40 万 8,000 から 1,600 万 クライアントデータベース、あるいは、1 日に 1,000 万から 4,100 万シンクロナイゼーションしていることに相当します。これに必要なハードウェアの要件は、非常に低いものです。Mobile Link は 1 インスタンスで非常に多数のクライアントデータベースを扱うことが可能で、通常、Mobile Link が接続する統合データベースで必要とされるものよりも少ないハードウェアリソースで十分です。

1 インスタンスの Mobile Link が占有するサーバーコンピューターを使用しても必要とするパフォーマンスに満たない、あるいは可用性要件に満たない場合、サーバーファーム内に、複数の Mobile Link を使用することが可能です。契約によっては別ライセンスが必要になる場合があります。 例えば、（SAP SQL Anywhere に含まれる）Relay Server と複数の Mobile Link サーバーを使用することもできます。あるいは、ハードウェアロードバランサーや、アプリケーションデリバリーコントローラーと複数の Mobile Link サーバーを組み合わせて使用することもできます。

Mobile Link サーバーが採用しているアーキテクチャーには、シンクロナイゼーションヒエラリキーもあります。シンクロナイゼーションヒエラルキーでは、クライアントは、定期的にプライマリー統合データベースとシンクロナイゼーションするセカンダリ統合データベースとシンクロナイゼーションすることも可能です。もしセカンダリー統合データベースが SAP SQL Anywhere データベースの場合、Mobile Link を双方のシンクロナイゼーションレイヤーに使用することができます。シンクロナイゼーションヒエラリキーは、Relay Server やロードバランサーを使用するよりもかなり複雑になります。また、このヒエラリキーは直接的にはスケーラビリティーには対応していません。なぜならは、プライマリー統合データベースとのシンクロナイゼーションの総データ量の問題があるからです。しかしながら、お客様のインフラストラクチャーや、ビジネスニーズによっては、シンクロナイゼーションヒエラリキーが合うかもしれません。例えば、Mobile Link サーバーが地理的に分散していなければならない、または、プライマリー統合データベースを低速プロセッサーや低速ネットワーク接続のクライアントからの長時間のシンクロナイゼーションを避けたい、などの場合です。

適用性

ここで報告したテスト結果が定量的な Mobile Link パフォーマンスのアイディアを与えられるとしても、お客様ご自身の Mobile Link セットアップのパフォーマンス評価を実施したい場合には、ご自身のスキーマ、データ、統合データベース、シンクロナイゼーションスクリプト、クライアント、ネットワークを使用したテストを行ってください。

この種のテストを行う場合、以下のプロシージャーを推奨します。

1. シンクロナイゼーションのニーズを決定してください。一定タイムに何人のユーザーがデータをシンクロナイゼーションするのか、同時（simultaneously）にどれだけのユーザーがシンクロナイゼーションするのか見積もる必要があります。アップロードやダウンロードされるデータのタイプやサイズ、そしてデータが挿入、更新、削除されるのかなどシンクロナイゼーションの典型的な特徴を決定する必要もあります。
2. 実装のためのパイロット版をセットアップしてください。できる限り、実際のシンクロナイゼーションスクリプト、実際の統合データベース、そして実際のクライアントとサーバーのハードウェアを使用してください。クライアントに関しても、実際のクライアントまたは検討中のクラアントハードウェアやネットワークを使用またはシミュレートして、典型的なデータで典型的なシンクロナイゼーションを実行するクライアントを作成してください。クライアントの数は、予測される総クライアント数を使用するのが現実的ではない場合、新しい Mobile Link リプレイユーティリティを使用して、典型的なシンクロナイゼーションを記録し、シミュレートした複数のクライアント経由でリプレイしてください。
3. Mobile Link サーバーとMobile Link モニターを実行してください。Mobile Link モニターは、各シンクロナイゼーションフェーズのタイミング情報を収集し、発生する警告やエラーを記録します。我々のテストでは、（Mobile Link サーバーとは異なるコンピューター上で）Mobile Link モニターを使用してもスループットに影響はありませんでした。SQL ボトルネックをロケートする必要がある場合には、あるレベルの verbosity でファイルへのログ取得を有効化する必要があるかもしれません（例えば -vcemns など）。SAP SQL Anywhere Monitor を使用して、Mobile Link サーバーを監視することも可能です。これは、Mobile Link サーバーの状態や可用性を監視する web ベースのツールです。これは、Mobile Link サーバーの -ppv オプションでログしたものと同じタイプの定期的な情報を収集して表示します。

www.sap.com/contactsap

THE BEST RUN SAP