SQL Anywhere によるアプリケーション・パフォーマンス問題の診断

Dan Farrar 著

2008/9/29

概要

SQL Anywhere のバージョン 10 を中心に、パフォーマンス問題の解決方法をご紹介します。パフォーマンス問題の性質について説明し、それらを CPU バウンド問題、I/O バウンド問題、および同時性バウンド問題に分類します。そして、DBA がパフォーマンス問題を分類する際の手順、製品に付属されている問題調査用のツール、および問題解決のために行う解析について説明します。

目次

図のリスト

1 はじめに

このマニュアルでは、データベース・アプリケーションの実行速度が遅いという報告をユーザから受けた際に、アプリケーション開発者または DBA が実行すべき手順について説明します。

この問題を出発点とし、「データベース・アプリケーション」とは何かという問題についても考察します。アーキテクチャ・レベルでは、データベース・サーバは、クエリが入ってデータが出てくるブラックボックス・クエリ・マシンであると考えられます。これは、アプリケーションの設計時に、このような考え方でデータベースをアプリケーションの付属物として扱う方が都合が良いためです。

しかし、データベース・サーバは、付属物よりもはるかに密接に一般的なアプリケーションに統合されているため、多くのデータベース・アプリケーションでは、データベース・サーバを分離して考えることは不可能です。それよりもむしろ、アプリケーション全体を端から端まで考慮する必要があります。つまり、データベース・サーバ側のリソース消費量やタイミングだけでなく、アプリケーションによるデータベース・サーバの使用パターンについても考慮する必要があります。

また、問題の残り半分、つまり「実行速度が遅い」ことが何を意味するかについても考察する必要があります。「遅い」というのは相対的な表現であるため、どのような期待によってこの比較がなされているかに注意する必要があります。ほとんどの場合このような期待は妥当なものですが、それを定量化するのは困難です。実際には、異なるシステムの類似するアプリケーションまたは操作セットとの比較は、ほとんどが感覚的に行われがちです。そのような比較の妥当性は、比較対象のシステムがどれだけ類似しているかという点のみに依存します。

実行状態の良いアプリケーションがインストールされていれば、より簡単にパフォーマンスの悪さを定量化できます。通常、これはアプリケーションのテスト・インスタンスか、または、以前はパフォーマンスが良かったにもかかわらず現在は悪くなっているアプリケーションの以前の動作です。そのようなインスタンスが見つかった場合、より簡単にアプリケーション・パフォーマンス問題を解決することができます。アプリケーションのパフォーマンスが低下した部分にのみ注目し、元のインスタンスとの違いを調べるだけで良いからです。

時間の経過とともにアプリケーションのパフォーマンスが低下していたり、別のインスタンスと比較してパフォーマンスが低下している場合に考えられる一般的な理由はいくつかあります。最も一般的な理由は、アプリケーションが、以下の点において元のサイズを超えて拡張されていることです。

- データの絶対量（データベース・インスタンスのサイズ）
- クライアント数／データベースへの接続数
- 要求量

もっと小さい拡張問題としては、データベース内のデータ分散が、テスト・データベースまたは元のアプリケーションのものと一致しなくなっていることが挙げられます。これは選択的な拡張の一種ですが、データベース・サイズの絶対的な増加と同じくらい大きな影響をパフォーマンスに与える可能性があります。

その他の可能性としては、アプリケーション自体がより多くのリソースを必要としていなくても、他のリソース・コンシューマと競争しなくてはならない場合も考えられます。たとえば、同一マシン上の他のアプリケーションにより、データベース・サーバのメモリ、I/O 帯域幅、または CPU の使用率が制限されている場合などです。アプリケーションの実行速度が遅くなるもう 1 つの原因としては、ネットワーク輻輳の増加が考えられます。特に、クライアントとサーバが短い要求を大量にやり取りするアプリケーションの場合は、これが当てはまります。

システマティックな方法でデータベース・アプリケーション・スケーラビリティを分析する方法の詳細につきましては、『SQL Anywhereにおける容量計画』（ホワイトペーパー）を参照してください。

2 アプリケーション・パフォーマンス問題の診断手順

このマニュアルでは、以下の 4 つの手順について説明します。

1. パフォーマンス問題とリソースの制約の性質を大まかに特徴付ける。
2. 手順 1 で決定した問題のカテゴリに基づいて、データベース・アプリケーションのパフォーマンスに関する情報を収集する。
3. 収集したデータを解析し、速度低下の原因を特定する。
4. 可能なソリューションを実装してテストを行う。

2.1 問題の特徴付け

大規模で複雑なデータベース・アプリケーションのパフォーマンス問題の解決にとりかかる場合、どこから始めたらよいかわからない場合があります。最も良い方法として、高レベルかつ取得しやすい問題に関する情報の収集から始めることをおすすめします。多くの場合、問題点を並べるという単純なプロセスからパフォーマンス問題の要因を見抜くことができます。

2.1.1 高レベルな特徴付け

高レベルな情報の収集とは、アプリケーション・パフォーマンスからパターンを探し出すことです。たとえば、以下のようなパターンを探します。

・問題の影響を受けるのは、すべてのユーザか、特定可能なユーザ・グループか、もしくはただ 1 人のユーザか ？
・問題発生の原因となる特定の操作が存在するか ？
・パフォーマンスに時間的なパターンが存在するか ？

> たとえば、営業日が進むにつれて、午前中はパフォーマンスが十分だが、午後は不十分になるなど、分散アプリケーションの場合、異なるタイム・ゾーンに存在するオフィスの営業時間に応じて、忙しい時間帯が移行することに注意してください。

これらの情報は、アプリケーションのどの部分を詳細に調査する必要があるかを知る手がかりになります。

2.1.2 リソース使用率の特徴付け

データベース・パフォーマンス問題は、大まかに 2 つのカテゴリに分類できます。1 つは、サーバの基本的な計算リソース（CPU サイクル、RAM、ディスク I/O 帯域幅、ネットワーク I/O 帯域幅）のいずれかが不足している場合で、もう 1 つは、これらのリソースがあるにもかかわらずそれを十分に使用できない場合です。
前者をリソース不足問題、後者を競合問題と呼びます。

問題解決プロセスの初期段階でどちらのカテゴリの問題を調べるかを決める必要があります。幸い、SQL Anywhere および基礎となるオペレーティング・システムの両方のツールを使用することで、ほとんどの場合、非常に簡単にパフォーマンス問題の基本的な性質を特徴付けることができます。このマニュアルでは、具体的な例を示す際に、データベース・サーバが Microsoft Windows 上で稼働していると仮定します。ただし、これらの例は、他のオペレーティング・システムにもすぐに適用できます。
たとえば、Windows でパフォーマンス・モニタ（perfmon）を使用する代わりに、Linux では iostat や top などを使用します。

OS によって表示される以下のパフォーマンス・カウンタは、サーバにおいてどのカテゴリの問題が発生しているかを知る手がかりになります。Windows では、タスク・マネージャ（taskman）または perfmon のいずれかを使用してこれらのカウンタを確認することができます。

・CPU 使用率
- データベース・サーバは、利用可能なすべての CPU を使用していますか ? これは、システム上の CPU 数かもしれませんが、サーバの構成やライセンスに応じて、システム上の CPU 数よりも少ない場合があります。答えが「はい」の場合、データベース・サーバにおいてアプリケーションが CPU バウンドである可能性があります。第 3.2 項を参照してください。
- マシン上で別のプロセスが CPU を使用しているため、データベース・サーバが CPU を使用できなくなっていませんか ? そのようなプロセスが存在し、それが当該アプリケーションの一部である場合、クライアント層でアプリケーションがバインドされている可能性があります。第 3.4 項を参照してください。

・ディスク・アクティビティ
データベース・サーバによって使用されるディスクが、どのくらいアクティブになっていますか ? Windows サーバでは、すべての Physical Disk オブジェクトの実行時に表示される %Idle Time の統計情報が非常に役立ちます。このアイドル時間の値が小さい場合、アプリケーションがディスクバウンドである可能性があります。第 3.1 項を参照してください。

非常に手早くディスク・アクティビティを評価するには、マシンを観察し、
いずれかのディスクに負荷がかかっているように見えるかどうかを判断します (LED が完全かまたはほぼ完全に点灯していたり、ディスクがうるさい音をたてているなど)。このようにディスクを直接観察する方法は、OS のツールを使用するディスク・アクティビティのモニタの代わりにはなりませんが、報告されたディスクのアイドル時間の短さが、物理的なディスク・アクティビティに明らかに関係していることを確認できます。

・アクティブな要求
アクティブな要求の数 (この値の取得方法については、第 3.3 項を参照) から、サーバが行うべき作業があるかどうかがわかります。この数が 0 以外の場合、マシンの CPU 使用率が低いかまたは比較的低い値 (最大約 30%) になり、サーバに属するすべてのディスクのディスク・アイドル時間が 5% 以上になります。
観察しているパフォーマンスは、同時性ボトルネックによって制約されている可能性があります。

2.2 アプリケーション・プロファイルの取得

一般的に、プロファイルとは、対象オブジェクトの特性を推定するための十分な情報がまとめられたものです。たとえば、人の顔のプロファイルに顔の輪郭に関する情報しか含まれていなくても、その情報からその顔の一般的な形態についてさまざまな推測ができます。

データベース・アプリケーションの場合、アプリケーション・プロファイルには、アプリケーションのためにデータベース・システムが実行する作業が簡潔に示されています。このプロファイルから、アプリケーションの多くの複雑な動作を推定して解析することができます。

扱っている問題の種類 (第 2.1 項を参照) がわかると、その問題を解決するために収集すべきデータ (プロファイル) がわかります。特定されたパフォーマンス問題のカテゴリについての詳細 (第 3 項) を参照し、問題解析のために具体的にどのような情報を収集すべきかを知る参考にしてください。

2.2.1 アプリケーション・プロファイリングの概念:アーキテクチャ

SQL Anywhere 製品では、さまざまな状況でアプリケーション・プロファイリングという用語を使用します。

この用語は、前述のアプリケーション・プロファイルの収集および解析作業を表す際に、最も一般的に使用させます。また、この解析を容易にするための Sybase Central 用 SQL Anywhere プラグインのモード名、バージョン 10 に追加された統合ツール名、および第 2.2.5 項で説明されているような既存のツール名としても使用されます。このモードからアクセス可能な [Application Profiling] ウィザードでは、単純な構成であればプロセスのほとんどが自動的に処理されます。

診断トレーシングは、SQL Anywhere データベース・サーバで以下の情報を統合的に取得するためのサポート機能を表す用語です。

・SQL 文
・ユーザ・コンテキスト
・ブロッキングおよびデッドロック・イベント
・クエリ・プラン
・パフォーマンス統計情報

診断トレーシングを実行すると、サーバは TCP/IP プロトコルを使用して、トレーシング・データベースが稼働しているデータベース・サーバへの専用の接続を確立します。このトレーシング・データベースは、プロファイル対象のデータベース（通常、運用データベースと呼ぶ）でも、運用データベースと同じサーバ上で稼働している別のデータベースでも構いません。理論上は、どのデータベースでもトレーシング・データベースとして使用できますが、プロファイリング解析ツールによっては、トレーシング・データベースを特別に作成しないと機能しないものもあります（第 2.2.3 項を参照）。

トレーシングの取得中は、グローバルに共有されるテンポラリ・テーブルにトレーシング・データが格納されます。トレーシング・セッションが終了すると、データがベース・テーブルに移動されてインデックス付けされ、解析の準備が整います。

2.2.2 [Application Profiling] ウィザード

[Application Profiling] ウィザードは、Sybase Central で初めてプロファイリング・モードに切り替えたときに実行されます。比較的小さいアプリケーションの場合や、このマニュアルで説明されている設定の多くをウィザードに行わせたい場合は、このウィザードを使用すれば、最も簡単にアプリケーション・プロファイルを収集することができます。このウィザードは、アプリケーション・プロファイルの作成手順をユーザに示し、第 3 項で説明されているいくつかの解析を実行します。

手動で結果を解析することもできます。ただし、ウィザードを使用しない場合、プロファイリング・プロセスをアプリケーションの特定の部分やユーザに限定する機能が制限されます。[Application Profiling] ウィザードを使用すれば、第 2.2.3 項で説明されているトレーシング構成を細かく気にする必要はありません。

このマニュアルで説明されている解析を行う場合、ウィザードで少なくとも 2 つ目のオプション ([Performance of your database application]) を選択する必要があります。また、ごくわずかな追加オーバヘッドでいくつかの推奨内容が追加される 3 つ目のオプション（[Overall database performance based on the database schema]）も便利な機能です。

2.2.3 アプリケーション・プロファイリングの概念：構成

SQL Anywhere サーバの診断トレーシング機能は、非常にさまざまな構成が可能です。調査対象のパフォーマンス問題の種類と範囲に関連するデータ項目のみトレースするよう要求できます。

取得するトレーシング情報をいくつかの範囲に限定することができます。

・データのレベル - SQL 文、クエリ・プラン、デッドロックなど

・データ収集の範囲 - データベース全体、特定のユーザ、および特定のテーブルなどのイベントをトレースする
・データ収集の条件 - 指定した期間の文のデータ、または推定実行時間との差が指定したパーセンテージである文のデータのみ収集する

トレーシング構成の詳細情報については、『SQL Anywhere サーバ - SQL の使用法』の「診断トレーシングの設定」を参照してください。

2.2.4 アプリケーション・プロファイリングの開始と停止

アプリケーション・プロファイルを取得するには、4 つの手順を実行します。この 4 つの手順とは、トレーシング・データを格納するためのデータベースの作成または配置、トレーシングの構成、トレーシングの開始とアプリケーションの実行、トレーシング・セッションの保存です。

トレーシング・データベースの作成

どのようなデータベースでもトレーシング・データベースとして使用できますが、トレーシング専用のトレーシング・データベースを作成するか、またはローカル・データベースを使用するのが最善策です。

ローカル・データベースは特別な設定の必要がないため、特にアプリケーションの開発中は、ローカル・データベースを使用するのが最も便利です。ただし、トレーシング中に取得されるデータのサイズにより、ローカル・データベースが膨張する可能性があります。データベース・ファイルは縮小できないため、トレーシングで取得されるデータが大量になることでローカル・データベースが膨張するのを防ぐ必要があります。

代わりに、トレーシング・データをローカル・データベースのコピーに送信したり、アプリケーション・プロファイルを保存するための専用のトレーシング・データベースを作成することもできます。専用のデータベースを作成すると、そのデータベースに不要なデータが格納されないという利点があります。

[Database Tracing] ウィザードを使用してトレーシングを構成すると、トレーシング・データベースの作成が促されます。ローカル・マシンにトレーシング・データベースを配置する場合、ウィザードがデータベースの起動も行います。トレーシング・データベースを別のマシン上で稼働させる場合は、トレーシング・データベースをそのマシンにコピーし、サーバを起動してからトレーシングを実行します。

トレーシング・データベースを手動で作成するには、dbunload コマンドで -n および -k オプションを使用します。

トレーシング・レベルの設定

『SQL Anywhere サーバ - SQL の使用法』の「トレーシング・レベルの選択」で説明されているように、トレーシング・レベルを設定するには 2 つの方法があり、Sybase Central の [Database Tracing] ウィザードを使用するか (SQL Anywhere プラグイン・ツリーでデータベース・オブジェクトを右クリックすると表示される)、または sa_set_diagnostic_level() プロシージャを呼び出し、sa_diagnostic_tracing_level テーブルのローを手動で変更します。

確実に有効な組み合わせのレベル／範囲／条件を指定するために、ウィザードを使用することをおすすめします。

トレースの開始

対象とするトレーシング・レベルをパフォーマンス調査の具体的な環境に合わせたら（調査する問題の種類については、第 3 項を参照）、トレーシング・セッションの開始準備が整います。

トレーシングを開始するには、トレーシング接続をトレーシング・データベースにアタッチ（ATTACH）し、セッションを終了するには、トレーシング接続をトレーシング・データベースからデタッチ（DETACH）します。[Database Tracing] ウィザードに、トレーシング・データベースの接続文字列が表示されます。

手動でトレーシングを開始するには、ATTACH TRACING 文を発行します。

例:

```
ATTACH TRACING TO 'uid=dba;pwd=sql;eng=mytracingeng;dbn=mytracingdb'
```

詳細については、『SQL Anywhere サーバ - SQL リファレンス』の「ATTACH TRACING 文」を参照してください。.

トレースの終了

目的の動作が発生したら、トレーシング・セッションを終了します。できるかぎり集中的にトレーシングを取得するのが理想的であるため、パフォーマンスの低下を示すサンプルが見つかったら、できるだけ早くトレーシングを終了します。たとえば、ビジーなサーバにおいてすべての情報を取得するようなトレーシング構成は、1 ～ 2 分以上実行すべきではありませんが、選択的な構成のトレーシングであれば、安心して数時間実行することができます。

トレーシング・セッションを取得したら、トレーシング接続を終了し、セッション情報を後で解析するためにデータベースに保存するかまたは捨てるかを選択します。通常はトレーシング情報を保存しますが、解決したいパフォーマンス問題がトレーシング・セッション中に発生しなかった場合などは、セッション情報を捨てます。

Sybase Central の場合、トレーシングを停止するには、データベース・オブジェクトを右クリックし、[Stop tracing with save] または [Stop tracing without save] を選択します。トレーシングを手動で終了するには、DETACH TRACING WITH SAVE 文（取得した情報を捨てる場合は DETACH TRACING WITHOUT SAVE）を使用します。

2.2.5 アプリケーション・プロファイリングに代わるもの

アプリケーション・プロファイリング／診断トレーシング・ツールの代わりになる技術がいくつかあります。ただし、SQL Anywhere バージョン 9 以前では、以下の技術しか使用できません。

要求ロギング

バージョン 10 よりも前のバージョンにおいてアプリケーション・プロファイリング（診断トレーシング）に最も類似する機能は、要求ロギング機能です。要求ロギングでは、サーバによって処理されたすべての要求（文）のテキストが、一般的なテキスト・ログ・ファイルに日付順に記録されます。さらに、ホスト変数の値やサーバによって実行されたクエリのプランなど、いくつかの情報をオプションで記録できます。この方法には以下の利点があります。

- 構成、操作、解析が単純である。
- サーバがクラッシュしたりマシンの電源が切れた場合でも復元可能である。ログが書き込まれるたびにフラッシュされるため、常に最新の状態である。

ただし、要求ロギングで取得されるデータは、アプリケーション・プロファイリングでアーカイブされるは

るかに大きいデータ・セットと比較すると限定的です。さらに、特に運用サーバでは、サーバによって処理されるすべての要求が 1 つのファイルに直列化されるため、要求ロギングのパフォーマンス・オーバヘッドが相当量あります。また、要求ロギングの場合、アプリケーションの特定の部分に焦点を当てる機能は少ししかありませんが、アプリケーション・プロファイリング・ツールの場合、アプリケーションの非常に具体的な部分のアクティビティを集中的にプロファイリングできます。たとえば、アプリケーション・プロファイリングを使用すれば、特定のユーザ、接続、テーブル・セット、プロシージャ・セット、またはクエリのコストなどに焦点を当てることができるため、運用システムでプロファイリングを行う場合の影響を非常に小さくすることができます。これらの理由から、アプリケーション・プロファイリングがサポートされるバージョンでは、アプリケーション・プロファイリングを使用することをおすすめします。

要求ロギングを有効にする方法の詳細については、『SQL Anywhere サーバ - SQL の使用法』の「要求ロギング」を参照してください。

プロシージャ・プロファイリング

アプリケーション・プロファイリングの代わりになる 2 つめの旧バージョンの技術は、プロシージャ・プロファイリング・ツールです。これは、ストアド・プロシージャまたはユーザ関数内の文が実行された回数と、その文によって生じたコストを記録する標準的なコード・プロファイリング・ツールによく似ています。この機能を使用して、プロシージャ・コードに加えられた可能性のあるプログラミング改良を特定したり、プロシージャ（またはプロシージャ呼び出しスタック）内から高コストな文を見つけ出すことができます。この方法ではアプリケーション・プロファイリング・ツールほどの情報は取得できませんが、使用方法が簡単であるため、上位レベルの概要プロシージャが多いアプリケーションのリソース使用率を把握できます。バージョン 10 以降では、Sybase Central のプロファイリング・モードからこの機能にアクセスできます。

バージョン 10 以降でも、ストアド・プロシージャ・コードのフロー制御を解析する場合に、プロシージャ・プロファイリングは引き続き有効な手段です。

インデックス・コンサルタント

インデックス・コンサルタントは、バージョン 9 ではスタンドアロン・ツールでしたが、バージョン 10 以降ではアプリケーション・プロファイリング機能に完全に統合されています。このツールを使用すると、単一のクエリまたはクエリ・セット（作業負荷）の実行をサポートする推奨インデックスが、サーバによって提示されます。バージョン 9 では、このツールは独自の作業負荷を取得していましたが、バージョン 10 以降では、診断トレーシング機能によって取得されたより詳細な情報を使用できるようになっています。インデックス・コンサルタントの使用方法については、第 3.1.2 項を参照してください。

SQL Anywhere コンソール・ユーティリティ

アプリケーション・プロファイリング・ツールによるパフォーマンス問題の診断を補完するもう 1 つのツールが dbconsole です。診断トレーシングが一定の期間にわたって詳細なパフォーマンス・データ・セットを取得するセッションベースのツールであるのに対して、dbconsole はデータベース・サーバの現在の状況を示すモニタリング・ツールです。dbconsole の使用により、第 3.1.1 項で説明されている I/O カウンタや、同時性の問題を検出するための ActiveReq（要求：アクティブ）カウンタなど、パフォーマンス・カウンタの状態をすばやく確認することができます。

2.3 データの解析

発生している問題の種類に関連するデータを収集したら、そのデータを解析してパフォーマンス問題の正確な原因を見つけ、解決策を提示する必要があります。

解析において調査すべき内容については、第 3 項で説明されている問題の各カテゴリの説明を参照してください。

一般に、以下の項目の調査を行います。

・高コストな文とそのプラン
・同時性の問題の兆候
・パフォーマンス・カウンタ

2.3.1 Sybase Central を使用した解析

アプリケーション・プロファイルを解析する最も良い方法として、Sybase Central 用 SQL Anywhere プラグインのプロファイリング・モードの使用が挙げられます。このブラウザには、比較的単純ながらも強力なデータのビューが表示されるため、第 3 項で説明されているような解析に役立つ情報が得られます。

Sybase Central からトレーシング・セッションを開くには、プロファイリング・モードに切り替えます。ウィザードが開始された場合はそれをキャンセルし、[Open an analysis file or connect to a tracing database] を選択します。トレーシング・データベースへの接続を選択し、そのデータベースの接続情報を指定します（すでに接続されているデータベースに情報が格納されている場合でも、プロファイリング解析モードにより、そのデータベースへの別の接続が開かれます）。

[Status] ウィンドウ枠

プロファイリング・モードでトレーシング・データベースを初めて開くと、[Status] ウィンドウ枠が表示されます。このデータベースに複数のトレーシング・セッションが格納されている場合は、トレースのタイムスタンプとサイズを確認しながら、解析するトレーシング・セッションを 1 つ選択します。

このウィンドウ枠からスタートし、[Database Tracing Data] ウィンドウ枠内にある [Summary]、[Details]、および [Connection Blocks] ウィンドウ枠を調査します。

[Summary] ウィンドウ枠

[Summary] ウィンドウ枠（図 1 を参照）には、トレース中に取得されたすべての文に関する高レベルのビューが表示されます。類似する文がグループ化されるため、文の種類ごとの正味効果がわかります。サーバにより、取得された文ごとのシグニチャが計算され、類似するシグニチャを持つ文がクループ化されます。各グループを代表する文が表示されるため、グループ内のすべての文がこの代表文と同じ文である訳ではありません。

クエリ（SELECT、INSERT、UPDATE、または DELETE 文）の場合、文で参照されるすべてのテーブルおよびカラムを調査して、シグニチャが計算されます。
シグニチャの計算では、変数や定数の値は無視されます。したがって、[Summary] ウィンドウ枠のビューでは、以下の 2 つの文は、同じグループとして表示されます。

```
SELECT * FROM product WHERE id = 1; SELECT * FROM product
WHERE id = 2;
```

クエリ以外のすべての文には、単一のシグニチャが割り当てられます。たとえば、すべての CREATE INDEX 文に同じシグニチャが割り当てられます。なお、すべての CALL 文にも同じシグニチャが割り当てられることに注意してください（これは望ましくないため、将来のバージョンでは修正される可能性があります）。プロシージャを構成する個々の文ごとにエントリが作成されます。

いずれかのカラムをクリックすると、結果がそのカラムによってソートされます。もう一度そのカラムをク

リックすると、ソート順が逆になります（降順から昇順へ、または昇順から降順へ）。また、[Summary] ウィンドウ枠でいずれかの文を右クリックすると、その文の実行に関する詳細情報が表示されます。

[Summary] ウィンドウ枠の右上にあるシェブロンをクリックすると、ビューのフィルタリング・プロパティが表示されます。表示する時間帯を限定したり、特定の文を検索したい場合、このウィンドウ枠にフィルタリング情報を入力し、それを適用（[Apply]）します。表示されたデータが予想よりも少なかった場合、フィルタをリセット（[Reset]）します（[Details] および [Connection Blocks] ウィンドウ枠でも同様のフィルタリング・ウィンドウ枠を使用できます）。

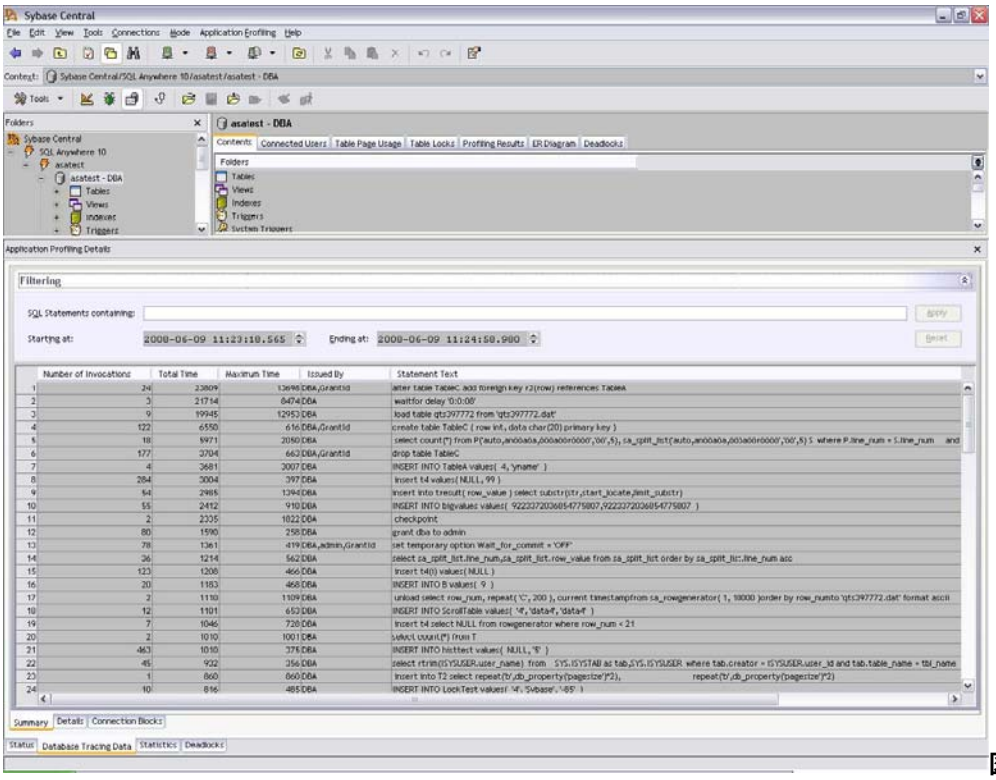

図 1: [Summary] ウィンドウ枠

[Summary] ウィンドウ枠を使用する主な目的は、[Total Time]（ミリ秒単位）で文のグループをソートすることにより、アプリケーションで（全体として）最も時間を消費する文の種類を見つけ出すことです。原因としては、その文自体が高コストである、つまり 1 回の実行が高コストであるか、もしくは、その文だけでは低コストだが、何回も実行するため総計すると高コストになることが考えられます。このような文を改善することで、アプリケーション全体に著しい改善が見られます。

[Details] ウィンドウ枠

[Details] ウィンドウ枠（図 2 を参照）は、レイアウトが [Summary] ウィンドウ枠と類似しており、同様のソートおよびフィルタリング機能を使用できます。このウィンドウ枠では、記録される SQL 文の実行ごとに行が 1 行ずつ表示されます。詳細情報として、文の実行時間、サーバが要求をアクティブに処理するのに

要した合計時間（[Duration]）、および文の SQLCODE（0 以外の場合）が記録されます。

カーソルを開いたクエリの場合、カーソルのオープン時間とクローズ時間が記録されます。カーソルのオープン時間とクローズ時間の差が、報告された合計時間よりもはるかに大きい場合があります。これは、サーバがクエリに対する回答をすばやく処理したにもかかわらず、クライアントが長時間待ってからカーソルを閉じた可能性があるためです。これは、特に同時性問題のトラブルシューティングでよく見られます（第 3.3.3 項を参照）。

このビューで右クリックして［SQL Statement Details］ウィンドウを開くと、個々の文の詳細情報を参照することができます。

［SQL Statement details］ウィンドウ

このウィンドウ（図 3 を参照）には、[Details] ウィンドウ枠に表示されるすべての情報がより見やすい形式で表示されます（スペースに余裕があるため）。さらに、トレースが情報を取得するよう構成されている場合、文で使用された変数の値などの詳細情報も表示されます。

SQL 文がクエリの場合、その文のプラン情報が表示されるタブも使用できます（図 4 を参照）。プラン情報は、CPU バウンドおよび I/O バウンドのアプリケーションのトラブルシューティングに特に役立ちます。クエリに対して表示される短いテキスト・プランは、必ずそのクエリが使用した実際のプランになります。トレーシングがグラフィカルなプランを取得するよう構成されていた場合（統計情報の有無にかかわらず）、取得されたグラフィカルなプランも表示されます。グラフィカルなプランを取得しなかった場合、サーバがクエリの再最適化を試み、文が実行されたときのサーバの状態をシミュレーションしながら、使用すべきプランを表示します。解析で使用する前に、この「最良の推測プラン」と短いテキスト・プランを比較する必要があります。

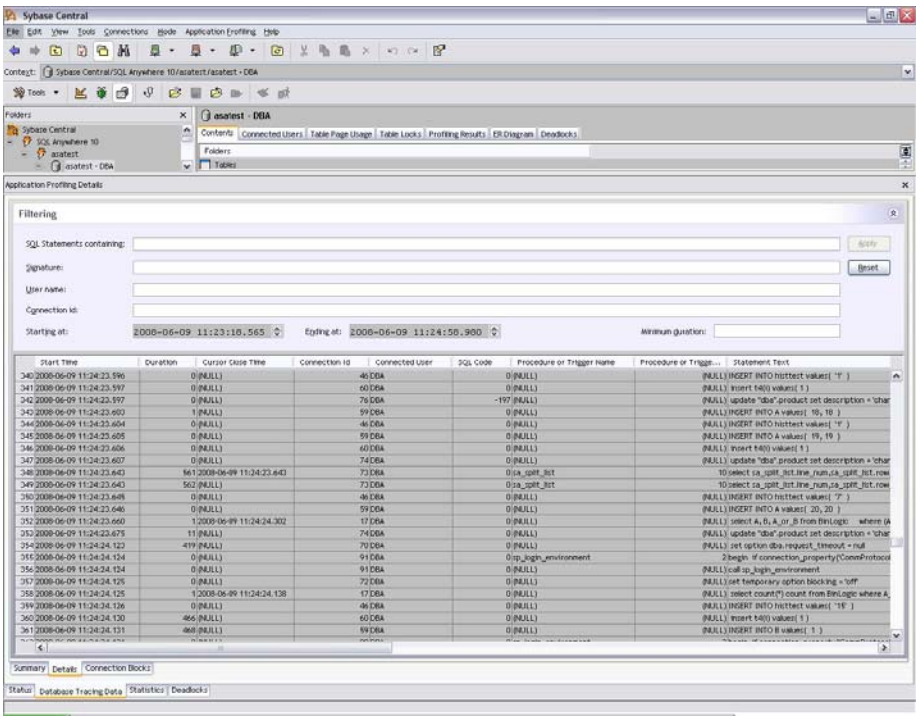

図 2:[Details] ウィンドウ枠

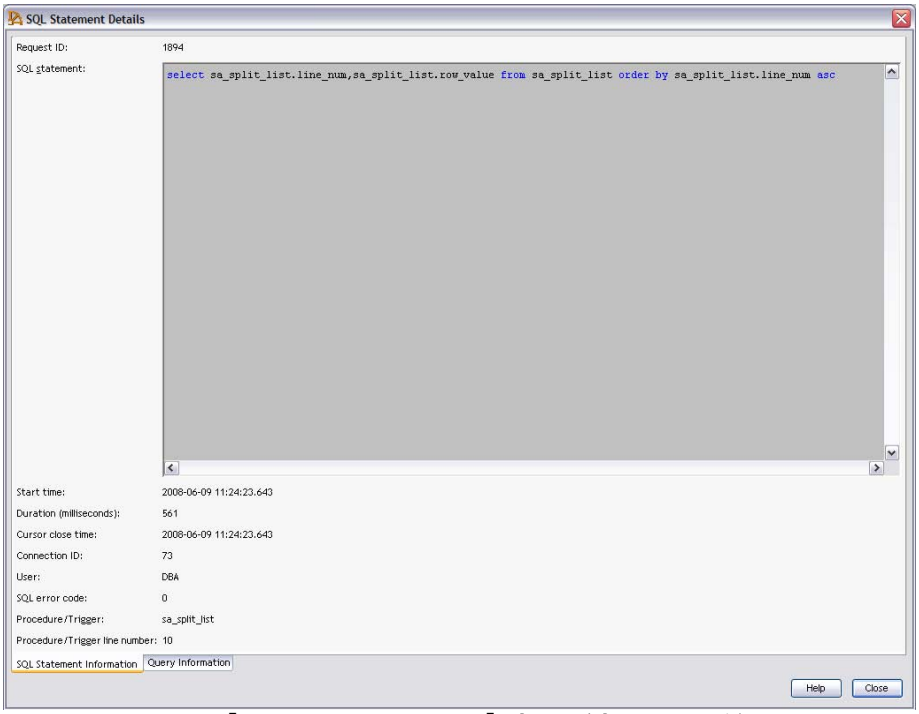

図 3:[SQL Statement Details] ウィンドウ - 文の詳細

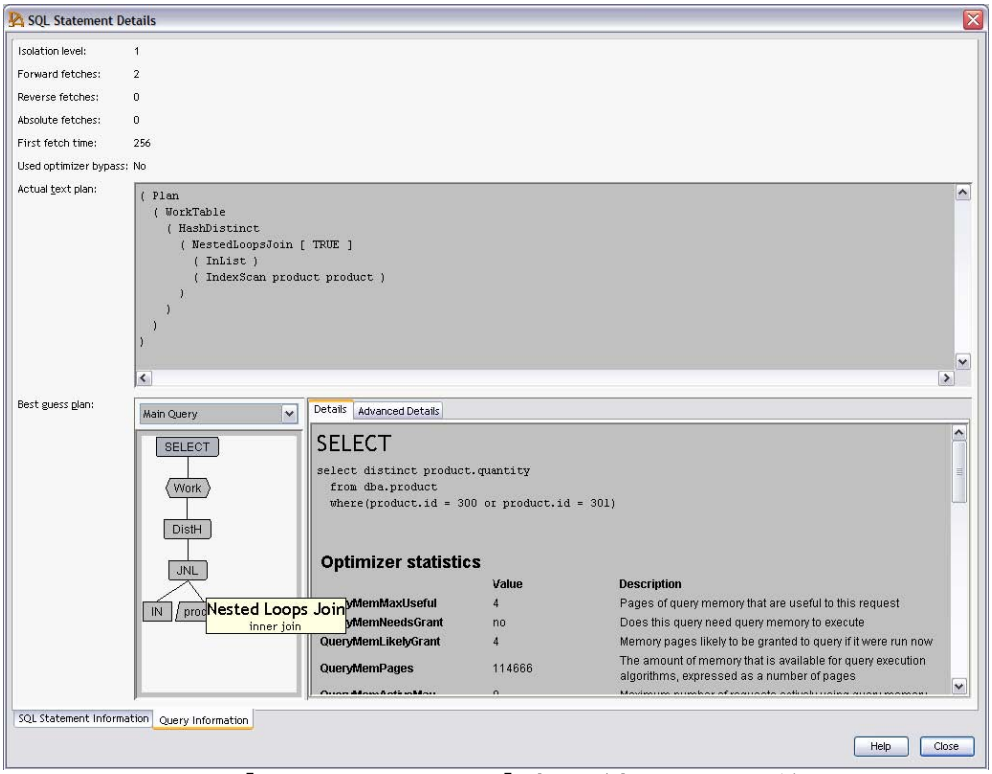

図 4:[SQL Statement Details] ウィンドウ - クエリの詳細

[Connection Blocks] ウィンドウ枠

このウィンドウ枠（図 5 を参照）には、取得されたすべてのブロッキング・イベントが表示されます（ブロッキングのレベルが指定されていた場合）。これらのイベントは、アプリケーションベースの同時性の問題の調査に特に関係があります（第 3.3.3 項を参照）。ブロックごとに、ブロックされた文、ブロックされた接続、およびブロックしている接続が表示されます。ブロッキング・イベントを右クリックすると、そのイベントがブロックされた状況についての詳細情報が表示されます。

ブロックしている接続が実行していた文を直接特定することはできませんが、ブロックが発生した時間を確認すれば、[Details] ウィンドウ枠でブロックが発生したときにその接続が実行していた文（または接続によって開かれたカーソル）を特定できます。

[Deadlocks] ウィンドウ枠

トレースがデッドロック・イベントを含むよう構成されている場合、トレース中に検出されたすべてのデッドロック・サイクルがこのウィンドウ枠に表示されます。ケースごとに、デッドロックに含まれるすべての接続、各接続のブロック要因、およびデッドロック対象になった接続が表示されます。トレーシング・データをローカル・データベースまたはローカル・データベースのコピーに送信する場合、各接続がブロックされたローのプライマリ・キー値（ある場合）も表示されます。デッドロック数が多い場合、特にそれらが 2 つ以上の接続に関連している場合は、アプリケーションに根本的な同時性の問題が存在することを意味します（第 3.3.3 項を参照）。

[Statistics] ウィンドウ枠

統計情報（数値的なパフォーマンス・カウンタ）を取得した場合、[Statistics] ウィンドウ枠（図 6 を参照）で、それらをグラフィカルに表示できます。不揮発性のカウンタ（秒単位で変化しないカウンタ）は、より高速な揮発性のカウンタと同じ比率で表示されます。したがって、不揮発性のカウンタのグラフは階段状に表示されます。

サーバはさまざまなパフォーマンス・カウンタを取得しますが、一般的なパフォーマンス解析に役立つカウンタは比較的少数です。ほとんどのカウンタは、SQL Anywhere の内部開発や、このマニュアルでは説明されないような非常に具体的なパフォーマンス状況のトラブルシューティングにしか使用されません。

イベントをカウントするパフォーマンス・カウンタは、単調で減少しない累積カウンタです。パフォーマンス解析の場合、これらのカウンタの絶対値よりも増加速度の方が重要であることが一般的です。たとえば、60 秒離れた 2 つの時点を選び、グラフでそれらの時点の上にマウスを移動すると、それぞれの時点のカウンタ値が表示されます。その 2 つのカウンタ値で減算を行えば、1 分あたりの平均増加がわかります。

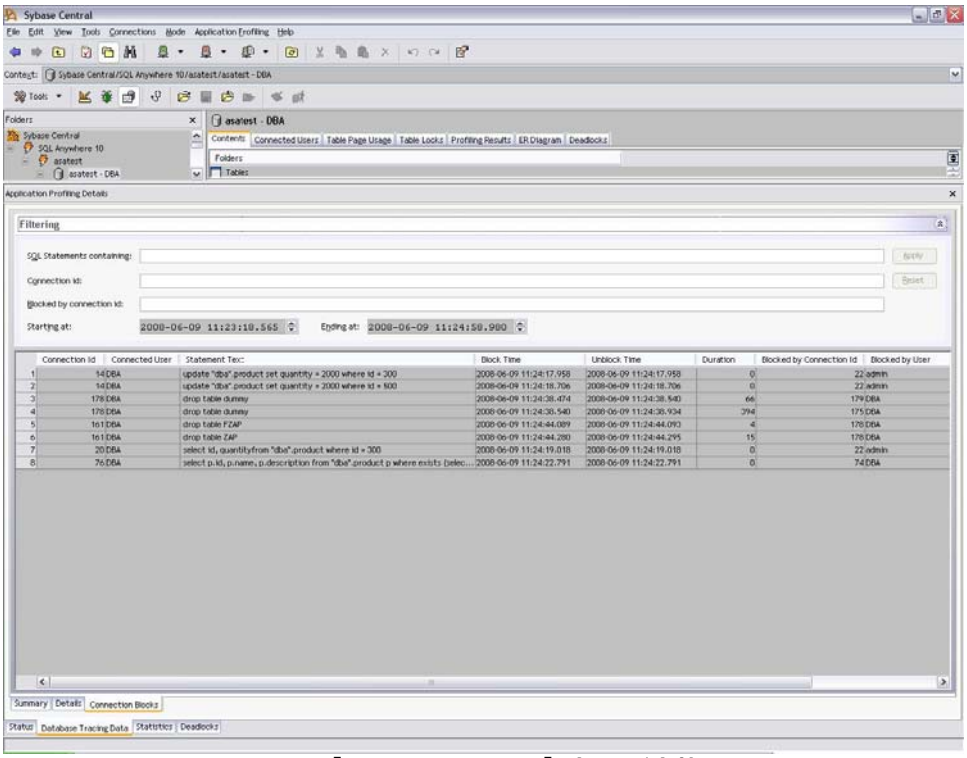

図 5:[Connection Blocks] ウィンドウ枠

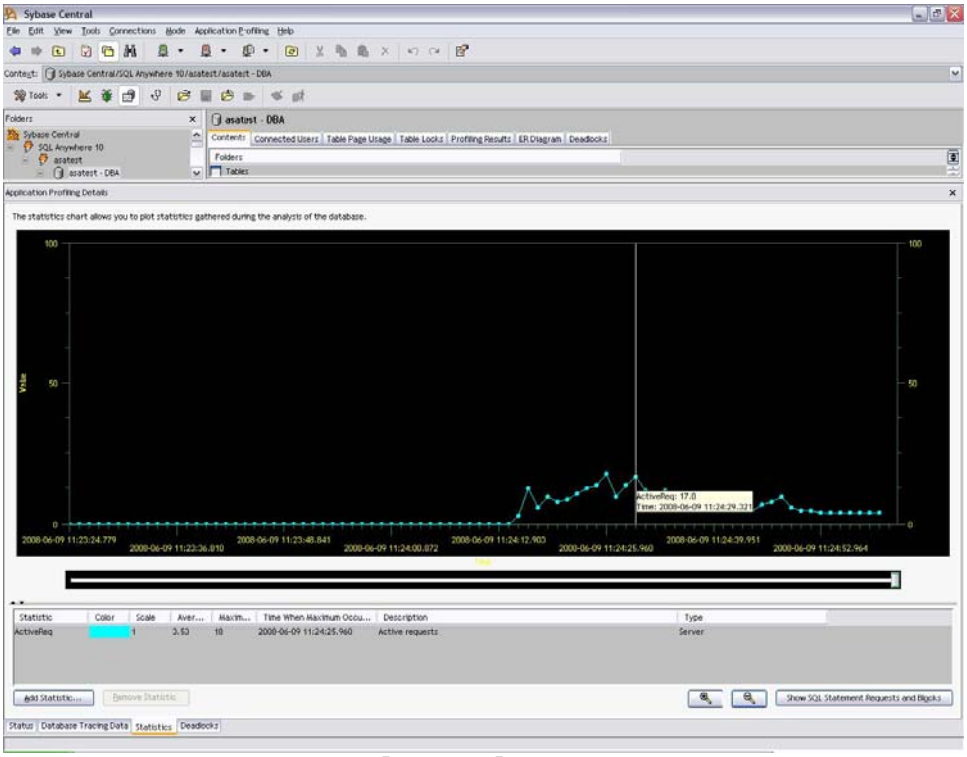

図 6:[Statistics] ウィンドウ枠

オブジェクトをカウントするパフォーマンス・カウンタの値は、カウンタが取得されたときの基本値です。たとえば、ファイルの断片化数、ヒープ・ページ数、総キャッシュ・サイズなどがあります。これらのカウンタの場合、カウンタの変化速度は、絶対値に比べてあまり重要ではありません。トレース中のある時点で絶対値が急激に変化した場合、そのときに何か重要なことがサーバで発生したことを意味します。

パフォーマンス問題のカテゴリごとに重要なカウンタについては、第 3 項の各カテゴリの説明を参照してください。

2.3.2 SQL クエリによる手動解析

Sybase Central のプロファイリング・モードで提供される解析は、すべて手動でも実行することができます。以下のような場合は、sa_diagnostic_* テーブルを直接照会して解析を行う方が便利かもしれません。

- 解析するトレースが非常に大きい場合（要求が 500,000 以上ある場合）、GUI ブラウザの動きが遅すぎて快適に作業できない可能性があります。
- 非常に具体的な環境を探している場合（特定の時間に特定のユーザから、特定の変数値が含まれたクエリが発行される場合など）、テーブルを直接照会する必要があります。
- アプリケーション解析のプロセスを自動化したい場合（高コストなクエリのカスタム・レポートを生成する場合など）、SQL クエリを使用してそれらのレポートを生成する必要があります。

sa_diagnostic_* テーブルについては、『SQL Anywhere サーバ - SQL リファレンス』の「診断トレーシング・テーブル」の項で説明されています。 ほとんど場合、これらのテーブルのジョインは自己説明的でなければなりません。

各トレーシング・セッションは、ロギング・セッション ID によって識別されます。この値を使用して、指定

されたローが属するセッションを識別するため、この値はすべての診断トレーシング・テーブルに含まれています。複数のセッションの結果が混在しないように、各テーブルに対する手動クエリは、ジョインまたは追加述部を使用して、単一のロギング・セッション ID カラムに限定してください。

たとえば、ロギング・セッション ID が 2 であるロギング・セッションで実行された高コストな上位 5 つの SQL 文を調べるには、以下のクエリを発行します。

```
SELECT TOP 5 *
FROM sa_diagnostic_request R, sa_diagnostic_statement S
WHERE R.statement_id = S.statement_id
AND R.logging_session_id = S.logging_session_id
AND R.logging_session_id = 2
ORDER BY R.duration_ms DESC;
```

2.3.3 その他のツールによるアプリケーション解析

SQL Anywhere バージョン 9 以前を使用している場合、または何らかの理由で統合されているアプリケーション・プロファイリング・ツールを使用したくない場合、第 2.2.5 項で説明されているツールを使用して、アプリケーション・パフォーマンス解析を行うことができます。ここでは、それらのツールでパフォーマンス解析用に取得されたデータの使用方法について説明します。

要求ロギング

要求ロギング機能では、サーバが外部のフラット・テキスト・ファイルから受信したすべての文が保存されます。このファイルは人間が読み取れる形式のファイルですが、ファイルを解析しやすくするためのプロシージャがサーバに用意されています。要求ロギングで取得されるデータは、アプリケーション・プロファイリングで取得されるデータの一部です。

sa_get_request_profile() プロシージャは、要求ログ・ファイルの名前を取得してそのファイルを解析し、サーバによって実行されたユニークな文と、それぞれの合計実行時間、平均実行時間、および最大実行時間のリストを satmp_request_profile テンポラリ・テーブルに作成します。このテーブルは、アプリケーション・プロファイリング解析の [Summary] ビューに似ています。ただし [Summary] ビューでは、文の SQL テキストが同じである場合、それらの文が「類似している」とみなされます。

同様に、sa_get_request_times() プロシージャは、要求ログ・ファイルの名前を取得してそのファイルを解析し、文とその実行時間を satmp_request_time テンポラリ・テーブルに入力します。これは、アプリケーション・プロファイリング解析の [Detail] ビューに似ています。

いずれの場合も、これらのプロシージャが、接続によって結果をフィルタするオプションのパラメータを受け取ります (Sybase Central のアプリケーション・プロファイリング・ビューアで使用するフィルタリング用のウィンドウ枠と同様)。

プロシージャ・プロファイリング

プロシージャ・プロファイリング・ツールで記録される情報は、Sybase Central 内から表示すると最も見やすく表示することができます。プロファイリング・モードに切り替えてプロシージャを開くと、プロシージャのソース・コードが注釈付きで表示されます。注釈として、各文の実行回数と実行所要時間に関する詳細情報が追加カラムに表示されます。このビューは、従来のソースコード・プロファイラによって表示される情報と似ています。なお、プロシージャ・プロファイリング・ツールは、Sybase Central のプロファイリング・モードから表示できますが、データベース・サーバのアプリケーション・プロファイリング機能とは無関係です。

2.4 アプリケーションのチューニングとテスト

発生しているパフォーマンス問題の種類に応じて、パフォーマンスの悪いアプリケーションに対して以下のような解決策が考えられます。

- ・データベース構成の変更（コマンド・ライン・オプション、物理的なファイル配置、その他）
- ・データベース構造の変更（新しいインデックスの追加、統計情報の再生成、その他）
- ・ハードウェアの追加
- ・アプリケーションの一部の再コーディング

アプリケーションのチューニングとテストは、おそらく繰り返し行うことになります。データベースに変更を加える場合、特に運用サーバの場合は、作業をゆっくり進めることをおすすめします。構成を変更したことで運用システムの他の部分が不安定になることがないように、不要なコード変更や高コストのハードウェアの購入は最小限に抑える必要があります。影響を最小限に抑えながら、増加する作業負荷に見合うシステムを整備する方法については、『SQL Anywhereにおける容量計画』（ホワイトペーパー）を参照してください。

3 パフォーマンス問題の種類

この項では、パフォーマンス・ボトルネックの具体的な原因について考察します。第 2 項で説明されている方法を使用して、パフォーマンス問題が以下のどのカテゴリに属するかを特定し、そのカテゴリに合ったデータ収集および解析を行います。

3.1 I/O バウンド・アプリケーション

データベース・パフォーマンスは、単純に I/O 帯域幅に制約されている場合があります。従来から、多くのデータベース・システムにおいて I/O 帯域幅はパフォーマンスを制限する主な要因でした。データベースがキャッシュ・サイズよりも（数十～数百ギガバイト）大きい場合にこのような問題がよく発生しますが、より小さなデータベースの場合でも、I/O 帯域幅が制約要因になる可能性があります。

データベースが I/O バウンドの場合、データベース・ファイルを格納しているディスクのアイドル時間の値が非常に小さくなります（数パーセント以下、おそらく 0.0% に近い値）。もう 1 つの確認方法として、マシンの近くでマシン自体を観察すると、ディスクが非常に激しく動作していることがわかります。

データベース・アプリケーションが I/O バウンドである原因としては、さまざまなことが考えられます。その原因のうちいくつかは、サーバの構成を変更するだけで問題を解決したり対処法を施すことができますが、たいていは他に方法がなければディスクを追加する必要があります。このタイプのパフォーマンス問題の良い点は、リソース（この場合はディスク）さえ追加すれば、追加した分だけパフォーマンスが向上することです。

3.1.1 データベース・キャッシュが小さすぎる

新しいディスクを購入する前に、確認すべきことがいくつかあります。設定されているデータベース・キャッシュ・サイズが小さすぎる可能性があり、その場合、スラッシングという現象が発生する可能性があります。データベース・サーバは、将来の要求時にディスクからページを読み込まなくて済むように、将来の要求への回答に必要だと思われるページをすべてキャッシュ・メモリ（バッファ・プール）内に保持します。しかし、将来の要求に回答するためにキャッシュに保持する必要があると思われるページが多すぎる場合、それらの一部をディスクに書き込み（リソース制限内に収めるため）、後からキャッシュに戻す必要があります。サーバが、キャッシュ・メモリとディスク間で同じページを絶え間なく往復させている場合、サーバがスラッシング状態であると言えます。これはあまり良い状態ではなく、パフォーマンス問題の重大な原因になり得ます。

スラッシングが発生しているかどうかを確かめるには、サーバによって提供されるいくつかのカウンタを

調べます。実行中にリアル・タイムでこれらのカウンタを見るか、またはアプリケーション・プロファイリングの診断トレースで取得した情報を見ます。特に、キャッシュが大きくないと思われる場合に調査すべきカウンタは、テーブル・ページやインデックス・ページに関連するカウンタです。これら 2 種類のページは、アプリケーションが使用する大量のデータの大半を占めます。この他にも、リソース割り当てやロック・メンテナンスなどのためにサーバの内部で使用されるページもありますが、アプリケーションのデータは、テーブル・ページとインデックス・ページが混在するページに格納されます。

統計情報としては、CacheReadTable （キャッシュ読み込み: テーブル／秒）および DiskReadTable（ディスク読み込み:テーブル／秒）カウンタを調査します。CacheReadTable カウンタは、サーバが（キャッシュまたはディスクから）テーブル・ページを読み込もうとするたびに増加します。サーバが読み込みを行うのは、ローのフェッチ、更新、またはインデックス付けなどを行うためです。DiskReadTable カウンタは、サーバがテーブル・ページをメモリに書き込むためにディスクにアクセスするたびに増加します。これら 2 つの数値は、どちらもデータベース起動時からの絶対カウントです。サーバの定常動作を調べるには、60 秒ほど離れた 2 つの固定された時点を選び、CacheReadTable カウンタの増加と DiskReadTable カウンタの増加を比較します。バッファ・サイズが適切であれば、CacheReadTable カウンタは、DiskReadTable カウンタの少なくとも 10 倍の速さで増加します。サーバが定常状態でなければならないことに注意してください（つまり、サーバの起動直後ではなく、同一アプリケーションが少なくとも 20 分間以上動作している必要があります)。

同様の比較を、インデックス・リーフ・カウンタ間（CacheReadIndLeaf （キャッシュ読み込み:インデックス・リーフ／秒）とDiskReadIndLeaf （ディスク読み込み:インデックス・リーフ／秒））およびインデックス内部ページ・カウンタ間 （CacheReadIndInt （キャッシュ読み込み:インデックス内部／秒）とDiskReadIndInt （ディスク読み込み:インデックス内部／秒））でも行うことができます。これらのカウンタの場合、*ReadTable カウンタに比べて、Cache* カウンタが Disk* カウンタよりも比較的速く増加します。IndInt* カウンタの場合、定常状態の適切なキャッシュ・サイズで少なくとも 100 の差が生じます。

適切な構成のバッファ・プールでは、インデックスの非リーフ・ページがほぼ必ずキャッシュに存在します。

これら 3 セットの統計情報を調査し、いずれかで（特にインデックス・ページで）ディスク読み込み数がキャッシュ読み込み数に近く、サーバが I/O バウンドであると思われる場合、サーバにスラッシングの問題が発生していることを意味します。したがって、サーバのキャッシュ・サイズを増加する方法を考える必要があります。これは、単純にデータベース・サーバのコマンド・ラインで最大および最小キャッシュ・サイズ・パラメータ（それぞれ -ch および -cl）を増加するだけで済む可能性もあります。しかし、使用できる物理 RAM がマシン上にそれ以上ない場合は（他のプロセスが使用していたり、データベース・サーバが使用可能な最大サイズのキャッシュを使用している場合など)、サーバのマシンに物理 RAM を追加することで問題を解決できる可能性があります。

3.1.2 インデックスの不足

I/O サブシステムを限界まで駆動しているアプリケーションでは、データベース内にもっと多くのインデックスが必要である場合があります。インデックスを正しくチューニングすることで、サーバがクエリに回答するためにディスクから読み込むページを選択できるようになるため、特定の種類のクエリに必要な I/O 量が著しく減少します。しかし、インデックスのマイナス面として、テーブルに追加されるそれぞれのセカンダリ・インデックスがスペースを取るという問題があります（メモリに読み込まれる場合はディスク・スペースとキャッシュ・スペースの両方)。さらに、セカンダリ・インデックスが追加されると、インデックスが作成されるテーブルのデータを更新するすべての文のメンテナンスが必要になります。したがって、適切に構成されていないインデックス・セットをデータベースに追加すると、実際には、データベース・システムの I/O 作業負荷が減少するどころか増加する可能性があります。

アプリケーションに関係があるインデックスを特定しやすくするために、SQL Anywhere には IC (Index Consultant ：インデックス・コンサルタント）が用意されています。IC は、アプリケーション・プロファイルの

個々のクエリやクエリ・セットを解析し、アプリケーションのスループットが上昇すると思われるインデックス・セットを推奨します。完全なアプリケーション・プロファイルを使用して IC を実行すれば、推奨内容の作成時に、データ更新のメンテナンスが生じる場合のマイナス面も考慮されます。更新が多いために作業負荷がかかる場合、IC はインデックスの推奨に対してより保守的になります。

IC は、以下の 4 とおりの方法で実行できます。

・Sybase Central のプロファイリング・モードから、全体的な作業負荷を解析できます。[Profiling] ウィザードを使用している場合、解析が自動的に行われます。保存されたトレースを参照している場合、[Profiling] メニューから解析を開始できます。
・プロファイリング・モードから、単一のクエリを解析できます。アプリケーション・トレースで、パフォーマンスが特に気になる特定のクエリまたは小さいクエリ・グループがあった場合、[Details] ビューでそれらのクエリを個々に選択し、各クエリについて [Profiling] メニューから IC を実行します。
・IC は dbisql に組み込まれています。dbisql ウィンドウで、アクティブな単一のクエリに役立つインデックスの推奨内容を取得するには、[Tools] メニューから [Index Consultant] を選択します。
・可能なインデックス作成についての what-if 解析を手動で実行するという IC の根本的な機能を使用します。CREATE INDEX 文の変形文である CREATE VIRTUAL INDEX 文では、提案する 1 つまたは複数のインデックスがデータベースに物理的に存在した場合、オプティマイザがこの文でそれらのインデックスを使用できるとみなすかどうかを調べることができます。

詳細については、『SQL Anywhere サーバ - SQL リファレンス』の「CREATE INDEX 文」を参照してください。

インデックスを手動でチューニングする場合、データベースを更新するクエリに関するマイナス面が自動的には考慮されないことに注意してください。

3.1.3 クエリ処理用に使用できるメモリ

過度なディスク I/O について考えられる 3 つ目の原因は、安価なインメモリ方式を使用してサーバに送信されたクエリを処理するのに十分なメモリがクエリ処理エンジンにないことです。その結果、クエリ処理エンジンは、メモリ不足時方式に戻らなければなりません。この方式は、I/O 要求の増加というデメリットを受けながら、カーソルのインメモリ占有容量が最小限になるように設計されています。つまり、インメモリ・アルゴリズムは、多くのバッファ・プールを過度に占有しないために、クエリ実行中にメモリ内容を一時的にディスクに書き込み、後で再びそれを読み込みます。SQL Anywhere バージョン 10 では、サーバがキャッシュ全体のうちのどれくらいをクエリ・メモリとして使用できるかを判断し、それをサーバのマルチプログラミング・レベル (-gn コマンド・ライン・オプションで設定) で割ることで、指定した接続のクエリ処理用に使用できるメモリ量を決定します。

-gn の値が非常に大きい場合 (50 を超える程度)、多数の同時要求をサーバが並行処理することを実際に予測する必要があります。サーバでは、-gn オプションによって指定された数よりも多くの接続がアクティブになる可能性があります。-gn オプションは、サーバが並行処理する要求の数を指定するオプションです。

3.1.4 次善のファイル配置

I/O パフォーマンスの低下について考えられるもう 1 つの理由は、データベース・アプリケーションによって使用されるデータベース・ファイルが最適な場所に配置されていないことです。

たとえば、トランザクション・ログはメイン・データベース・ファイルとは別のドライブに配置するのが最善で

す。I/O が非常に頻繁に行われるアプリケーションや非常に大きいデータベースを使用するアプリケーションの場合、データをセカンダリ dbspace に格納し、チェックポイント・ログが格納されるシステム dbspace (.db ファイル) には、アプリケーション・データをあまり格納しないようにするのが有益な方法です。データを挿入、更新、または削除するトランザクションがサーバによって処理されているときはいつも、トランザクション・ログとチェックポイント・ログの両方に多数のディスク・アクティビティが記録されます。最善のパフォーマンスを得るには、可能なかぎりこれらのファイルを別のドライブに移動し、可能なかぎりテンポラリ・ファイルを専用のドライブに配置します。いかなる場合も、特にテンポラリ dbspace またはトランザクション・ログを RAID 5 デバイスに配置しないようにします。冗長な RAID 構成を使用して安全を確保する必要がある場合、トランザクション・ログ・ファイルには小さい RAID 1 ドライブをおすすめします。

3.2 CPU バウンド・アプリケーション

アプリケーションがデータベース・サーバにおいて CPU バウンドである場合はすぐにわかります。Windows の場合は taskman または perfmon、他のオペレーティング・システムの場合はそれに相当するものを開くだけで、データベース・サーバが割り当てられたすべての CPU を使用していることがわかります。また、サーバが割り当てられている CPU の 90% 以上を使用している場合、アプリケーションが CPU バウンドである可能性があります。実際にアプリケーションが CPU バウンドである場合、サーバが割り当てられている CPU の 99% もしくは 100% を使用しています。

CPU バウンド・アプリケーションの問題点は、その原因が多いため、CPU バウンド・アプリケーションの具体的な原因の調査および特定に時間と労力がかかることです。チューニングおよび修正が可能な CPU バウンドの原因となり得る問題を、大きく以下の 3 つのカテゴリに分けて考えます。これらは、連続的に相互に関連します。

- 次善のプラン - クエリ・オプティマイザが、特定のクエリについて考えられる最善のプランよりもはるかに高コストなプランを選択した。
- 次善のクエリ - サーバが効率的にクエリを実行するのが難しくなるような方法でクエリ自体が構成されている。
- 次善のクエリ・パターン - データベース・アプリケーションの一部である個々のクエリが独立した状態では良好だが、パターンとして発行されると、全体としてデータベース・サーバにとって次善のクエリ・パターンになる。

3.2.1 次善のクエリ・プラン

データ分散統計情報

オプティマイザによる最適化が困難なクエリの場合、次善のクエリ・プランが生成されます。可能なアクセス・プランの範囲が膨大であるため (特に重要なクエリの場合)、最適化プロセスはクエリ処理における複雑なフェーズです。

複雑さが中程度のクエリの場合、何万もの使用可能なクエリ・プランがある可能性があります。何十ものテーブルに関係する非常に複雑なクエリの場合、可能なプランが数十万以上ある場合があります。結果として、それらすべての可能性の中から、どのプランが許容可能なパフォーマンスを提供するかを、クエリ・オプティマイザが判断する必要があります。クエリ・オプティマイザは、クエリ・プランの各部分の推定コストに基づいてこの判断を行います。コストの推定は、SQL Anywhere がテーブルに格納されているデータから自動的に収集する統計情報に基づいて行われますが、特に変化の激しいデータベース・アプリケーションの場合 (大量のデータが削除され、それに代わる新しいデータが挿入されるような場合)、これらの統計情報が古い情報である可能性があります。これが、オプティマイザが最適なクエリ・プランを選択できない主

な原因の 1 つです。

最適化ゴール・オプション

オプティマイザが最適なプランを選択できないもう 1 つの理由としては、最適化ゴール・オプションが正しく設定されていないことが考えられます。ほとんどのクエリでは、all-rows 設定（バージョン 8.0.2 以降ではデフォルト）が意味を持ちます。この設定では、サーバがただちに最初のローを返すのではなく、可能な最短時間（推定に基づく）ですべてのローを返します。しかし場合によっては、アプリケーションが処理を開始できるように、クエリの最初のローまたは最初のいくつかのローをできるだけ早く取得することを重視し、結果セット全体のフェッチにかかる時間をほとんど考慮しない場合もあります。このオプションを設定したことがある場合、遅いと思われる各クエリのこの設定をもう一度確認すると、問題を特定できる可能性があります。

次善のプランの解析

次善のクエリ・プランを検出するには、高コストなプランを見つけ、取得されたグラフィカル・プランを調べます。統計情報付きのグラフィカル・プランが取得されている場合、プランの各部分のロー数および実行コストについてオプティマイザが予測する推定値と、フェッチされた実際のロー数およびそれらのローのフェッチにかかった実際の時間の両方が表示されます。推定値と実際値が大きく異なる場合、オプティマイザが推定値を作成する際に基にした統計情報が古い情報である可能性があります。統計情報付きのグラフィカル・クエリ・プランでは、推定値と実際値が大幅に異なる部分が強調表示されます。プロファイリング・アナライザでグラフィカル・クエリ・プランを開くと、赤いボックスを探すだけで何らかの大きな異常があるかどうかがわかります。クエリ・プランについて考察する場合は、ローの大部分を占めるテーブルを調べる必要があります。なぜなら、これらのテーブルについての推定が不正確である場合、選択されるプランやそのプランの実行時間などに大きく影響するためです。クエリ・プランのどの部分が最も高コストで負荷がかかる部分であるかを調べるには、太線の黒いボックスと演算子をつなぐ太線を探します。太線があるテーブルは、そのクエリにとって重要なテーブルであるため、テーブルの推定値が実際値に対して適正であるかどうかをよく確認する必要があります。高コストであることが判明したクエリについて、この解析を行う必要があります。

オプティマイザが演算子やクエリ全体の実行時間を正確に予測することはめったにありません。通常、オプティマイザは各プランの相対的なコストのみを考慮するため、実際の実行時間の 3 ～ 5 倍以内でコストを推定できれば十分です。オプティマイザは、最も低コストなプランを選ぶためにプランの比較を行うため、プランの絶対的なコストは考慮しません。しかし、比較的単純なクエリでプランの実際のコストと推定コストが 20 倍をはるかに超えるほど異なる場合や、比較的単純なクエリで推定ロー数が 10 倍近く異なる場合は、統計情報に根本的な問題がある可能性があります。複雑なクエリの場合、特にサブクエリや UDF を含むクエリの場合は、桁違いに異ならないかぎり、決定的に問題があるとは言えません。

不十分なクエリ・プランの分析についてより多くの情報を得るためには、『SQL Anywhere サーバ - SQL の使用法』の「実行プランの解釈」を参照してください。

3.2.2 次善のクエリ

オプティマイザが非常に正確な統計情報を使用して最善を尽くしていても、オプティマイザが効率の良いアクセス・プランを作成しにくくなるような方法でクエリが記述されている場合があります。不要なカラムを要求するクエリなどがその一例です。たとえば、顧客名とともに注文リストを取得するアプリケーションについ

て考えます。

```
SELECT *
FROM orders, customer
WHERE orders.o customer_id = customer_id;
```

このアプリケーションは、Orders テーブルのカラム（および顧客名フィールド）のみを必要とするにもかかわらず、両テーブルのすべてのカラムのジョインを計算するようサーバに要求しています。この場合、特に Customer テーブルの幅が非常に広いと、サーバはそれらのカラムをあちこちに移動する余分な作業を何度も行わなければならず、さらには、中間結果としてそれらの余分なカラムを格納するために余分なメモリを使用することになります。クライアントも、それらの結果をフェッチして保存するために、余分なスペースと CPU サイクルが必要になります。
この例では、アプリケーションがこれらの余分なカラムを実際に使用しないのであれば、以下のようなクエリの記述方法の方が適切であると言えます。

```
SELECT orders.*, customer_name FROM orders, customer WHERE orders.o customer_id = customer_id;
```

結果セット内の不要なカラムを見つけ出すには、高コストなクエリとそれらを呼び出すアプリケーション・コードの両方を調査する必要があるため時間がかかりますが、最も高コストなクエリの中からそのようなケースを見つければ、効果を上げることができます。アプリケーションでバインドされていない／使用されていないカラムを見つければ、オプティマイザがより効率的なアクセス・プランを選択できる可能性があります。しかし、この解析は時間がかかるため、アプリケーション・プロファイリング・トレースに現れる最も高コストなクエリでないと、この解析を行う意味がありません。

ユーザ定義関数

データベース・サーバが処理しにくいもう 1 つのクエリ・タイプは、UDF（user-defined function ：ユーザ定義関数）の呼び出しが多いクエリです。UDF は非常に強力で、多数のロジックをクエリに詰め込むことができますが、ネイティブ・サーバ・コードではなく解釈型の SQL（もしくはサーバでサポートされる別のストアドプロシージャ言語）で記述されているため、実行関連のパフォーマンスに悪影響を及ぼします。特に、結果セットに数千以上のローが含まれている場合、結果セットのローごとに UDF を呼び出すことは避けた方が賢明です。UDF は、有限かつ規定された回数だけ呼び出される場合に、最も効率的な使用が可能になります。これは絶対的なルールではなく、大規模な結果セットでも UDF を正常に使用できますが、アプリケーションで最も高コストなクエリに UDF が含まれている場合、その構成を使用しない方法でクエリを記述し直すことを検討する必要があります。クエリを記述し直す場合、追加ジョインを使用する必要がある可能性があります。

上記の例以外にも、最適なクエリが作成されない可能性は数多くあります。非常に遅く見えるクエリ・セットを見つけても原因がわからない場合は、クエリの実験が可能であるかどうかを検討してください。セマンティック上等しいクエリを記述し直すことで、複雑なクエリのパフォーマンスが著しく改善されることがあります。

3.2.3 次善のクエリ・パターン

前述のとおり、個々のクエリが独立した状態では非常に速いにもかかわらず、それらが何回も実行されるため、最も高コストな要求としてプロファイリング・モードの［Summary］ビューの上部に表示されることがあります。そのようなクエリは、アプリケーション内の次善のクエリ・パターンの一部である可能性があります。この問題は、わずかに異なるパラメータを渡しながら同じ基本クエリを何回も発行する場合に発生します。たとえば、以下のような文のシーケンスを発行するとします。

```
SELECT * FROM orders WHERE o_custname = 'Alice'; SELECT * FROM orders WHERE o_custname = 'Bob';
```

このパターンのクエリを単一のクエリとして発行した方がほぼ確実に効率的です。顧客リストがアプリケーションでのみ使用可能な場合、以下のような 1 つの文として記述できます。

```
SELECT * FROM orders WHERE o_custname IN ('Alice', 'Bob', ...);
```

顧客リストがデータベースに格納されていた場合、クエリをジョインとして記述できます。

```
SELECT orders.*
FROM orders, customer
WHERE order.o_custname = customer.name;
```

単一のクエリが 2 往復以上せず、また、個々の顧客に解析やカーソルの開き直しを行わないため、サーバが必要な結果をすばやく配信できます。ただし、この方法を使用するには、アプリケーション・コードを変更する必要があります。しかし、この方法を使用すれば、パフォーマンスが著しく向上する可能性があります。これは特に、クエリを発行しているアプリケーションの一部とサーバの間にネットワーク接続が存在する場合に当てはまります。

このタイプの次善のパターンの変形で、より簡単に修正できるのが、同じデータを繰り返しフェッチするクエリです。同じクエリを繰り返し何度も実行する代わりに、データをクライアントにキャッシュすることができ、なおかつその方が都合が良い場合は、同じデータを繰り返し実行したりフェッチする不要な作業をやめることで、アプリケーションの全体的な遅延時間および実行時間を減らすことができます。プロファイリング・モードの [Summary] ウィンドウ枠で、パフォーマンスの悪いクエリ・パターンの兆候を見つけることができます。

3.3 同時性バウンド・アプリケーション

アプリケーションが同時性バウンドである場合、実際はリソースバウンドでもあるのですが、問題なのは、データベース・サーバが使用できるリソースが他の考慮事項によって制限されていることです。つまり、サーバがそれ以上の CPU または ディスクを利用できないということです。アプリケーションが同時性バウンドである場合、I/O バウンドや CPU バウンドでもある場合が多いのですが、問題はスケーラビリティにあり、絶対的な有界性の問題ではありません。

同時性バウンド・アプリケーションの場合、サーバがクエリの処理を待機していてもそれを実行することはできません。サーバ内でアプリケーションが同時性バウンドであるかどうかを最も簡単に判断するには、ActiveReq パフォーマンス・カウンタを調べます。このカウンタは、以下を含む多くの場所で確認できます。

- perfmon で、SQL Anywhere 10 の Server オブジェクト用に表示されるカウンタを確認する。
- 手動でプロパティを照会する。

```
SELECT PROPERTY('ActiveReq');
```

または

- アプリケーション・プロファイリング・トレースで取得されたパフォーマンス・カウンタを調べる (「[Statistics] ウィンドウ枠」を参照)。

3.3.1 競合タイプの判断

アクティブ要求の数が多いにもかかわらず (少なくとも 10 ～ 20で、おそらくそれよりもはるかに多い)、サーバが大量の CPU または I/O リソースを使用していないように見える場合、それはアプリケーションが同時性バウンドである確実な兆候です。同時性バウンド・アプリケーションは、サーバに関

係している場合とアプリケーションに関係している場合があります。
　競合問題がサーバベースとアプリケーションベースのどちらであるかは、以下の方法で判断します。

・dbconsole を調べます。他の接続で接続がブロックされている（ローロック）と報告されている場合、その競合はアプリケーションに関係します。他の接続で接続がブロックされていると報告されていない場合、サーバ内の競合である可能性があります。
・Sybase Central でアプリケーション・プロファイリング・トレースを調べる際に（トレースが非常に大きい場合）、[Blocking Events] タブを見ます。ブロッキング・イベント数がデータベース要求数の重要な一部（20 ～ 30% 以上）である場合、アプリケーションベースの同時性ボトルネックが存在すると思われます。
この情報は、sa_diagnostic_blocking テーブルを照会することでも確認することができます。

```
SELECT COUNT(*) FROM sa_diagnostic_blocking WHERE logging_session id = 1;
```

これらの結果を比較します。

```
SELECT COUNT(*) FROM sa_diagnostic_request WHERE logging_session id = 1;
```

3.3.2 サーバベースの同時性制限

データ構造を保護し、サーバが使用するマシン・リソースへのアクセスを抑制するために、サーバが内部ロックを維持することにより、サーバ関連の同時性ボトルネックが発生します。たとえば、1 つのトランザクションのみがトランザクション・ログの特定の部分に書き込むことができるため、サーバはトランザクション・ログの各部の予約プロセスを制限する必要があります。SQL Anywhere の以前のバージョンでは、非常に並列的なマシンにおけるサーバ内の競合は、サーバ・パフォーマンス問題の大きな原因でした。たとえば、バージョン 8 の場合、8 ウェイ・マシンではパフォーマンスに大きな制約がありました。バージョン 9 および 10 では改善が行われ、サーバの競合ポイント数が減りました。そして、バージョン 10.0.1 のサーバでは、競合ポイントが発生する可能性が非常に少なくなりました。

バージョン 10.0.1 のサーバの場合、サーバ競合が最も発生しやすいポイントは以下のとおりです。

・トランザクション・ログ
・チェックポイント・ログ
・ロック・テーブル

これらのタイプのサーバ競合が発生するのは、通常、クライアントの同時接続数が 50 をはるかに上回る場合のみです。

トランザクション・ログ競合

多数の接続が非常に小さいトランザクションの挿入、更新、または削除を行っている場合、トランザクション・ログ競合が発生する可能性があります。たとえば、数百の接続があり、各接続が 1 つのローの挿入と送信を繰り返し行っている場合、トランザクション・ログ競合が発生する可能性があります。その場合、他のドライブが完全にアイドル状態であったとしても、トランザクション・ログが格納されているディスク・ドライブは、ディスク・アクティビティが多くなります。この問題が疑われる場合、最初に実行する解決策としては、トランザクション・ログをあらかじめ大きくしておきます。これを行うには、以下の文を発行します（環境に合ったサイズを指定しますが、大容量のアプリケーションの場合はおそらく 200MB 以上必要です）。

```
ALTER DBSPACE TRANSACTION LOG ADD 200MB;
```

さらに、この問題が疑われる場合は、I/O バウンド・サーバに関する説明で述べたように（第 3.1.4 項を参照）、トランザクション・ログを専用のディスクに移動することも非常に効果的です。

チェックポイント・ログ競合

アプリケーションが更新や削除を選択的に少しだけ行っている場合、チェックポイント・ログ競合が発生する可能性があります。サーバは、チェックポイント間に、変更するページの変更前のコピー（変更前のイメージ）を保存する必要があります。複数ページに広く散在する少量のデータを更新している接続がアプリケーションに多数ある場合、変更が加えられたページの数が急激に増加し、その結果、チェックポイント・ログに保存しなければならない変更前のイメージ数も急激に増加します。それにより、チェックポイント・ログで競合が発生する可能性があります。アプリケーションのデータ更新パターンがこれと一致し、この問題が疑われる場合、バージョン 10.0.1 以前では選択の余地が限られます。最善策としては、データをセカンダリ dbspace に移動し、システム dbspace（メイン .db ファイル）が高速なドライブに単独で存在するようにします。

ロック・テーブル競合

アプリケーションが上記のいずれのケースにも当てはまらず、サーバが単一の CPU（またはおそらく 2 つの CPU）について CPU バウンドであると思われる場合、ロック・テーブルが競合の原因である可能性があります。この場合も、バージョン 10.0.1 以前ではできることはほぼありません。ただし、この問題が発生しやすいのは、I/O によって制約されないデータベース・ロードが非常に多い場合のみであり、通常は、数千の接続をホストするサーバでデータベース・サイズがキャッシュ・メモリに対して非常に小さい場合のみです。

不十分な同時 I/O

同時性ボトルネックのもう 1 つの形態として、サーバのマルチプログラミング・レベル（-gn オプション）の設定が低すぎる場合があります。ここでは、データベース・ファイルが、スピンドル数の多い高速な RAID 0 のディスク上にあり、ランダム・データを非常に高速に取り出すことができるとします。この場合、サーバは、できるかぎりディスクを駆動しますが、より多くの同時接続を処理すれば、より一層ディスクを駆動することができます。このケースに当てはまるかどうかを判断するには、perfmon の [Current Disk Queue Length] を調べます。大きい RAID ディスクの場合、ディスクが完全にロードされていると、カウンタが 50 以上になります。このカウンタが 10 ～ 20 以下である場合、ディスクをより駆動することができるにもかかわらず、それに十分な速さでサーバが I/O 要求を発行していないということです。

この問題の解決策として、サーバのマルチプログラミング・レベルを増加します。試しに 30% ほど -gn オプション（デフォルトは 20）を増加し、データベース・スループットおよびディスク・キューの長さへの効果を観察します。同時 I/O が本来の問題であった場合、この変更後、ドライブでディスク・キューの長さが増加し、ディスクのアイドル時間が減少します。このタイプのボトルネックも、大規模な並列 I/O 機能を持つ高負荷のサーバでしか発生しません。マルチプログラミング設定は、あまり大幅に変更しないでください。第 3.1.3 項で説明されているようなクエリ実行に関する問題を引き起こす可能性があります。

3.3.3 アプリケーションベースの同時性制限

アプリケーションベースの同時性の問題の根本的な理由は、ある接続がローを必要としているときに、他の接続がそのローをロックしていることです。これは、アプリケーションが一貫性のある予測可能な動作をとるためには必要なことですが、プログラミングのしやすさとパフォーマンスの間の妥協点を見つける必要

があります。

ホット・ロー

この状況が発生する一般的なケースの 1 つが、ホット・ロー問題です。これは、多数の接続が 1 つのローを更新しようとする場合です（もしくは、独立性レベルによっては、ローを更新しようとしている接続は少数でも、多数の接続がそのローを読み取ろうとしている場合があります）。

新しいプライマリ・キーの値を保存するために 1 つのローが使用されている場合、ホット・ローがよく出現しますが、より効率的にこれを行う方法があります。
たとえば、グローバル・オートインクリメントを使用します。

長時間の読み込みロック

すべての接続から頻繁に検索される単一のロー（または小規模なローのセット）がなくても、個々の接続が非常に長時間ローに読み込みロックをかける場合があります。ロックの詳細については、『SQL Anywhere サーバ - SQL の使用法』の「ロックの仕組み」を参照してください。

問題は、トランザクションの間中、読み込みロックが頻繁にかけられることです。カーソルがローにあり、長時間開いたままである場合、独立性レベル 1 以上で読み込みロックが長時間かけられる場合があります。独立性レベル 1 以上で接続を実行している場合、カーソルを開いたままにするのであれば、そうする必要があるかどうかを確認する必要があります。独立性レベル 1 以上でカーソルが開かれていてカーソルがローにあるかぎり、他の接続はそのローを更新できません。独立性レベル 2 以上では、カーソルが閉じられるまでにアクセスしたすべてのローにロックがかけられるため、より一層制限的です。

独立性レベル 3 では、トランザクションでアクセスされたすべてのローに対してロックが取得され、トランザクションの終了までロックがかけられます。独立性レベル 3 でトランザクションを実行する必要がある場合、トランザクションを短くし、アクセスするロー数を最小限にするためにできることをすべて行ってください。独立性レベル 3 では、テーブル全体をスキャンするクエリを実行しないでください。クエリが特定のローを検索する必要がある場合は、処理を円滑にするためにインデックスを作成します。独立性レベル 3 における大規模または重要なテーブルの逐次スキャンは、それらのテーブルの更新も行う必要がある場合、同時性に非常に重大な影響を与えます。

カーソルが開いている時間の長さを調べるには、Sybase Central のプロファイリング・モードの [Details] タブを見るか、または sa_diagnostic_cursor テーブルでカーソルが開かれた時間と閉じられた時間を照会します。カーソルが使用する独立性レベルを調べるには、[SQL Statement Details] ウィンドウ ([Details] ビューでクエリを右クリックすると表示される) を見るか、または sa_diagnostic_cursor テーブルを照会します。アプリケーション・プロファイリングを使用していない場合、アプリケーションまたはデータベース・オプション設定を参照し、カーソルがどの独立性レベルを使用するように設定されているかを調べます。

長時間の書き込みロック

長時間かかる大規模な書き込みトランザクションがアプリケーションに含まれている場合があります。そのようなトランザクションは、トランザクションの間中、基礎となるすべてのローにロックをかけます。長時間のトランザクションを実行する必要がある場合は、トランザクションによって開かれているローが、独立性レベル 1 以上で他の接続によって更新または読み込みのために検索されないようにするための対策を講じてください。

複数の書き込みトランザクションが同じローを更新する場合、同時性の問題が発生する可能性があります。この問題は、書き込みトランザクションと独立性レベルが高い読み込みトランザクションが入り混じっ

て同じローを求めて競争する場合に、最も重大な問題になります。

スナップショット・アイソレーション

スナップショット・アイソレーションは、従来の独立性レベルに代わって、同時に実行されるトランザクションに一貫性のある予測可能なセマンティックを提供します。スナップショット・アイソレーション・レベルの 1 つ（実際には、いくつかのスナップショット・レベルがあり、動作にわずかな違いがある）を実行するトランザクションは、データベース全体をスナップショット・トランザクションの開始時の状態で参照します。そのために、トランザクションの開始後に変更されるすべてのローのコピー（変更前のイメージ）を保持します。トランザクション中に更新されないローについては、特別な作業は必要ありません。したがって、アプリケーションの観点からすると、これは、独立性レベル 3 でトランザクションを開き、トランザクションが使用するすべてのローにアクセスし、その後、適切なトランザクションを開始するようなものです。

スナップショット・アイソレーション・レベルは、トランザクションに非常に限定的なセマンティックを与え、主に読み込みが書き込みをブロックしないという理由で、読み込みと書き込みのトランザクションが混在する場合に高い同時性レベルをサポートできる可能性があります。適切なスナップショット・レベルを選択することで、書き込みの競合検出を遅らせることもできます。

ただし、この方法を使用することによる悪影響もあります。長時間のトランザクションは、他のトランザクションの実行の妨げにはなりませんが、他のすべてのトランザクションは、アクセスするローのバックアップ・コピーを作成する必要があります。トランザクションが多数のローを更新するアプリケーションの場合（それらのローがトランザクション間で完全に切断されるとしても）、それらのコピーを作成するために、大量の追加作業が必要になります。同時にアクティブになるスナップショット・トランザクションがごくわずかである場合も、このオーバヘッドは発生します。

スナップショット・アイソレーションを使用すると、アプリケーションが主に読み込み専用である場合（つまり、挿入／更新／削除が行われるローの総数が、読み込まれるローの総数よりもはるかに少ない場合）、一貫性のあるデータ・ビューの予測可能なセマンティックを必要とするアプリケーションのスループットが改善されます。書き込みが多いための作業負荷による同時性への悪影響はありませんが、CPU コスト（および、場合によってはディスク I/O コスト）が余計にかかります。

詳細については、『SQL Anywhere サーバ - SQL の使用法』の「スナップショット・アイソレーション」を参照してください。

デッドロックのパフォーマンスへの影響

同時性制限を持つアプリケーションでは、デッドロックもよく発生します。デッドロックが発生すると、デッドロックが発生したトランザクションを中止するかまたは後で実行し直すためのエラー処理コードを記述する必要があるため手がかかります。

デッドロックは、パフォーマンス問題の原因になる場合もあります。トランザクションで多数の作業が完了していて、そのトランザクションがデッドロック・サイクルに入っている場合、すべての作業をロールバックし、後で適用し直す必要があります。

アプリケーション設計

アプリケーション・トレースの解析によってアプリケーションのボトルネックが判明した場合、サーバの能力が制限されてしまいます。この場合、サーバは、単一の接続によって使用される CPU またはディスクについてリソースバウンドであると思われますが、CPU やディスクを追加しても効果が得られません。このような状況ではほとんどの場合、アプリケーションを設計し直さなければなりません。アプリケーション

設計の最初の段階から同時性やスケーラビリティを考慮することが重要なのはそのためです。高いロック競合がよく発生するデータベース・アプリケーションは、小規模なテスト・インスタンスではたいてい順調に動作しますが、作業負荷が大きくなると動作しなくなります。

3.4 クライアント層のパフォーマンス問題

アプリケーションのパフォーマンス問題が、これまで検討してきたいずれの状況にもあてはまらない場合、アプリケーションのクライアント部分に根本的な問題があり、データベース・サーバとはまったく関係のない可能性があります。これについて確認するには、perfmon または保存されているアプリケーション・プロファイル内の ActiveReq カウンタと、サーバでのリソース消費量を調べます。CPU およびディスク I/O の値が低く、ActiveReq カウンタの値も低い場合（おそらく 20 以下）、サーバがビジー状態を保つことができるほどの作業を受信していません。修正方法としては、従来のソースコード・プロファイラなどを使用してアプリケーション・コードを調査し、クエリが十分な速さでサーバに発行されない理由を突き止めます。この問題はおそらく、クライアント層が重要な処理を行っている多層アプリケーションで最も発生しやすくなります。つまり、その部分でパフォーマンスの改善が見られるということです。

4 まとめ

データベース・アプリケーションのパフォーマンス問題の調査は時間がかかる可能性があります。しかし、このマニュアルで説明されているツールや方法を使用すれば、比較的簡単にパフォーマンス問題の原因を特定することができます。最初にアプリケーション・パフォーマンスの包括的な特徴付けを行い、次に、SQL Anywhere に組み込まれているプロファイリング・ツールを使用して、直面しているパフォーマンス問題の種類を特定することのメリットについて説明しました。バージョン 10 以降に組み込まれているアプリケーション・プロファイリング・ツールは、この作業に役立つ、これまでで最も包括的かつ統合的な機能セットを提供しますが、それ以前のバージョンに付属しているツールも使用できます（第 2.3.3 項を参照）。

しかし、多くの場合、パフォーマンス問題の原因を特定するのに必要な時間と労力と、それを解決するために必要な時間と労力との間に相関関係はほとんどありません。単一の構成オプションを変更したり、ハードウェアを追加するだけで解決できる問題もあれば、より大きな作業負荷でも許容可能なパフォーマンスが得られるようにするために、データベース・アプリケーションを大幅に変更しなければならないような問題（特に、第 3.3 項で説明されているような問題）もあります。

第 3 項では、データベース・アプリケーションのパフォーマンス問題の大まかなカテゴリを示し、それらのカテゴリに属する最も一般的な個々の問題について説明しました。さらに、それぞれの問題について、最も役立つと思われる解決方法も提案しました。このマニュアルで推奨されている解決方法は、SQL Anywhere の開発チームの経験に基づいて提案されたものであり、最も幅広い適用可能性を持つ方法です。しかし、大規模なデータベース・アプリケーションはソフトウェアの複雑な部分であるため、推奨される限られた解決方法で、発生し得るすべてのパフォーマンス問題を十分に解決できるわけではありません。この方法を繰り返し実行することで、直面しているパフォーマンス問題をより正確に特徴付けることができれば、外部の情報源（SQL Anywhere のニュースグループなど）から得た解決方法を、状況に合わせて正確に調整して使用することができます。

法的注意