# SQL Serverのパフォーマンスを最適化するための10のヒント

作成: Patrick O'Keefe、Richard Douglas

## 要旨

SQL Serverのパフォーマンスの最適化は、困難な作業になることがあります。一般に、パフォーマンスの問題に対応する方法に関しては、さまざまな情報を入手できます。しかし、特定の問題についてはあまり詳細な情報がなく、特定の知識を各自の環境に適用する方法についてはほとんど情報がありません。

このホワイトペーパーでは、SQL Serverのパフォーマンスを最適化するための10のヒントを、SQL Server 2008/2012を念頭に置いて紹介します。最も重要なヒントの完全なリストはありませんが、このホワイトペーパーを開始点とすることで、誤った方向に進まないようにすることができます。

DBAである皆さんはきっと、独自の観点を持ち、最適化に関するお気に入りのヒント、コツ、シナリオを利用していることと思います。ぜひ、SQLServerPediaの議論に参加してください。

## 概要

**チューニングの目標の検討**

誰もがSQL Server環境を最大限に活用したいと考えています。データベースサーバーの効率性を高めることで、ビジネスレポートやアドホッククエリといった他のタスクのためにシステムリソースを解放することができます。組織のハードウェアへの投資を最大限に活用するには、データベースサーバー上で実行するSQLワークロードやアプリケーションワークロードをできる限り迅速かつ効率的に実行する必要があります。

しかし、チューニングの方法は各自の目標によって異なります。「SQL Server環境の効率性は最高になっているだろうか?」、「アプリケーションを拡張できるだろうか?」といった疑問を持っている人もいると思います。ここでは、システムのチューニングに関する主な方法を紹介します。

- サービスレベルアグリーメント(SLA)や主要なパフォーマンス指標(KPI)の目標を達成するためのチューニング
- 効率性を高め、それによって他の目的のためにリソースを解放するためのチューニング
- 拡張性を維持し、その結果、将来にわたってSLAやKPIを維持するためのチューニング

ビジネス要件を現在満たしているとしても、すべてのデータベースサーバーの拡張性と効率性を最大限に高めることに取り組んでください。

**チューニングは継続的なプロセスであり、一時的な処置ではない。**

パフォーマンスの最適化は継続的なプロセスです。例えば、SLAの目標を達成するためのチューニングには「終わり」があります。しかし、効率性の向上や拡張性の維持を目的としたチューニングでは、作業が本当に完了することは決してありません。この種のチューニングは、パフォーマンスが「十分」になるまで継続されます。その後、アプリケーションのパフォーマンスが十分でなくなったときには、チューニングサイクルを再び実行する必要があります。

通常は、ビジネス原則（SLAやシステムスループットの要件など）で「十分」の定義を行います。すべてのデータベースサーバーの拡張性と効率性を最大限に高めるには、要件の定義に加え、動機付けも必要です。これは、ビジネス要件を現在満たしているとしても必要です。

**本書について**

SQL Serverのパフォーマンスの最適化は難しい作業です。各種のデータポイント（パフォーマンスカウンターや動的管理オブジェクト（DMO）など）に関する一般的な情報は豊富にありますが、そのデータの扱い方や解釈の仕方に関する情報はほとんどありません。このホワイトペーパーでは、さまざまな場面で役に立つ、10個の重要なヒントを紹介します。これらのヒントを活用することで、各種のデータを実用的な情報に変えることができます。

## ヒント10: 基準設定とベンチマーキングを使用した手法を通じて問題を特定する。

**基準設定とベンチマーキングのプロセスの概要**

図1に、基準設定プロセスの手順を示します。次のセクションで、このプロセスにおける重要なステップについて説明します。

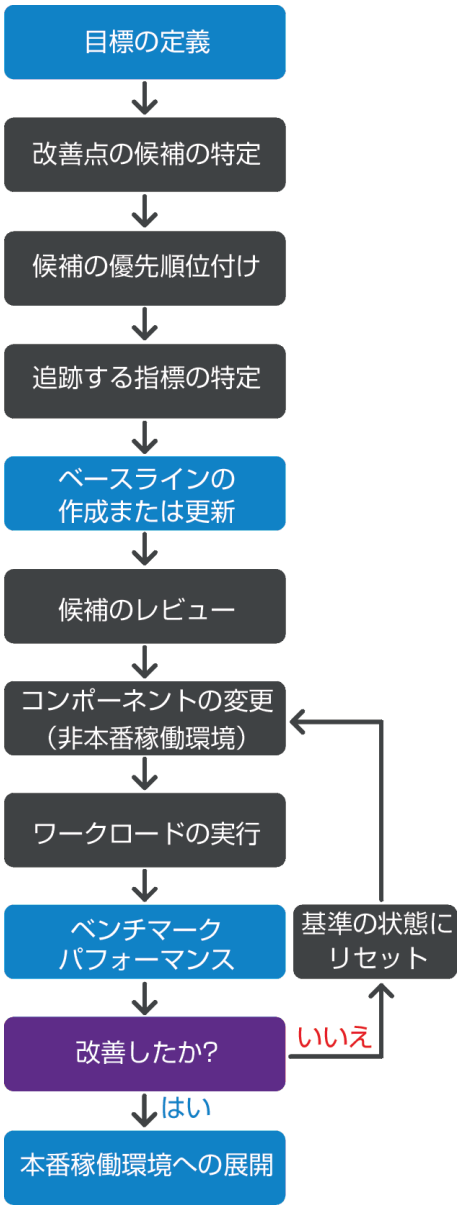


*図1.基準設定プロセス*

**変更を開始する前に、現在のシステムパフォーマンスの記録を入手する。**
チューニングや最適化を行うとき、すぐに作業に取り掛かろうとしてしまう人がほとんどではないでしょうか。車が修理から帰ってきたとき、前よりも調子が悪くなったのではないかと感じたことはありませんか。そんなとき、文句を言いたくなるものの、本当に調子が悪いのか確信を持てないものです。また、現在起こっていると思われる問題の原因が、整備士が行ったことにあるのか、それとも修理が終わった後に起きたことにあるのか、迷うこともあるでしょう。残念ながら、データベースのパフォーマンスに関しても同じ問題が起こり得ます。

しかし、このホワイトペーパーを読むことで、さまざまなアイデアを得ることができ、基準設定とベンチマーキングの戦略をすぐに実践したくなるはずです。最初に行う必要があるステップはそれほど面白いものではありませんが、最も重要なステップの1つであることは間違いありません。このステップでは、既存の環境が変更する予定の基準にどの程度達しているかを確認します。

**目標を決定する。**
システムに対して何らかの処置を施す前に、達成したいことを決定します。SLAの要素の中で、パフォーマンス、リソース消費量、容量に関して対応する必要があるものはありますか?本番稼働環境における現在の問題に対応していますか?リソースの時間に関する不満が挙がっていますか?設定する目標は明確にしてください。

私たちのほとんどは、多くのデータベースやインスタンスを管理しています。作業の効果を最大限に高め、特定のシステムが優れたパフォーマンスを発揮し、ユーザーの期待に応えるためには、そのシステムの要件を慎重に検討する必要があります。分析やチューニングを過剰に行うと、優先度の低いシステムに必要以上の時間を費やし、主要な本番システムに悪影響を与えることもあります。測定やチューニングの作業によって達成したいことを明確にしてください。また、それらの作業に優先順位を付けてください。ビジネススポンサーからの賛同が得ることができれば理想的です。

**基準を設定する。**
達成したいことに関して目標を定めたら、次に、成功の基準を決定する必要があります。必要な洞察を与えるOSカウンター、SQL Serverカウンター、リソースの指標、およびその他のデータポイントには何がありますか?

そのリストを作成したら、基準(選んだ条件に対するシステムの標準的なパフォーマンス)を設定する必要があります。また、システムの標準的なパフォーマンスの標本を得るのに十分なデータを十分な期間をかけて収集する必要があります。データを収集すると、その期間における平均値を求めることによって、最初の基準を設定できるようになります。システムの変更を行った後に新しいベンチマークを取得して元のベンチマークと比較することにより、変更の効果を客観的に評価することができます。

**平均値だけでなく、基準からの偏差も追跡する。**
平均値に注意することも重要ですが、少なくとも、各カウンターの標準偏差を算出し、経時的な変化を認識する必要があります。例えば、ロッククライマーのことを考えてみましょう。彼には、ロープの平均直径が1 cmであると伝えます。彼は自信を持って、斜面に挑みます。険しい岸壁を数百フィート登ったところで、得意げな笑みを浮かべます。そこで、ロープの最も太い部分は2 cmで、最も細い部分は0.1 cmだと伝えます。彼はどうなるでしょうか。

標準偏差について詳しくない人は、統計学の入門書を読んでください。それほど高度な知識を身に付ける必要はありません。基本を知るには入門書で十分です。

重要なことは、平均だけを追跡するのではなく、標準(平均)からの偏差も追跡する必要があるということです。そのためにはまず、標準の内容を決定します(通常はSLAで定義します)。行うべきことは、最高のパフォーマンスを得ることではなく、パフォーマンスの目標を達成するのに最適な状態を実現し、その目標からの偏差をできるだけ抑えることです。目標からの逸脱は、時間や労力が無駄になっていることを表しており、

ベンチマーキングを行うことで、通常の動作がどのようなものかを示す優れた指標が得られるため、異常な動作を特定しやすくなります。

行うべきことは、最高のパフォーマンスを得ることではなく、パフォーマンスの目標を達成するのに最適な状態を実現し、その目標からの偏差をできるだけ抑えることです。

インフラストラクチャリソースの使用率が低下している可能性もあります。

**基準設定に必要なデータの量**

基準設定に必要なデータの量は、負荷の経時変化の程度によって異なります。システム管理者、エンドユーザー、およびアプリケーション管理者に話を聞いてみてください。彼らは概して、使用パターンに対する優れた感覚を持っています。オフピーク期間、平均的な期間、およびピーク期間のすべてにわたって十分なデータを収集する必要があります。

負荷の変化量と変化頻度を測定することが重要です。パターンを予測できるシステムでは、データの収集量を減らしても構いません。変化が大きいほど、測定間隔を短くし、測定期間を長くして、信頼できる基準を作成する必要があります。先ほどのロッククライマーの例えでもう少し細かい話をすると、調べるロープが長ければ長いほど、変化が見つかる可能性が高くなります。また、システムの重要度やその故障による影響も、実施する調査の量やサンプルに求められる信頼度に影響します。

**データの保存**

追跡するパラメーターが多くなり、追跡の頻度が高くなるほど、収集するデータセットは大きくなります。これは明らかなことのように思えるかもしれませんが、測定データを保存するには、必要な容量を実際に検討する必要があります。データをいくらか保存すれば、リポジトリが時間と共にどの程度成長するかを推定することは極めて簡単なはずです。長期間にわたって監視を行う予定の場合は、履歴データを定期的に集約して、リポジトリが大きくなりすぎないようにすることを検討してください。

また、パフォーマンス上の理由から、測定するリポジトリを監視対象のデータベースと同じ場所に配置しないでください。

**同時に行う変更の数を制限する。**

各ベンチマーク間に行う変更の数を制限してください。また、変更を行って、特定の仮定をテストしてください。こうすることで、各改善点候補の取捨選択を入念に行うことができます。1つのソリューションに焦点を合わせると、動作の変化を監視する理由が正確に分かるだけでなく、一連の潜在的な改善の選択肢がさらに得られることも多々あります。

**データの分析**

システムに対する変更を行った後、望んでいた効果が得られたかどうかを確認したくなると思います。そのためには、元の基準に対して実施した測定を、同様の標本期間にわたって繰り返し行います。これにより、2つの基準を比較して、次のことを実行できます。

- **変更によって望んでいた効果が得られたかどうかを確認する**: 構成の設定の調整、インデックスの最適化、またはSQLコードの何らかの変更を行う場合は、基準を用いることで、変更によって望んでいた効果が得られたかどうかを確認できます。パフォーマンスの低下に関する苦情が寄せられた場合は、データベースの観点から、その内容が正しいかどうかを判断することができます。

経験の浅いほとんどのDBAが最も頻繁に犯す過ちは、結論を早急に出すことです。変更を行った人が、目に見えるパフォーマンスの向上が即座に確認されたことで、小躍りするほど喜んでいる姿を見たことがありませんか。そして、本番稼働環境に変更を展開し、問題が解決したことを知らせるEメールをすぐに送ってしまうのです。しかし、しばらくして同じ問題が再び顕在化したり、未知の副作用が原因で別の問題が発生したりすれば、その喜びの時間はあっという間に終わります。変更によって以前よりも状況が悪化することも少なくありません。問題の解決策が見つかったと思ったら、テストを実施し、基準に照らして結果のベンチマークを行います。これが、変更が成功したことを確認するための唯一の信頼できる方法です。

- **変更によって想定外の副作用が生じたかどうかを確認する**: 基準を用いることで、変更がカウンターに影響したかどうかを客観的に確認したり、変更による予想外の影響を測定したりすることもできます。
- **問題が発生する前に予測する**: 基準を用いることで、通常の負荷状況に対する正確なパフォーマンスの状態を定めることができます。これにより、最近のリソース消費量の傾向に基づいて、あるいは将来のシナリオに対して予想されるワークロードに基づいて、今後問題が発生するかどうか、およびいつ問題が発生するかを予測することができます。例えば、キャパシティプランニングを行う場合、接続ユーザー1人あたりの現在の標準的なリソース消費量から推定することにより、システムにユーザー接続のボトルネックがいつ発生するかを予測できます。
- **より効果的に問題のトラブルシューティングを行う**: データベースのパフォーマンスに関する問題を解決するために何昼夜も費やした結果、その問題がデータベース自体とは実際には関係がないことが分かったことがありませんか?基準を定めることで、はるかに簡単にデータベースインスタンスを調査対象から外し、問題の原因に的を絞ることができます。例えば、メモリ消費量が突然急増したとします。あなたは、接続数が急増し、基準を大きく超えていることに気付きます。すぐにアプリケーション管理者に電話をしたところ、eStoreに新しいモジュールが導入されていたことを確認できました。新任の若い開発者が作成したコードで、データベース接続を切断すべきところで切断されていないことがほどなく分かりました。このようなシナリオが他にも多数思い浮かぶのではないでしょうか。

問題の原因でないことを排除し、余計なものを取り除き、問題の正確な原因に焦点を当てることで、時間を大幅に節約することができます。システムカウンターとSQL Serverカウンターを比較することで、問題の原因がデータベースにあるかどうかを素早く判断できる例は数多くあります。よくある原因を排除した後で、基準からの大幅な逸脱があるか調査し、関連する指標をまとめて収集して、根本原因を詳細に調べ始めることができます。

**基準設定プロセスを必要に応じて繰り返す。**
優れたチューニングは、反復的かつ科学的なプロセスです。このホワイトペーパーで紹介するヒントは、極めて優れた開始点になりますが、逆に見れば開始点に過ぎないとも言えます。パフォーマンスチューニングは、個々のシステムの設計、構成、および使用状況に応じて、高度にカスタマイズされ、管理されます。

基準設定とベンチマーキングを使用した手法は、高度で信頼性に優れたパフォーマンスチューニングの基礎になります。この手法は、目標とその達成方法を明確化するのに必要な、計画、基準、および中間目標を与え、途中で挫折しないようサポートしてくれます。この体系化された手法を利用することで、デルのデータベースポートフォリオのすべてにおいて、信頼性と一貫性に優れたパフォーマンスを実現することができます。

基準設定に必要なデータの量は、負荷の経時変化の程度によって異なります。

## ヒント9: パフォーマンスカウンターは、現在の運用に関する手軽で有用な情報になる。

### パフォーマンスカウンターを監視する理由

SQL Serverのパフォーマンスの最適化に関連するよくある質問は、「どのカウンターを監視すべきか?」というものです。SQL Serverの管理に関しては、パフォーマンスカウンターを監視する大きな理由として以下の2つがあります。

- 運用効率の向上
- ボトルネックの防止

この2つの理由はいくらか重複しますが、これらを考慮することで、監視する多数のデータポイントを容易に選択することができます。

### パフォーマンスカウンターの監視による運用効率の向上

運用の監視では、全体的なリソースの使用量をチェックします。これにより、以下のような疑問に対する回答が得られます。

- サーバーにおいてCPU、ディスク領域、メモリなどのリソースが不足しそうか?
- データファイルを増やすことができるか?
- 固定サイズのデータファイルにデータを追加するための十分な空き容量があるか?

各ベンチマーク間で行う変更の数を制限することで、各変更の効果を入念に評価することができます。

傾向を調べる目的でデータを使用することも可能です。例えば、すべてのデータファイルのサイズを収集することで、データファイルの成長率の傾向をつかみ、将来のリソース要件を予測することができます。

上の3つの疑問に対する回答を得るには、以下のカウンターを監視する必要があります。

| カウンター | 実行できること |
|---|---|
| プロセッサー\プロセッサー時間(%) | サーバーでのCPUの消費量を監視する |
| 論理ディスク\空き容量(MB) | ディスク上の空き容量を監視する |
| MSSQL$インスタンス:データベース\データ・ファイル・サイズ(KB) | 経時的な増加傾向を分析する |
| メモリ\ページ/秒 | ページングを確認する(メモリリソースが不足している可能性があることを示す優れた指標になる) |
| メモリ\利用可能サイズ(MB) | システムが使用できる物理メモリの量を監視する |

パフォーマンスカウンターの監視によるボトルネックの防止

ボトルネックの監視では、パフォーマンス関連の要素により重点を置きます。収集したデータは、以下の疑問に対する回答を得るのに役立ちます。

- CPUのボトルネックは存在するか?
- I/Oのボトルネックは存在するか?
- バッファキャッシュやプランキャッシュなど、SQL Serverの主要なサブシステムが正常に稼働しているか?
- データベース内に競合が存在するか?

これらの疑問に対する回答を得るには、以下のカウンターを監視します。

| カウンター | 実行できること |
|---|---|
| プロセッサー\プロセッサー時間(%) | CPUの消費量を監視することで、サーバー上のボトルネックをチェックすることができます(消費量が高い状態が維持されることによって示されます)。 |
| 高い割合のシグナル待機 | シグナル待機とは、CPUでロック、ラッチなどの待機などが終了した後に、作業者がCPU時間の待機に費やす時間です。CPUに対する待機に時間が費やされる場合は、CPUのボトルネックが存在します。シグナル待機は、SQL Server 2000ではDBCC SQLPERF(waitstats)を実行することによって、SQL Server 2005ではsys.dm_os_wait_statsに対するクエリを実行することによって見付けることができます。 |
| 物理ディスク\平均ディスクキューの長さ | ディスクのボトルネックをチェックします。2を超える値の場合は、ディスクのボトルネックが存在する可能性があります。 |
| MSSQL$インスタンス:バッファマネージャ\ページの予測保持期間 | ページの予測保持期間は、ページがバッファキャッシュ内に留まる秒数です。小さい値は、ページがキャッシュに格納されてからそれほど時間が経たないうちに削除されることを示し、キャッシュの効果を低減します。 |
| MSSQL$インスタンス:プランキャッシュ\キャッシュヒット率 | 低いプランキャッシュのヒット率は、プランが再利用されていないことを表します。 |
| MSSQL$インスタンス:一般統計\プロセスブロック | 長いブロックは、リソースの競合が存在することを示します。 |

## ヒント8: サーバーの設定を変更することで、より安定した環境を実現できる。

製品内の設定を変更することによって安定性を高めることは、直感と相いれないことのように聞こえるかもしれませんが、これは実際に効果があります。DBAとしての任務は、ユーザーがアプリケーションに対してデータを要求する際のパフォーマンスレベルを一貫して保つことです。このホワイトペーパーの残りの部分で説明する設定の変更を行わないと、何の前触れもなくパフォーマンスの低下を招く可能性があるシナリオが現実のものになるかもしれません。以下に挙げるオプションは、サーバーレベルの設定が列記されているsys.configurations内にあり、他の情報と組み合わせて使用することができます。sys.configurationsのIs_Dynamic属性は、設定の変更後にSQL Serverインスタンスを再起動する必要があるかどうかを示します。変更を行うには、関連するパラメーターを指定してsp_configureストアドプロシージャを呼び出します。

**メモリの最小/最大設定により、一定レベルのパフォーマンスを保証できる。**

アクティブ/アクティブ構成のクラスタがあるとします（実際は複数のインスタンスを持つ単一ホスト）。同じ物理ホスト上に両方のインスタンスが存在する場合、フェールオーバーが発生してもSLAの達成を保証可能な、特定の設定変更を行うことができます。

このシナリオでは、メモリの最小/最大設定の両方に対して変更を行うことにより、物理ホストが各インスタンスに対処するのに十分なメモリを確保できるようにします。常に他方のインスタンスのワーキングセットを積極的に減らそうとする必要はありません。一定レベルのパフォーマンスを保証するために、特定のプロセッサーを使用するように同様の設定変更を行うことができます。

最大メモリを設定することが、クラスタ上のインスタンスだけでなく、他の任意のアプリケーションとリソースを共有するインスタンスにとっても都合が良いことに気付くことが重要です。SQL Serverのメモリ使用量が多すぎる場合、SQL Server自体や他のアプリケーションが稼働できる余地を残すために、オペレーティングシステムが使用可能なメモリの量を積極的に減らすことがあります。

- **SQL Server 2008**: SQL Server 2008 R2以前では、メモリの最小/最大設定を行っても、バッファプールが使用するメモリの量しか制限されません（具体的には、1ページ（8 KB）の割り当てのみ）。つまり、バッファプールの外部でプロセス（拡張ストアドプロシージャ、CLR、またはIntegration Services、Reporting Services、Analysis Servicesといったその他のコンポーネントなど）を実行する場合は、この値をさらに小さくする必要がありました。
- **SQL Server 2012**: SQL Server 2012では、中央メモリマネージャがあるため、若干状況が変わります。このメモリマネージャには、現在、8 KBを超える大規模なデータページやキャッシュプランなどのマルチページ割り当てが組み込まれています。このメモリ領域には一部のCLR機能も含まれます。

**パフォーマンスの向上に間接的に役立つ2つのサーバーオプション**

パフォーマンスの向上に直接役立つオプションはありませんが、間接的に役立つオプションなら2つあります。

- **バックアップ圧縮の既定**: このオプションでは、デフォルトで圧縮するバックアップを設定します。このオプションを設定すると、圧縮時に余分なCPUサイクルが発生することがありますが、一般には、非圧縮方式のバックアップに比べて、使用されるCPUサイクルが全体的に少なくなります。これは、ディスクに書き込まれるデータが減るためです。I/Oのアーキテクチャにもよりますが、このオプションを設定することで、I/Oの競合が減る可能性もあります。
- もう1つのオプションについては、トップ10リストを引き続き読み進めてみてください（後述するプランキャッシュのヒントに着目してください）。

トラブルシューティングに関する賢明なアドバイス:「原因の排除と追及」

パフォーマンスカウンターを監視することで、運用効率を向上させ、ボトルネックを防止することができます。

## ヒント7: プランキャッシュ内の無効なクエリを検出する。

ボトルネックを特定したら、ボトルネックの原因となっているワークロードを見付ける必要があります。SQL Server 2005で動的管理オブジェクト(DMO)が導入されて以来、この作業はかなり容易になりました。SQL Server 2000以前のユーザーがこの作業を行うには、プロファイラーまたはトレースを使用するしかありません(ヒント6を参照)。

### CPUのボトルネックの診断

SQL ServerでCPUのボトルネックが特定された場合にまず行うことは、サーバーのCPUを最も消費しているユーザーを特定することです。これは、sys.dm_exec_ query_statsで極めて簡単に照会できます。

```
SELECT TOP 50
    qs.total_worker_time / execution_count AS avg_worker_time,
    substring (st.text, (qs.statement_start_offset / 2) + 1,
       ( ( CASE qs.statement_end_offset WHEN -1
            THEN datalength (st.text)
            ELSE qs.statement_end_offset END
         - qs.statement_start_offset)/ 2)+ 1)
      AS statement_text,
   *
FROM sys.dm_exec_query_stats AS qs
   CROSS APPLY sys.dm_exec_sql_text (qs.sql_handle) AS st
ORDER BY
avg_worker_time DESC
```

このクエリの本当に有用な部分は、cross applyとsys.dm_exec_sql_textを使用してSQLステートメントを取得できることにあります。このため、SQLステートメントの分析が可能になります。

### I/Oのボトルネックの診断

I/Oのボトルネックの診断とほぼ同じです。

```
SELECT TOP 50
      (total_logical_reads + total_logical_writes) AS total_logical_io,
      (total_logical_reads / execution_count) AS avg_logical_reads,
(total_logical_writes / execution_count) AS avg_logical_writes,
(total_physical_reads / execution_count) AS avg_phys_reads,
       substring (st.text,
            (qs.statement_start_offset / 2) + 1,
      ((CASE qs.statement_end_offset WHEN -1
               THEN datalength (st.text)
       ELSE qs.statement_end_offset END
               - qs.statement_start_offset)/ 2)+ 1)
             AS statement_text,
   *
FROM sys.dm_exec_query_stats AS qs
    CROSS APPLY sys.dm_exec_sql_text (qs.sql_handle) AS st
ORDER BY total_logical_io DESC
```

## ヒント6: SQL Server Profilerは味方になる。

### SQL Server Profilerと拡張イベントについて

SQL Server Profilerは、SQL Serverに付属しているネイティブツールです。SQL Server Profilerを使用すると、SQL Serverで発生するイベントをキャプチャするトレースファイルを作成できます。このトレースから、ワークロードや低パフォーマンスのクエリに関する情報を提供することの重要性が分かります。このホワイトペーパーでは、SQL Server Profilerの使用方法については詳しく説明しません。詳細な使用方法については、SQLServerPediaのビデオチュートリアルをご覧ください。

SQL Server 2012では、SQL Server Profilerよりも拡張イベントの方が優先されていることは確かですが、これは、データベースエンジンだけを対象としたもので、SQL Server Analysis Servicesではそうでないことに注意する必要があります。SQL Server Profilerは引き続き、多くのSQL Server環境でアプリケーションの機能に関する優れた洞察をリアルタイムで提供します。

拡張イベントの使用方法については、このホワイトペーパーでは説明しません。拡張イベントの詳細については、ホワイトペーパー『How to Use SQL Servers Extended Events and Notifications to Proactively Resolve Performance Issues(SQL Serverの拡張イベントと通知を利用して、パフォーマンスの問題をプロアクティブに解決する方法)』をご覧ください。拡張イベントは、SQL Server 2008で導入され、イベント数を増やし、待望のUIを搭載するために、SQL Server 2012でアップデートされました。拡張イベントの説明はこれだけにしておきます。

なお、SQL Server Profilerを実行するには、ALTER TRACE権限が必要です。

### SQL Server Profilerの使用方法

ここでは、パフォーマンスモニタ(Perfmon)でデータを収集し、リソース使用量の情報をSQL Server内で発生するイベントのデータと関連付けるプロセスを構築する方法について説明します。

1. Perfmonを起動します。
2. データコレクターセットをまだ構成していない場合は、ヒント9を参考に、詳細オプションとカウンターを使用して、ここで作成してください。データコレクターセットはまだ起動しないでください。
3. SQL Server Profilerを起動します。
4. 監視するインスタンス、イベント、列に関する詳細情報、およびデスティネーションを指定して、新しいトレースを作成します。
5. トレースを開始します。
6. Perfmonに切り替えて、データコレクターセットを開始します。
7. 必要なデータがキャプチャされるまで、両方のセッションを実行したままにします。
8. SQL Server Profilerのトレースを停止します。トレースを保存して閉じます。
9. Perfmonに切り替えて、データコレクターセットを終了します。
10. SQL Server Profilerで、最近保存したトレースファイルを開きます。
11. 「ファイル」、「パフォーマンスデータのインポート」の順にクリックします。
12. データコレクションのデータファイルに移動して、興味のあるパフォーマンスカウンターを選択します。

これで、パフォーマンスカウンターとSQL Server Profilerのトレースファイルを同時に参照できるようになります(図2を参照)。こうすることで、極めて迅速にボトルネックを解決することができます。

追加のヒント: 上の手順では、クライアントのインターフェイスを使用します。リソースを節約するために、サーバー側でトレースを行うとより効率的です。サーバー側のトレースの開始/終了方法については、オンラインブックを参照してください。

運用の監視では、全体的なリソースの使用量をチェックします。ボトルネックの監視では、パフォーマンス関連の要素により重点を置きます。

sys.configurationsのIs_Dynamic属性は、設定の変更後にSQL Serverインスタンスを再起動する必要があるかどうかを示します。

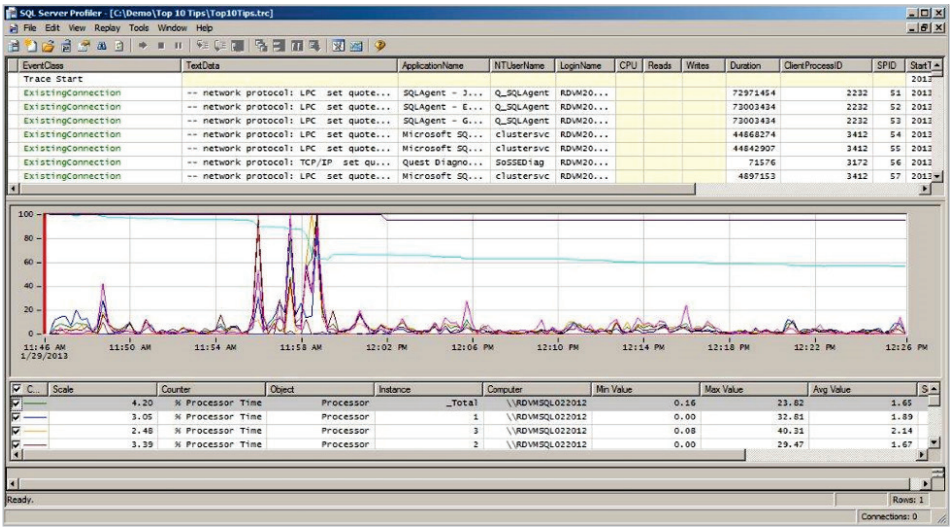

*図2.パフォーマンスカウンターとプロファイラーのトレースファイルの相関表示*

### ヒント5: SANの構成を通じてSQL Serverのパフォーマンス向上を実現する。

ストレージ・エリア・ネットワーク(SAN)は優れたソリューションです。SANを使用すると、ストレージのプロビジョニングおよび管理をシンプルな方法で行うことができます。SQL Serverの観点から、SANの構成を通じてパフォーマンスを向上させることもできますが、そのように構成することはほとんどありません。組織がSANを実装する理由は、通常、パフォーマンスの向上のためではなく、ストレージの統合や管理のしやすさのためです。さらに悪いことに、SANでは通常、プロビジョニングの実行方法を直接管理することができません。このため、1つの論理ボリュームに対してSANが構成されていることがほとんどです。この場合、そのボリュームにすべてのデータファイルを格納する必要があります。

#### I/Oパフォーマンスの向上のためにSANを構成するためのベストプラクティス

一般に、すべてのファイルを1つのボリュームに格納することは、I/Oパフォーマンスを最大限に高めたい場合には得策とは言えません。そのため、ベストプラクティスとしては以下のことを行います。

- **ログファイルを専用のボリュームに格納する。** データファイルとは別のボリュームにログファイルを格納します。ログファイルはほぼ独占的に、シーケンシャルに書き込まれ、読み取りは行われません(データベースミラーリング、AlwaysOn可用性グループなどは例外)。このため必ず、書き込みパフォーマンスを向上できるように構成する必要があります。
- **tempdbを専用のボリュームに格納する。** tempdbは、SQL Serverの内部でさまざまな目的のために使用されるため、専用のI/Oサブシステムに格納することをお勧めします。さらにパフォーマンスのチューニングを行うには、まず一部の統計データが必要です。
- **VLDB内に複数のデータファイルとファイルグループを作成することを検討する。これにより、**I/Oの並列処理によるメリットを得ることができます。
- **バックアップを専用のドライブに格納する。**これにより、冗長性が得られるほか、メンテナンス期間中の他のボリュームとのI/Oの競合を減らすことができます。

#### データの収集

Windowsディスクカウンターでは、言うまでもなく、Windowsで起きていることを把握することができます(RAID構成に基づいて、生の数値データを忘れずに調整してください)。SANベンダーは、独自のパフォーマンスデータを提供することがよくあります。

SQL Serverでは、以下のファイルレベルのI/O情報も提供されます。

- **SQL Server 2005より前のバージョン**: fn_virtualfilestats関数を使用します。
- **SQL Server 2005以降のバージョン**: sys.dm_io_virtual_file_stats動的管理関数を使用します。

この関数を次のコードで使用することにより、以下のことを実行できます。

- 読み取りと書き込みの両方のI/Oレートの抽出
- I/Oスループットの取得
- I/Oあたりの平均時間の取得
- I/Oの待ち時間の監視

```
SELECT db_name (a.database_id) AS [DatabaseName],
      b.name AS [FileName], a.File_ID AS [FileID],
      CASE WHEN a.file_id = 2 THEN 'Log' ELSE 'Data' END AS [FileType],
      a.Num_of_Reads AS [NumReads],
      a.num_of_bytes_read AS [NumBytesRead],
      a.io_stall_read_ms AS [IOStallReadsMS],
      a.num_of_writes AS [NumWrites],
      a.num_of_bytes_written AS [NumBytesWritten],
      a.io_stall_write_ms AS [IOStallWritesMS],
      a.io_stall [TotalIOStallMS],
      DATEADD (ms, -a.sample_ms, GETDATE ()) [LastReset],
      ( (a.size_on_disk_bytes / 1024) / 1024.0) AS [SizeOnDiskMB],
      UPPER (LEFT (b.physical_name, 2)) AS [DiskLocation]
  FROM sys.dm_io_virtual_file_stats (NULL, NULL) a
    JOIN sys.master_files b
      ON a.file_id = b.file_id AND a.database_id = b.database_id
ORDER BY a.io_stall DESC;
```

デフォルトでバックアップを圧縮するように設定すると、I/Oの競合を低減できます。

**データの分析**

クエリの「LastReset」の値には特に注意を払ってください。この値から、SQL Serverのサービスが最後に開始された時刻が分かります。動的管理オブジェクトのデータは保持されないため、チューニングの目的で使用されているデータはすべて、サービスが実行されていた期間に照らして検証する必要があります。そうしないと、誤った想定が行われる可能性があります。

これらの値を利用することで、I/O帯域幅を消費しているファイルを素早く絞り込むことができるほか、以下のような質問をすることができます。

- このI/Oは必要か?インデックスが不足しているか?
- 原因となっているファイル内のテーブルまたはインデックスは1つか?そのインデックスまたはテーブルを別のボリューム上の別のファイルに格納することは可能か?

ハードウェアのチューニングに関するヒント

データベースファイルが適切に配置され、すべてのオブジェクトホットスポットが特定されて別々のボリュームに分離されていれば、次はハードウェアを詳しく調べます。

ハードウェアのチューニングについてはスペシャリスト向けのトピックのため、このホワイトペーパーの範囲外です。そのためここでは、ハードウェアのチューニングをシンプル化するためのベストプラクティスとヒントをいくつか紹介します。

- SQL Serverで使用するためのボリュームを作成する場合、デフォルトのアロケーションユニットサイズを使用しないでください。SQL Serverは64 KBのエクステントを使用するため、この値を最小にする必要があります。
- パーティションが適切に配置されているかどうかを確認してください。この件については、Jimmy May氏が執筆したホワイトペーパーを参照してください。パーティションが適切に配置されていないと、パフォーマンスが最大30 %低下することもあります。
- SQLIOなどのツールを使用して、システムのI/Oのベンチマークを実施してください。このツールの詳細については、チュートリアルをご覧ください。

| DatabaseName | FileName | FileID | FileType | NumReads | NumBytesRead | IOStallReadsMS | NumWrites | NumBytesWritten | IOStallWritesMS | TotalIOStall... | LastReset | SizeOnDiskMB | DiskLocation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AWTarget2012 | AdventureWorks2012_Data | 1 | Data | 91 | 5709824 | 10626 | 8 | 73728 | 2 | 10628 | 31/01/201... | 205.000000 | C: |
| NewTest | NewTest | 1 | Data | 36 | 2113536 | 9963 | 1 | 8192 | 19 | 9982 | 31/01/201... | 5.000000 | C: |
| AWSource2012 | AdventureWorks2012_Data | 1 | Data | 84 | 5251072 | 9731 | 2 | 16384 | 0 | 9731 | 31/01/201... | 205.000000 | C: |
| AW20121 | AdventureWorks2012_Data | 1 | Data | 74 | 4677632 | 9151 | 1 | 8192 | 0 | 9151 | 31/01/201... | 205.000000 | C: |

*図3.SQL Serverからのファイルレベルの*I/O*情報*

SQL Server Profilerは引き続き、多くのSQL Server環境において、アプリケーションの機能に関する優れた洞察をリアルタイムで提供します。

## ヒント4: カーソルやその他の不適切なT-SQLのせいで、アプリケーションに問題が発生することがよくある。

### 悪いコードの例

私は以前の仕事で、これまでに見た中で最悪と思われるコードに遭遇したことがあります。システムが交換されてからしばらく経ちましたが、関数が行っていた処理の概要は次のようなものでした。

1. ストリップするパラメーター値を受け取る。
2. ストリップするパラメーター式を受け取る。
3. 式の長さを求める。
4. 式を変数にロードする。
5. ループ処理を行い、変数内の各文字がストリップする値のいずれかと一致するかどうかをチェックする。一致した文字を削除して変数を更新する。
6. 式のチェックが完全に終わるまで、次の文字に移る。

驚いた人も少なくないでしょう。私が説明しようとしているのは、もちろん、T-SQLの独自の「REPLACE」ステートメントを作成しようとしている人のことです。

最悪なのは、この関数がメーリングルーチンの一部としてアドレスを更新するために使用され、1日に何万回も呼び出されていたことです。

サーバー側でプロファイラーのトレースを実行すると、サーバーのワークロードを表示して、コードで頻繁に実行される部分を見付けることができます（上の「貴重な処理」が見つかったのは、この方法のおかげです）。

### クエリプランを使用したクエリのチューニング

不適切なT-SQLは、インデックスを使用しない非効率的なクエリとして現れることもあります。これは主に、インデックスが不適切であるか、不足していることが原因です。クエリプランを使用してクエリをチューニングする方法を十分に理解することは非常に重要です。

このホワイトペーパーでは、クエリプランを使用したクエリのチューニングに関する詳細な説明は行いません。ただし、このプロセスを最も簡単に開始できる方法を紹介しておきます。それは、SCAN処理をSEEKに変える方法です。SCANではテーブルまたはインデックス内のすべての行を読み込むため、大規模なテーブルの場合にはI/Oの負荷が大きくなります。一方、SEEKでは、インデックスを使用して、必要な行に直接アクセスします（これには当然、インデックスが必要です）。ワークロード内にSCANがある場合は、インデックスが不足する可能性があります。

このトピックに関しては、以下を含め、優れたドキュメントが多数あります。

- 『Professional SQL Server Execution Plan Tuning（SQL Serverの実行プランのチューニング: プロフェッショナル向け）』、Grant Fritchey著
- 『Professional SQL Server 2012 Internals & Troubleshooting（SQL Server 2012の構成とトラブルシューティング: プロフェッショナル向け）』、Christian Bolton、Rob Farley、Glenn Berry、Justin Langford、Gavin Payne、Amit Banerjee、Michael Anderson、James Boother、Steven Wort共著
- 『T-SQL Fundamentals for Microsoft SQL Server 2012 and SQL Azure（Microsoft SQL Server 2012およびSQL AzureにおけるT-SQLの基本）』、Itzik Ben-Gan著

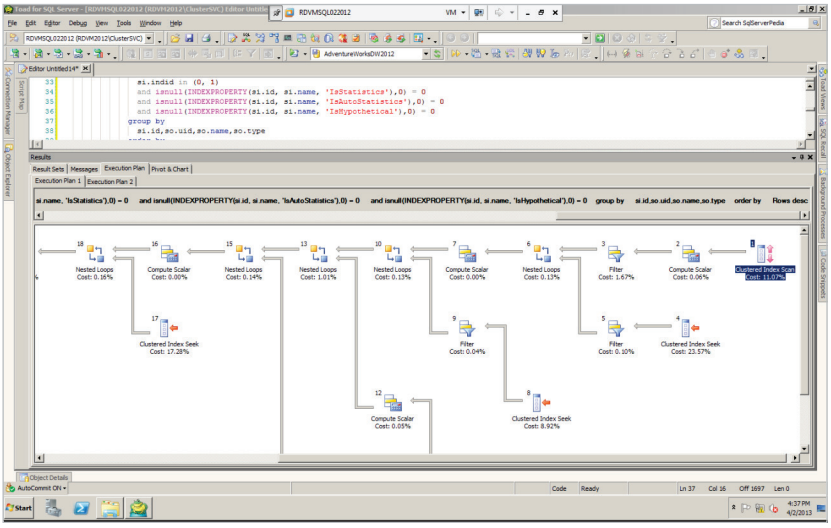

*図4.大規模なクエリプランの例*

## ヒント3: SQL Serverのキャッシングを改善するため、プランを最大限に再利用する。

**クエリプランを再利用することが重要な理由**
SQL Serverは、SQLステートメントを実行する前に、まずクエリプランを作成します。これにより、クエリの論理的な命令を取得し、それをデータに対する物理的なアクションとして実装するためにSQL Serverが使用する方法が定義されます。

クエリプランの作成には、大量のCPUリソースが必要になることがあります。そのため、SQLステートメントを実行するたびに新しいクエリプランを作成するのではなく、クエリプランを再利用することができれば、SQL Serverはより効率的に動作できます。

**プランを適切に再利用できているかどうかの評価**
SQL Statisticsのパフォーマンスオブジェクトには、使用可能なパフォーマンスカウンターがいくつか含まれています。これらのカウンターを通じて、プランを適切に再利用できているかどうかを確認できます。次の式では、コンパイルに送信されるバッチの比率を求めることができます。

この値はできる限り小さくする必要があります。1:1の比率は、送信されたすべてのバッチがコンパイルされ、プランの再利用がまったく行われていないことを意味します。

(Batch Requests/sec - SQL Compilations/sec) / Batch Requests/sec

**プランの再利用率が低い場合の対処**
プランの再利用率を低くしているワークロードを正確に突き止めるのは簡単なことではありません。その理由は、多くの場合、クエリを送信しているクライアントアプリケーションのコード内に問題があるためです。その結果、クエリを送信しているクライアントアプリケーションのコードの調査が必要になる場合があります。

クライアントアプリケーション内に埋め込まれているコードを見付けるには、拡張イベントかプロファイラーのいずれかを使用する必要があります。SQL:StmtRecompileイベントをトレースに追加することにより、再コンパイルイベントが発生するタイミングを確認できるようになります(SP:Recompileというイベントもあります。このイベントは、SQL Server 2005において再コンパイルの実行がプロシージャレベルからステートメントレベルに変更されたため、後方互換性を実現するために追加されたものです)。

よくある問題として、コードが作成済みのパラメーター化されたステートメントを使用していないことがあります。パラメーター化されたクエリを使用すると、プランの再利用やコンパイルのオーバーヘッドが改善されるだけでなく、文字列の連結を介してパラメーターを渡すことで生じるSQLインジェクション攻撃のリスクを低減することもできます。図5に、2つのサンプルコードを示します。これらのコードは工夫して作成されており、文字列の連結を利用してステートメントを作成する場合と、作成済みのステートメントをパラメーターと組み合わせて使用する場合の違いが示されています。

SQL Serverの観点から、SANの構成を通じてパフォーマンスを向上させることもできますが、そのように構成することはほとんどありません。

SQL Serverからのファイルレベルの I/O 情報は、I/O帯域幅を消費しているファイルを特定するのに役立ちます。

```
public void ExecuteSomeSQL(int aParam) {
    //cmd is a SqlCommand created somewhere else

    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "select foo1, foo2, foo3 from bar where foo1=" + aParam.ToString();

    SqlDataReader dr = cmd.ExecuteReader();
    try {
        while (dr.Read()) {

        }
    } finally {
        dr.Close();
    }
}
```

悪い

```
public void ExecuteSomeSQL(int aParam) {
    //cmd is a SqlCommand created somewhere else

    cmd.CommandType = CommandType.Text;
    cmd.CommandText = "select foo1, foo2, foo3 from bar where foo1 = @foo1";

    cmd.Parameters["@foo1"].Value = aParam;

    SqlDataReader dr = cmd.ExecuteReader();
    try {
        while (dr.Read()) {

        }
    } finally {
        dr.Close();
    }
}
```

良い

*図5.文字列の連結を利用してステートメントを作成するコードと作成済みのステートメントをパラメーターと組み合わせて使用するコードの比較*

SQL Serverでは、図5の「悪い」例のプランを再利用することはできません。パラメーターが文字列型であった場合は、この関数を使用してSQLインジェクション攻撃を展開することができました。「良い」例は、SQLインジェクション攻撃を受けにくくなっています（パラメーターが使用されているため）。また、SQL Serverでプランを再利用することができます。

## ヒント8で説明できなかった構成設定

記憶力の良い人は、ヒント8の内容を覚えていると思います（構成の変更について説明しました）。構成設定に関して、もう1つアドバイスが残っています。SQL Server 2008で、「アドホックワークロードの最適化」という構成設定が導入されました。これを設定すると、SQL Serverに対して、プランキャッシュに完全なプランではなくスタブプランを格納するように指示されます。この設定は、動的に作成されたT-SQLコードや、コードが再利用されなくなる可能性があるLINQを使用する環境では特に有用です。

プランキャッシュに割り当てられたメモリは、バッファプール内に配置されます。このため、プランキャッシュが肥大化すると、バッファキャッシュに格納できるデータページの量が少なくなります。これにより、I/Oサブシステムからデータをフェッチするためのラウンドトリップが増え、I/O負荷が極めて大きくなることがあります。

## ヒント2: SQL Serverのバッファキャッシュを読み取る方法と、キャッシュスラッシングを最小限に抑える方法を身に付ける。

### バッファキャッシュが重要な理由

既に簡単に説明しましたが、バッファキャッシュとは、SQL Serverが物理I/Oを実行する必要性を低減するために使用する大きな領域のメモリです。SQL Serverのクエリを実行しても、データがディスクから直接読み取られることはありません。データベースページは、バッファキャッシュから読み取られます。人気の高いページがバッファキャッシュ内にない場合、物理I/O要求がキューに入れられます。その後、クエリが待機し、ページがディスクからフェッチされます。

DELETEまたはUPDATE操作によるページ上のデータに対する変更は、バッファキャッシュ内のページに対しても行われます。このような変更は、後でディスクにフラッシュされます。このメカニズム全体を利用することにより、SQL Serverでは複数の方法で物理I/Oを最適化することができます。

- 複数ページの読み取り/書き込みを1回のI/O動作で行うことができます。
- 先読みを実装できます。SQL Serverでは、特定のタイプの操作において、連続するページを読み取るのに先読みが役に立つ場合があります。ただし、要求されたページを読み取った直後という前提のため、隣接するページを読み取る必要があります。

**メモ**: インデックスが断片化されていると、SQL Serverが先読みの最適化を実行できなくなります。

### バッファキャッシュの正常性の評価

バッファキャッシュの正常性に関する指標には、次の2つがあります。

- MSSQL$Instance:Buffer Manager\Buffer cache hit ratio: これは、キャッシュ内で見つからなかったページ(ディスクから読み取る必要があるページ)に対するキャッシュ内で見つかったページの比率です。理想としては、この数値をできる限り大きくする必要があります。ヒット率を高くすることは可能ですが、それでもキャッシュスラッシングは発生します。
- MSSQL$Instance:Buffer Manager\Page Life Expectancy: これは、SQL Serverがバッファキャッシュ内のページを削除するまでに維持する時間です。マイクロソフトによると、ページの予測保持期間が5分を超えている場合は正常です。予測保持期間が5分未満の場合は、メモリ不足またはキャッシュスラッシングが発生している可能性があります。

このセクションは例えで締めくくりたいと思います。多くの人がアンチロックブレーキやその他のサポートテクノロジーの革新によって制動距離が短くなるはずだと主張していますが、こうした新しいテクノロジーの出現に伴って制限速度を上げることができるとも主張しています。ページの予測保持期間に対する警告値が300秒(5分)であることに関して、SQL Serverコミュニティにおいて同様の議論が巻き起こっています。この値を厳しいと考える人もいれば、今日のほとんどのサーバーではメモリ容量が増加しているため、この値を百秒単位ではなく千秒単位にするべきだと考えている人もいます。この意見の違いは、基準の重要性に加え、環境内で使用されている各パフォーマンスカウンターの警告レベルがどのような値になっているかを詳細に把握することが極めて重要な理由を認識させてくれます。

### キャッシュスラッシング

大規模なテーブルやインデックスのスキャンでは、すべてのページがバッファキャッシュを通過する必要があります。このため、複数回読み取られる可能性が低いページ用のスペースを作るために、有用と思われるページが削除されてしまいます。これにより、削除されたページをディスクから再度読み取らなければならなくなるため、I/O負荷が高くなります。こうしたキャッシュスラッシングの発生は、通常、大規模なテーブルやインデックスがスキャンされていることを示します。

バッファキャッシュ内のほとんどのスペースを占有しているテーブルやインデックスを特定するには、sys.dm_os_ buffer_descriptors DMV(SQL Server 2005以降で使用可能)を調べます。以下のサンプルクエリは、SQL Serverのバッファキャッシュ内のスペース

データベースファイルが適切に配置され、すべてのオブジェクトホットスポットが特定されて別々のボリュームに分離されていれば、次はハードウェアを詳しく調べます。

```
SELECT o.name, i.name, bd.*
  FROM sys.dm_os_buffer_descriptors bd
     INNER JOIN sys.allocation_units a
        ON bd.allocation_unit_id = a.allocation_unit_id
     INNER JOIN
     sys.partitions p
        ON (a.container_id = p.hobt_id AND a.type IN (1, 3))
           OR (a.container_id = p.partition_id AND a.type = 2)
     INNER JOIN sys.objects o ON p.object_id = o.object_id
     INNER JOIN sys.indexes i
        ON p.object_id = i.object_id AND p.index_id = i.index_id
```

サーバー側でプロファイラーのトレースを実行すると、サーバーのワークロードを表示して、コードで頻繁に実行される部分を見付けることができます。

を消費しているテーブルまたはインデックスのリストにアクセスする方法を示しています。

インデックスDMVを使用することで、大量の物理I/Oを伴うテーブルやインデックスを特定することも可能です。

### ヒント1: インデックスの使用方法を理解し、無効なインデックスを見付ける。

SQL Server 2012には、インデックスに関する極めて有用なデータがいくつかあります。それらのデータは、SQL Server 2005で実装されたDMOを使用してフェッチできます。

#### sys.dm_db_index_operational_stats DMOの使用

sys.dm_db_index_operational_statsには、インデックスごとの現在の低レベルのI/O、ロック、ラッチ、およびアクセスメソッドのアクティビティに関する情報が格納されます。このDMFを使用すると、以下の疑問に対する回答が得られます。

- 「ホット」インデックスがあるか?競合が存在するインデックスがあるか? row_lock_wait_in_ms/page_lock_wait_in_ms列では、このインデックス上に待機が存在していたかどうかが分かります。
- 非効率的な方法で使用されているインデックスがあるか?現在、どのインデックスがI/Oのボトルネックになっているか? page_io_latch_wait_ms列では、インデックスページをバッファキャッシュに格納している間にI/Oの待機が存在していたかどうか分かります。これは、スキャンのアクセスパターンがあることを示す優れた指標になります。
- どのようなアクセスパターンが使用されているか? range_scan_count列とsingleton_lookup_count列では、特定のインデックスでどのようなアクセスパターンが使用されているかが分かります。

図6に、PAGE_IO_LATCH waitの合計ごとにインデックスをリストするクエリの出力を示します。これは、I/Oのボトルネックに関与しているインデックスを確認する際に、非常に役に立ちます。

#### sys.dm_db_index_usage_stats DMOの使用

sys.dm_db_index_usage_statsには、各種のインデックス操作の回数と、それらの操作を最後に実行した時刻が格納されます。このDMVを使用すると、以下の疑問に対する回答が得られます。

- ユーザーがインデックスをどのように使用しているか? user_seeks、user_scans、user_lookupsの各列では、インデックスに対するユーザーの操作のタイプと重要度が分かります。
- インデックスのコストはどのようなものか? user_updates列では、インデックスのメンテナンスレベルが分かります。
- 最後にインデックスを使用したのはいつか? last_*列では、インデックスに対する操作が最後に行われた日時が分かります。

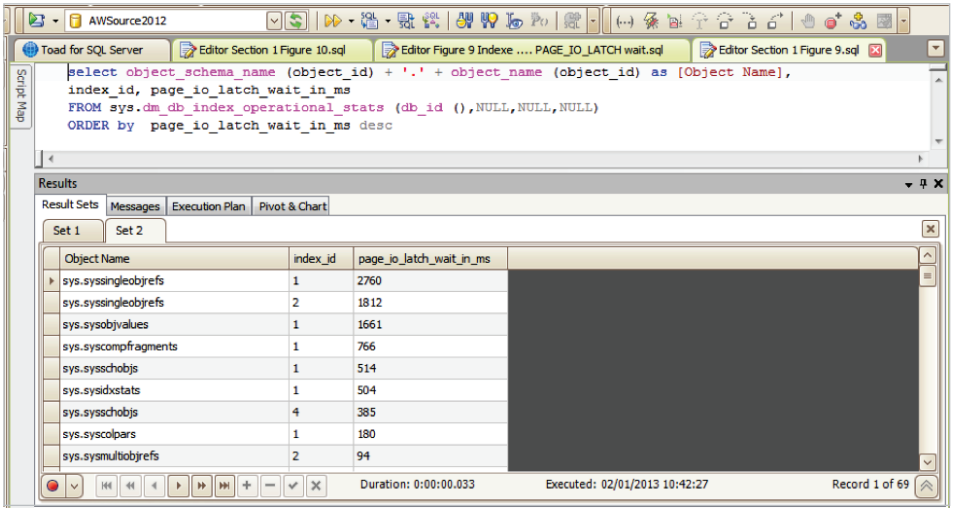


*図6.PAGE_IO_LATCH waitの合計ごとにリストされたインデックス*

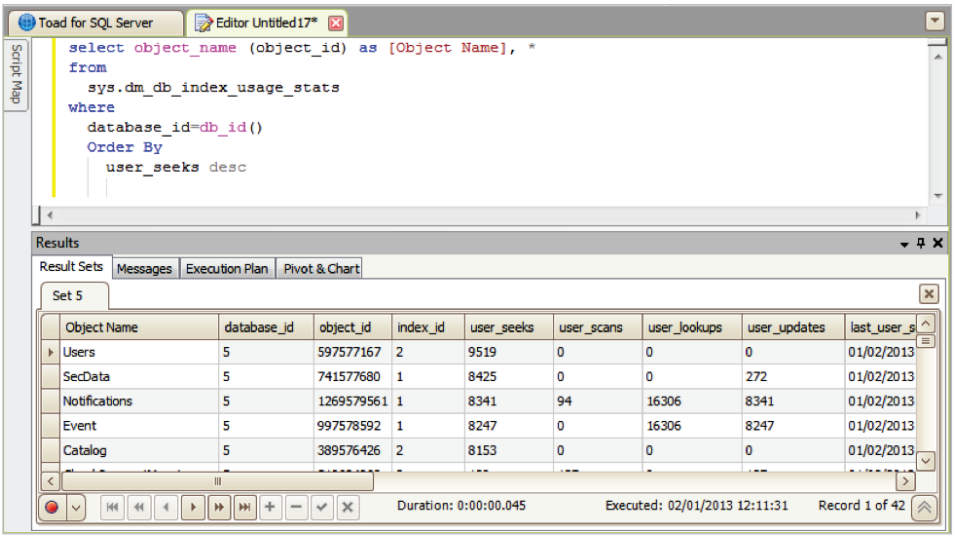

図7.user_seeksの合計数ごとにリストされたインデックス

図7に、user_seeksの合計数ごとにインデックスをリストするクエリの出力を示します。そうではなく、スキャンの割合が多いインデックスを特定したい場合は、user_scans列で並べ替えます。これで、インデックス名が特定されます。そのインデックスを使用したSQLステートメントを特定できれば便利だと思いませんか? SQL Server 2005以降のバージョンでは可能です。

もちろん、設計戦略、統合、メンテナンスなど、インデックスには他にもさまざまな側面があります。SQL Serverのパフォーマンスチューニングに関する重要な要素についてさらに学習したい場合は、SQLServerPediaにアクセスするか、該当するデルのWebキャストやホワイトペーパーを調べてみてください。

## まとめ

もちろん、SQL Serverのパフォーマンスに関して知っておくべきことは10個だけではありません。しかし、このホワイトペーパーは、優れた開始点になるだけではありません。既存のSQL Server環境に適用可能な、パフォーマンスの最適化に関する実用的なヒントも紹介しています。

最後に、SQL Serverのパフォーマンスを最適化する際の10個のヒントを思い出してみましょう。

10. ベンチマーキングを行うことで、通常の動作がどのようなものかを示す優れた指標が得られるため、ワークロードの動作を比較しやすくなり、異常な動作を特定できるようになる。
9. パフォーマンスカウンターは、現在の運用に関する手軽で有用な情報になる。
8. サーバーの設定を変更することで、より安定した環境を実現できる。
7. DMOは、パフォーマンスのボトルネックを素早く特定するのに役立つ。
6. SQL Profiler、トレース、および拡張イベントの使い方を身に付ける。
5. SANは、I/Oを実行する単なるブラックボックスではない。
4. カーソルやその他の不適切なT-SQLのせいで、アプリケーションに問題が発生することがよくある。
3. SQL Serverのキャッシングを改善するため、プランを最大限に再利用する。
2. SQL Serverのバッファキャッシュを読み取る方法と、キャッシュスラッシングを最小限に抑える方法を身に付ける。

そして、SQL Serverのパフォーマンスの最適化に関する最後のヒントは次の通りです。

1. インデックスの使用方法と、不適切なインデックスを見付ける方法を身に付けて、インデックス作成をマスターする。

ページの予測保持期間が5分未満の場合は、メモリ不足またはキャッシュスラッシングが発生している可能性があります。

パラメーター化されたクエリを使用すると、プランの再利用やコンパイルのオーバーヘッドが改善されるだけでなく、文字列の連結を介してパラメーターを渡すことで生じるSQLインジェクション攻撃のリスクを低減することもできます。

## 次のステップ

皆さんはきっと、このホワイトペーパーで学んだ教訓を実践したいと思っているでしょう。以下の表には、より最適なSQL Server環境を実現するために、最初に行うべきことが記載されています。

| アクション | サブタスク | 実施予定日 |
|---|---|---|
| プロジェクトを開始する承認を得る | ラインマネージャと話をして、実施するプロジェクトに関するケースを提示することで、受け身的でなく積極的になることができます。 | |
| パフォーマンスの目標を定める | 社内の利害関係者と話をして、受け入れ可能なパフォーマンスレベルを決定します。 | |
| システムパフォーマンスの基準を設定する | 関連するデータを収集して、カスタム構成またはサードパーティ製のリポジトリに保存します。 | |
| 最優先するパフォーマンスカウンターを特定し、トレースおよび拡張イベントを設定する | デルのPerfmonカウンターポスターをダウンロードします。 | |
| サーバーの設定を見直す | メモリと「アドホックワークロードの最適化」の設定には特に注意を払います。 | |
| I/Oサブシステムを見直す | 必要に応じて、SAN管理者と話をして、SQLIOなどのツールを使用したI/O負荷テストを実施することを検討します。または単に、負荷の大きな読み取り/書き込み操作（バックアップの実行時など）を行うことができる頻度を決定します。 | |
| 低パフォーマンスのクエリを特定する | トレース、拡張イベントセッション、およびプランキャッシュから返されたデータを分析します。 | |
| 低パフォーマンスのコードをリファクタリングする | SQLServerPediaのシンジケート化されたブログサービスで、最新のベストプラクティスをチェックします。 | |
| インデックスのメンテナンス | インデックスをできる限り最適な状態に維持します。 | |

## 詳細情報



本書に記載されている情報は、デルの製品の概要説明を目的としたものです。本書によって、あるいはデル製品の販売に関連して、明示または黙示にかかわらず、禁反言やその他の方法によって生じる、いかなる知的財産に対するライセンスも許諾されません。当該製品のライセンス契約で指定されているデルの約款に記載されている場合を除き、デルはいかなる責任も負うものではなく、商品性、特定目的への適合性、または非侵害性に関する黙示的保証を含め(ただしこれらに限定されない)、その製品に関連する一切の明示的、黙示的、または法令による保証を行いません。デルは、いかなる場合においても、本書の使用または使用不可能に起因する直接損害、間接損害、結果的損害、懲罰的損害、特別損害、または付随的損害(営業利益の損失、ビジネスの中断、情報の紛失を含むがこれらに限定されない)について、仮にそれらの発生の可能性を知らされていたとしても、一切の責任を負いません。デルは、本書の内容の正確性または完全性に関する保証または表明を行わず、仕様および製品の説明に対する変更をいつでも予告なく行う権利を有します。デルは、本書に記載されている情報を更新する確約を一切行いません。

## デルについて

Dell Inc.(NASDAQ: DELL)は、お客様の声に耳を傾け、お客様の信頼に応えて、価値ある革新的なテクノロジー、ビジネスソリューション、およびサービスを世界中に提供しています。詳細については、www.dell.comをご覧ください。

本書の使用に関して不明な点がありましたら、以下までお問い合わせください。

### Dell Software

5 Polaris Way
Aliso Viejo, CA 92656
www.dell.com
地域およびグローバルオフィスの情報についてはWebサイトをご覧ください。