パフォーマンスチューニングガイド

Release 4.0
Document Version 4.0-1

www.calpont.com

InfiniDB パフォーマンスチューニングガイド
2013 年 10 月

目次

はじめに

『InfiniDBパフォーマンスチューニングガイド』へようこそ。行ベースのデータベースでの経験が豊富な開発者またはDBAの観点から見ると、InfiniDB操作には従来のデータベース操作に関連するものと、直接関係のないものがあります。また、従来の行ベースのDBMSシステムに共通の基礎となる操作が、InfiniDB内には存在しない場合があります(たとえば、InfiniDBでは全表スキャンは実行されません)。

本書は、I/O要件を大幅に削減して大容量データの大幅なパラレル化および拡張を可能にする、列指向RDBMSのInfiniDBのチューニングに役立ちます。

分析用データベースInfiniDBでの操作は、マルチスレッド化および分散(オプション)されたシステムの大容量データのスキャン、結合、集計が優れたパフォーマンス特性で行われるように最適化されています。ここで使用される大容量データは、数億行から数千億行に渡る場合があり、また、数百GBから数百TBのデータになる場合があります。InfiniDBは、これらの数値をはるかに超える場合でも対応できるように設計されています。また、InfiniDBでは、スケールが小さい場合でも優れたパフォーマンスが提供されます。

対象読者

本書は、以下に示す様々な役割の読者を対象としています。

- データベース管理者
- アプリケーションおよびWebの開発者
- データの設計者
- システムおよびWebの管理者

マニュアルリスト

Calpont データベースプラットフォームのマニュアルは、様々な読者を対象とした複数のガイドで構成されています。次の表を参照してください。

マニュアル	説明
『InfiniDB 管理者ガイド』	Calpont InfiniDB を管理するための詳細な手順について説明します。
『InfiniDB 概要』	分析用データベース Calpont InfiniDB の概要について説明します。
『InfiniDB 最小推奨仕様ガイド』	Calpont InfiniDB の実装に必要なハードウェアおよびソフトウェアの最小の推奨仕様を示します。
『InfiniDB インストレーションガイド』	Calpont InfiniDB をインストールするために必要な手順の概要について説明します。
『InfiniDB マルチ UM 構成ガイド』	マルチユーザーモジュールの構成情報について説明します。
『InfiniDB SQL 構文ガイド』	Calpont InfiniDB に固有の構文について説明します。

マニュアルの入手

英語版のマニュアルは、(http://www.infinidb.org/およびhttp://www.calpont.com)で入手することができます。追加の支援が必要な場合はinfinidb_doc@ashisuto.co.jpにご連絡ください。

マニュアルへのフィードバック

マニュアルの改善に向けて、フィードバック、コメントおよび提案をいただけますようお願いいたします。マニュアル名、バージョンおよびページ番号を添えてコメントをinfinidb_doc@ashisuto.co.jpにご送付ください。

追加リソース

Calpont InfiniDBのインストールおよびチューニング、またはCalpont InfiniDBを使用したデータの問合せに関して支援が必要な場合はinfinidb_doc@ashisuto.co.jpまでご連絡ください。

パフォーマンスチューニングの概要

InfiniDB のアーキテクチャの目標

パフォーマンスに関連する設計を決定づける多くのコアアーキテクチャの目標があります。InfiniDBの目標は次のとおりです。

1. メモリーアクセスおよびストレージアクセスの両方で問合せのI/Oコストを大幅に削減する。
   - 大規模データに対するほとんどの問合せで必要なI/Oを根本的に削減する。
   - データのブロック(ページ)へのランダムなアクセスを排除する。
   - 大規模なデータセットの部分キャッシュまたは完全キャッシュを可能にするグローバルなデータバッファキャッシュを実装する。
2. 最小限の同期で、データベース操作を並行して実行できるようにする。
   - データのブロックの読取りに関連する同期を排除する。
   - 重要なスレッドプールに対して送信および受信を行うキューなど、多くのメカニズムを使用してスレッドの同期の問題を最小限にする。
3. 次のようなパラレルのデータベース操作の作業単位を定義する。
   - 様々なストレージシステムで正常に機能する。
   - パフォーマンスモジュールが常に100%近くのCPU使用率で動作できるようにする。
   - スレッドを問合せ専用にしない(つまり、ロード時でも小規模の問合せが短時間で実行されるようにする)。
4. データの送信コストを最小限にする(つまり、ローカルスレッド間または分散されたスレッド間でのデータの送信を最小限にする)。
   - 可能な場合、データを送信するのではなく、データの操作を送信する。
   - 可能な場合、最大の表に対する送信コストが発生しないように操作を設定する。
5. チューニング要件を最小限にし、詳細データの非定型分析のパフォーマンスが向上する。

現時点では、1行の検索にかかる経過時間の最小化は主要な目標に含まれないことに注意してください。10億行から1行を検索するパフォーマンスは一瞬にして実現できますが、従来の索引と表の組合せほど効率的ではない場合があります。

InfiniDB を使用した結合の高速化

InfiniDBでは、従来のMySQLの結合を使用せずに、InfiniDBエンジン内で結合が実現されます。これらの結合の特性は次のとおりです。

1. ハッシュ結合操作によって数百万または数十億の行に対して最適化されます。
2. スケーラブルなスレッドプールでマルチスレッド化および分散されます。
3. 中間結果をマテリアライズせずに任意の数のディメンション表に対して大規模なファクト表を1回のパスでストリームします。

4. データ送信コストを大幅に削減する可能性がある集計操作を行って、送信されます。

このマニュアル内のパフォーマンスチューニングのガイドラインは、分析用データベースInfiniDB内でのみ実行される結合およびその他の操作に適用されます。

構文に関する注意点

InfiniDBでは、使用可能なすべての構文に対して完全なパフォーマンス機能を備えているわけではありません。サポートされている構文については、『InfiniDB SQL構文ガイド』マニュアルを参照してください。新しい機能が追加されると、構文に対するその拡張機能では、他の操作の場合と同様に、マルチスレッド化されたスケーラブルなパフォーマンスが積極的に活用されます。

現在、追加の構文は代替構成（つまり、InfiniDBを標準ストレージエンジンとして構成すること）によって利用できます。ただし、この場合は、マルチスレッド結合または分散結合および集計操作を行うことができなくなります。

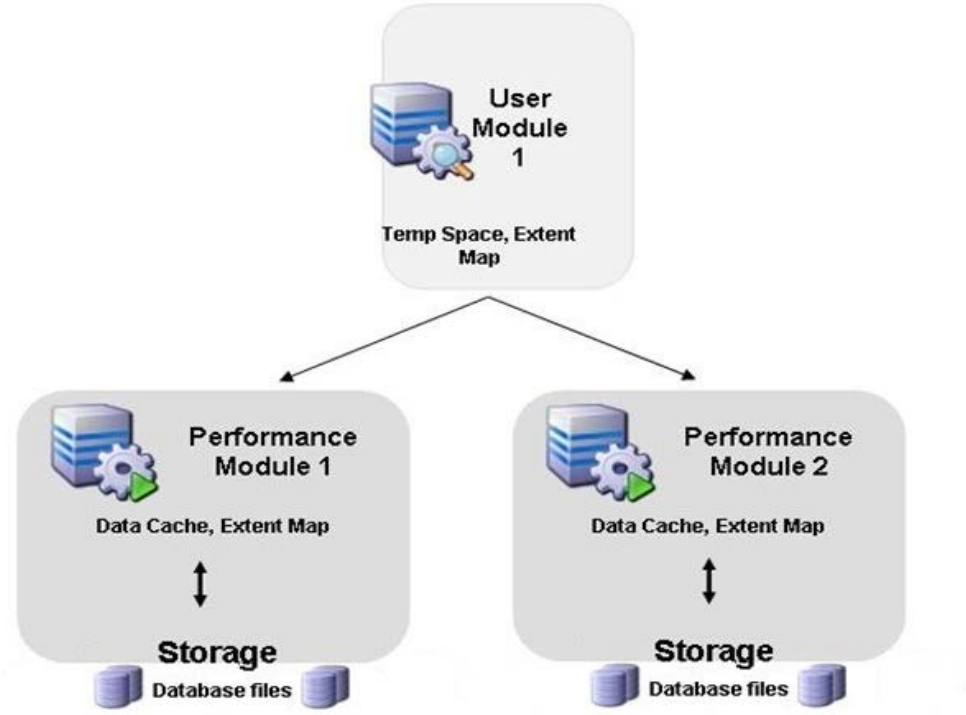
InfiniDB の主要コンポーネント

InfiniDBでは、3つの主要コンポーネントで構成されるモジュールアーキテクチャが提供されます。3つすべてのコンポーネントが連携してInfiniDBインスタンスを構成しています。これらのコンポーネントには次のものがあります。

- **ユーザーモジュール（UM）**：ユーザーモジュールは、SQLリクエストを分割して1つ以上のパフォーマンスモジュールに分散します。パフォーマンスモジュールによって、リクエストされたデータがメモリーキャッシュまたはディスクのいずれかから実際に取得されます。1つのユーザーモジュールで1つの問合せの状態が保持されます。
- **パフォーマンスモジュール（PM）**：パフォーマンスモジュールは、マルチスレッド方式で、ユーザーモジュールから受信した細かい作業単位を実行します。また、InfiniDBでは、多数のパフォーマンスモジュール間で作業を分散できます。
- **ストレージ**：InfiniDBでは、ローカルストレージまたは共有ストレージ（SANなど）のいずれかを使用してデータを格納できます。ユーザーは、1つのみのサーバーをInfiniDBサーバーとして動作させ、すべてを構成してそのサーバー上で実行するか、または複数のサーバーにスケールアウトできます。

また、InfiniDBでは、エクステントマップと呼ばれる共有オブジェクト内の各列に関するメタデータが保持されます。エクステントマップは、ユーザーモジュールで発行する操作とその対象のデータを判断するときに参照されます。さらに、ディスクからのブロックの読取りに必要な場合はパフォーマンスモジュールによって参照されます。各列は1つ以上のファイルで構成され、各ファイルに複数のエクステント（通常はデータの連続した割当て）を含めることができます。エクステントも、エクステントマップ内で追跡されます。

UMでは問合せ計画が認識され、PMではデータブロックおよび操作が認識されます。エクステントマップによってこの抽象化が可能になります。



2つのパフォーマンスモジュールがインストールされた環境を簡単に示した図

分析用データベース InfiniDB のエディション

InfiniDBは、大規模なデータのロードおよび問合せを行うために構築された列指向のデータベースで、オープンソース版および製品版の両方で使用できます。InfiniDBのすべてのエディションに、パフォーマンスで重要となる次の機能が含まれています(機能についての詳細は、他のマニュアルを参照してください)。

- **列指向のアーキテクチャ**:InfiniDBでは、行ごとにではなく列ごとにデータが格納されます。これによって、スキャン操作でスキャンに含まれない列を無視したり、問合せで参照されない列を無視する列操作を選択することができます。
- **マルチスレッド設計**:InfiniDBでは、問合せをサポートするために操作が非常に細かいレベルで分散され、同期が最小限または行われない状態でそれらの操作を実行できるようにほとんどの問合せが構築されます。操作はすべてのパフォーマンスモジュールに自動的にマップされ、分散されます。結果は削減され、コール側のユーザーモジュールに返されます。
- **垂直および水平の自動パーティション化**:列ストレージに基づいたI/Oの大幅な削減に加えて、InfiniDBでは、ほとんどの列に対して小さいエクステント(パーティション)が作成され、状況によってはエクステントの除外を発生させることができるように一部のメタデータが格納されます。
- **柔軟な同時実行**:InfiniDBでは、データへの同時アクセスが可能です。これによって、大規模な問合せで、使用可能なすべてのパフォーマンスモジュールのリソースを短時間使用できるだけでなく、大規模な問合せが完了するまで待機せずに小規模な問合せを実行できます。

- **大規模パラレル処理（MPP）対応**：InfiniDBは複数の汎用ハードウェアマシンを使用して、総合的なパフォーマンスをほぼリニアに向上させることができます。複数のパフォーマンスモジュールを使用すると、使用可能なすべてのパフォーマンスモジュール上のすべてのスレッド間で細かい作業を分散できます。
- **シェアードナッシング分散データキャッシュ**：複数ノードによるInfiniDB構成では、データは様々なノードとそのデータキャッシュに分散されます。ノード間でデータを共有することはありませんが、問合せのためにデータを読み取るときは、InfiniDB MPPアーキテクチャ内のすべてのノードがアクセスされます。つまり、InfiniDBは、すべての参加ノードが並行して分散形式でアクセスする、1つの大きな論理データキャッシュを作成すると考えられます。これによって、十分な数のノードがあれば、InfiniDBは文字通り、大規模なデータベースをキャッシュできます。

InfiniDB の分散処理モデル

InfiniDBのジョブ発行処理は、Map/Reduce処理に例えることができます。UMは、スレッドプール（分散される場合がある）に操作を発行します。このスレッドプールは、データに対して個別にデータベース操作（フィルタ、集計、結合など）を実行し、次の操作を行うためにデータセット（削減されている場合がある）をUMに戻します。

ただし、InfiniDBとMap/Reduceにはいくつかの相違点があります。ジョブ（プリミティブ操作と呼ばれる）はSQL構文にマップ済で、利用可能な外部APIはないため、これはツールキットやソフトウェアコールではありません。また、ジョブのスケジュール処理にも相違点があります。

ユーザーモジュールからパフォーマンスモジュールへのリクエストの処理の粒度

データベース操作のリクエストはユーザーモジュールから1つ以上のパフォーマンスモジュールに発行され、800万行を含むデータブロックの範囲（エクステントと呼ばれる）に対してスキャンが実行されます。データがまだキャッシュされていない場合は、リクエストにより、この操作に加えてストレージからの読取りが発生する場合があります。システムのマルチブロック読取り機能を最大限に活用するために、1つのエクステント内のすべてのブロックがターゲットPMにマップされ、発行されます。また、同じエクステントのデータを必要とするその他の問合せも同じPMに送信されます。PMの数が増加した場合は、新たに追加されたPMを新しいリクエストに含めることができるように再マッピングが迅速に実行されます。非分散システムの場合でも、この粒度で操作を実行すると、隣接するブロックがまとめて読み取られて短い時間で処理されるなど、ブロックアクセスの空間的および時間的局所性が向上します。

大きい表内の部分的に移入されているエクステントと、小さい表用の部分的かつ唯一のエクステントの場合、操作のリクエストは各エクステント内の最高水位標（HWM）までに制限されることに注意してください。

エクステントのテスト済のデフォルト構成は800万行です。ディスク上のこの割当てサイズは、8-64MB（列内の1エクステントのディスク上のデータサイズ）の範囲で変わります。値はシステムのインストール時に設定されます。この値は、特定のInfiniDBインスタンス全体に対するものであり、スタートアップ後は変更できないことに注意してください。新しいインスタンスの動作は、別のマニュアルに記載されているCalpont.xmlパラメータによって設定できますが、追加の検証および確認が必要になります。

パフォーマンスモジュール内の処理の粒度（バッチプリミティブ）

パフォーマンスモジュール内のデータベース操作は、よりきめ細かい方法で分割され、個々のスレッドはエクステント内の個々のデータブロックで個別に動作します。最小の操作はプリミティブとも呼ばれます。プリミティブ処理を行う各PM上で実際に実行されるソフトウェアのプロセスはPrimProcです。バッチプリミティブでは、区切られたブロックセット内に格納されているすべての行に対して1つ以上のデータベースプリミティブが実行されます。

InfiniDBのブロックは8192バイトのデータで、固定長ストレージにおいては1024-8192行をサポートし、varchar列においては長さによって行数が変わります。varchar(7)およびそれより小さいものは固定長のフィールドにマップされ、固定長のフィールドとして格納されることに注意してください。

様々な状況において、ある特定のブロックがバッチプリミティブに含まれるかどうかを考える場合には、少し概念的な説明が役立ちます。実際、バッチプリミティブはデータサイズに応じて、1-8個のブロックに格納されている1列の小さい固定された範囲の行（8000行）に対して操作を実行します。また、バッチプリミティブ内の複数の個々のプリミティブは、複数の列でこの小さい固定されたサイズ範囲の行に対して操作を実行することがあります。そのため、最小のプリミティブは単一の8Kブロックのデータを読み込むことで終了する場合もありますし、その他の場合は多数の列にわたる問合せに対応するために100を超える数の個々のブロックからの読取りを行う場合があります。

バッチプリミティブステップ（BPS）

バッチプリミティブステップ（BPS）は、操作がUMから1つ以上のPMに発行される問合せ実行計画ステップで、UMで実行される可能性がある補足ステップも含まれます。利用可能なすべてのPM内の個々のスレッドは、割り当てられた範囲の行に対して、リクエストされたバッチプリミティブを実行します。実際、サーバーが1つか多数かにかかわらず、バッチプリミティブの処理に利用できるグローバルスレッドプールがあります。

問合せは、1つ以上のバッチプリミティブステップによって処理されます。バッチプリミティブステップは、問合せ要件に応じて、次の一部またはすべての処理を実行できます。

- **単一列のスキャン**：単一列の述語（=、<>、in（list）、between、is nullなど）に基づいて、特定の列の1つ以上のエクステントをスキャンします。
  詳細は、「最初のスキャン操作のチューニング」を参照してください。

- **単一列の追加フィルタ**：以前のスキャンで検出された行に対する追加の列を投影し、必要に応じて単一列の追加述語を適用します。ブロックのアクセスは行識別子に基づいており、ブロックに直接アクセスされます。
  詳細は、「追加の列の読取りのチューニング」を参照してください。

- **表レベルのフィルタ**：表レベルの任意のフィルタ（column1 < column2や、より高度な関数および式など）に必要な追加の列を投影します。この場合も、ブロックのアクセスは行識別子に基づいており、ブロックに直接アクセスされます。

- **結合での結合列の投影**：任意の結合操作に必要な追加の結合列を投影します。この場合も、ブロックのアクセスは行識別子に基づいており、ブロックに直接アクセスされます。
  詳細は、「InfiniDBの複数結合のチューニング」を参照してください。

- **複数結合の実行**：投影された結合列に対して1つ以上のハッシュ結合操作を適用し、その値を使用して、以前に構築されたハッシュマップを調べます。内部結合または外部結合の要件を満たすのに必要なタプルを構築します。以前に構築されたハッシュマップのサイズに応じて、PM処理を実行しているサーバーまたはUM処理を実行しているサーバーで実際の結合動作を実行できることに注意してください。いずれの場合も、バッチプリミティブステップの機能は同じです。
  詳細は、「InfiniDBの複数結合のチューニング」および「メモリー管理」を参照してください。

- **クロス表レベルのフィルタ**：クロス表レベルの任意のフィルタ（table1.column1 < table2.column2や、より高度な関数および式など）に必要な追加の列をプリミティブステップの行の範囲から投影します。この場合も、ブロックのアクセスは行識別子に基づいており、ブロックに直接アクセスされます。前提条件の結合操作がUMで実行された場合は、この操作もUMで実行されます。それ以外の場合は、PMで実行されます。
- **集計/重複行の削除操作（1回目）**：特定のバッチプリミティブに割り当てられた結合済の行のセットに対してローカルのグループ化操作、重複行の削除操作または集計操作を適用します。この処理の1回目は、パフォーマンスモジュールによって処理されます。
- **集計/重複行の削除操作（2回目）**：特定のバッチプリミティブに割り当てられた結合済の行のセットに対して最後のグループ化操作、重複行の削除操作または集計操作を適用します。この処理は、ユーザーモジュールによって処理されます。
  集計の詳細は、「メモリー管理」を参照してください。

バッチプリミティブは、I/Oを最小限に抑えるために、フィルタをできるかぎり早く適用し、追加の列の投影をできるかぎり遅くすることによって、列データファイルに対して表指向のSQLコマンドを効率的に実行します。さらに、バッチプリミティブは、グループ化操作、集計操作、重複行の削除操作を実行して、ユーザーモジュールに戻されるバイト数を削減します。

エクステントマップ

エクステントは、物理的なセグメントファイル内に存在する、指定した列の領域の論理ブロックで、サイズは8-64MBの間です。各エクステントでは、デフォルトで800万行がサポートされており、小さいデータ型であれば、ディスク上で消費される領域は少なくなります。

エクステントマップは、ストレージに保持されるすべてのデータのカタログとみなすことができます。各エクステントに1つのエントリが含まれます。各カタログエントリには、ブロックの範囲ごとの論理識別子（行識別子の一部）、部分的にデータが移入されている各エクステントの最高水位標、ほとんどのデータ型に対する各エクステントの最小値および最大値を保持する場所が含まれています。現在、8バイトを超える文字データ型および7バイトを超えるvarcharデータ型では最小値および最大値は移入されません。日付型、10進型、整数型、小さい文字列などの他のすべてのデータ型では最小値および最大値が移入されます。

スキャンによるエクステントマップ群

エクステントマップはDMLまたはバルクロード機能によって自動的に移入されるため、特別な統計収集は必要ありません。これらの最小値および最大値はディスクに保持されるため、システムの再起動後でもその情報を使用できます。

列の各エクステントに格納された最小値および最大値は、特定の状況でI/Oを除外するために使用され、セッションで最近実行された文に関する情報を表示する`select calgetstats();`関数を使用して表示できます。次の例では、`BlocksTouched`および`PartitionBlocksEliminated`に対してレポートされる値を示します。

- `BlocksTouched`：問合せをサポートするためにアクセスされた8KBのデータブロックの数。
- `PartitionBlocksEliminated`：最小値と最大値の比較にのみ基づいて除外されるブロックの数。

次の問合せは、60億行が含まれているスケール係数1000のスタースキーマベンチマーク（SSB）データセットに対して実行されます。`lo_orderdate`フィールドに基づいて一度に1か月分のデータがロードされており、ソートは実行されていません。この問合せでは、（大幅に値下げされた項目を除外するために）7700万のレコードに関して価格とコストを比較して、特定の日付範囲の分析を行います。その後、約6000万の個別の計算値の平均が計算されて、純収入が特定されます。

```
select  lo_discount, avg(lo_extendedprice - lo_supplycost), count(*)
    from  lineorder
   where  lo_orderdate between 19940101 and 19940131
          and lo_supplycost < lo_extendedprice * .5
group by  1 order by 2;

+-------------+---------------------------------------+----------+
| lo_discount | avg(lo_extendedprice - lo_supplycost) | count(*) |
+-------------+---------------------------------------+----------+
|        3.00 |                    3806802.786682 |  5510648 |
|        5.00 |                    3807880.542760 |  5511323 |
|        6.00 |                    3808025.905069 |  5511063 |
|        9.00 |                    3808060.552264 |  5503826 |
|        7.00 |                    3808450.935915 |  5506420 |
```

```
|       10.00 |                          3808529.811736 |  5508111 |
|        1.00 |                          3808842.678177 |  5509157 |
|        0.00 |                          3809090.349685 |  5510113 |
|        2.00 |                          3809309.979333 |  5507300 |
|        4.00 |                          3809375.700146 |  5506796 |
|        8.00 |                          3810136.417883 |  5509236 |
+-------------+-----------------------------------------+----------+
11 rows in set (3.20 sec)

+-------------------------------------------------------------------------------------------
-----------------------------------------+
| calgettats()
|
+-------------------------------------------------------------------------------------------
-----------------------------------------+
| Query Stats: MaxMemPct-0; NumTempFiles-0; TempFileSpace-0MB; ApproxPhyI/O-0;
CacheI/O-456180; BlocksTouched-456180; PartitionBlocksEliminated-2890042;
MsgBytesIn-4MB; MsgBytesOut-0MB; Mode-Distributed| 1256230262 897059 |
+-------------------------------------------------------------------------------------------
-----------------------------------------+
1 row in set (0.00 sec)
```

他の列に対するパーティションブロックの除外

最小値または最大値に基づいたパーティションブロックの除外は他の列に対しても発生する場合があります。通常は注文日に関連する別の日付フィールド（lo_commitdate）を使用するように問合せを変更すると、ブロックアクセスと経過時間の両方で同様の削減が行われます。昇順のキー値などの1つ以上のデータパターンを含むほとんどのデータセットで、同様のメリットが見込まれます。

```
mysql>   select  lo_discount, avg(lo_extendedprice - lo_supplycost), count(*)
    ->     from  lineorder
    ->    where  lo_commitdate between 19940101 and 19940131
    ->   and  lo_supplycost < lo_extendedprice * .5
    -> group by  1 order by  2 ;

+-------------+-----------------------------------------+----------+
| lo_discount | avg(lo_extendedprice - lo_supplycost) | count(*) |
+-------------+-----------------------------------------+----------+
|        1.00 |                          3807548.123096 |  5509631 |
|        2.00 |                          3807764.108009 |  5506100 |
|        3.00 |                          3808203.553947 |  5508280 |
|       10.00 |                          3808530.528851 |  5508275 |
|        0.00 |                          3808644.366889 |  5508041 |
|        8.00 |                          3808727.418191 |  5507120 |
|        6.00 |                          3809173.101862 |  5507031 |
|        4.00 |                          3809189.422454 |  5508216 |
|        9.00 |                          3809475.131073 |  5507767 |
|        5.00 |                          3809746.089397 |  5511763 |
|        7.00 |                          3809870.163028 |  5510417 |
+-------------+-----------------------------------------+----------+
11 rows in set (4.08 sec)

mysql>
mysql> select calgetstats();
+-------------------------------------------------------------------------------------------
-----------------------------------------+
| calgettats()
|
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------+
```

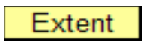
```
| Query Stats: MaxMemPct-0; NumTempFiles-0; TempFileSpace-0MB; ApproxPhyI/O-0;
CacheI/O-1561569; BlocksTouched-1561569; PartitionBlocksEliminated-2779450;
MsgBytesIn-14MB; MsgBytesOut-0MB; Mode-Distributed| 1256231405 276729 |
+-------------------------------------------------------------------------------------
-------------------------------------------+
1 row in set (0.00 sec)
```

列ストレージ、バッチプリミティブおよびエクステントマップ機能による I/O の除外

InfiniDBの列ストレージでは、問合せに含まれていない列に対するI/Oを自動的に削減できます。また、バッチプリミティブステップ内のデータをフィルタ処理する操作を積極的に使用することで、フィルタに従って参照される列のI/Oを大幅に除外できます。さらに、InfiniDBでは、バッチプリミティブの発行前に問合せが分析されるため、各列に格納されている最小値および最大値を利用することで、最初のスキャンのI/Oも削減される可能性があります。

それぞれのI/Oの除外方法を図で示すことができます。5つの列と、列ごとに13個のエクステント(12個は完全に移入され、1つは半分だけ移入されている)に分散された1億の行が含まれている単純な表があるとします。列aおよびbのフィルタに基づいて表から列a、bおよびcを選択する問合せでは、問合せを処理するためにアクセスされるエクステントは65個のうち5つのみとなる可能性があります。

- 列ストレージの最適化
  - 列dおよびeのブロックは、問合せで参照されなかったため無視されます。次の図では、除外されたエクステントは黄色で表示されています。

- エクステントマップの最適化
  - 列aのフィルタに対するエクステントマップの最小値および最大値は、エクステント1-9を除外します。次の図では、除外されたエクステントは緑色で表示されています。

  - 列bのフィルタはエクステント9-11を除外します。これによって、エクステント12および13のみがスキャンされます。次の図では、除外されたエクステントは青色で表示されています。

- バッチプリミティブの最適化
  - 列aおよびbに対してフィルタを実際に実行することによってエクステント13内の行が除外され、データセットがエクステント12内の数行に絞られた場合、参照されるのは列cのエクステント12とブロックの小さいサブセットのみが参照されます。次の図では、列cの除外されたエクステントはオレンジ色で表示されています。

Extent #	column a	column b	column c	column d	column e
1	Extent	Extent	Extent	Extent	Extent
2	Extent	Extent	Extent	Extent	Extent
3	Extent	Extent	Extent	Extent	Extent
4	Extent	Extent	Extent	Extent	Extent
5	Extent	Extent	Extent	Extent	Extent
6	Extent	Extent	Extent	Extent	Extent
7	Extent	Extent	Extent	Extent	Extent
8	Extent	Extent	Extent	Extent	Extent
9	Extent	Extent	Extent	Extent	Extent
10	Extent	Extent	Extent	Extent	Extent
11	Extent	Extent	Extent	Extent	Extent
12	Extent	Extent	Extent	Extent	Extent
13	Extent	Extent	Extent	Extent	Extent

1億行、5列のエクステントの図。

物理 I/O のチューニング

InfiniDBで物理I/Oをチューニングする最良の方法について説明する前に、まずこの項で使用するいくつかの用語を定義します。

- **エクステント**:通常、エクステントはストレージシステム内で連続して割り当てられた領域です。表の増大とともに、必要に応じて各列でエクステントが追加されます。エクステントは、ほとんどの場合、実際には8-64MBの範囲の可変サイズで、800万行を格納します。
- **DBRoot**:InfiniDBインスタンスの作成時にInfiniDBインスタンスで使用可能になるマウントポイント。
- **マルチブロック読取り**:ストレージからの多数の個々のブロック読取りを1つの読取り操作にまとめます。ほとんどのストレージサブシステムでは、主としてリクエストされた読取り操作のサイズに応じて、異なる持続速度(帯域幅)でデータブロックが提供されます。個々のランダムなブロックがリクエストされると、各ブロックをサポートするために機械的なディスクヘッド移動が発生する場合があるため、ディスクパフォーマンスが大幅に低下することがあります。

データベース内の特定の表に行が追加されると、必要に応じてエクステントが追加され、その表のすべての列のサイズが増大します。特定の表の各列の最初のエクステントセットが1つのDBRootに書き込まれます。同じ表の2番目のエクステントセットはラウンドロビン方式で次のDBRootに書き込まれます。その他の表も異なるDBRootで始まります。このデータの分散は、列ファイルの同時実行読取りの様々な条件を満たすために使用可能なストレージリソースを最大限に活用するために設計されています。

前述の動作は、定義されたエクステントサイズに基づいてデータがロードされると自動的に発生します。実際に、表作成操作に関連する追加のストレージパラメータは存在しません。

最初のスキャン操作のチューニング

I/Oをチューニングするには、バッチプリミティブステップ(BPS)で実行される操作の順序を理解する必要があります。0個以上のフィルタを使用して、特定のBPSに対して1つ以上の列が参照されます。すべての場合で、まず1つの列が読み取られ、この最初の操作をサポートするためにエクステント全体が読み取られます。前のフィルタの選択に応じて、追加の列ブロックが読み取られる場合があります。ただし、追加の列では、個々のブロックアクセスを可能にする行識別子を使用できます(指定されている場合)。

「最初のスキャン」操作には、エクステント内のすべてのブロックが必要です。「追加の列の読取り」は、すべてのブロックのスキャン(マルチブロック読取りを最大活用)または個々のブロックの読取り(一部の状況でのブロックの合計数の最小化)のいずれかを行うことで実現できます。

通常、最初のスキャン操作のチューニングは、2つの相反する目標および主要作業単位(エクステント)に基づいて行われます。目標は次のとおりです。

1. 作業単位(エクステント)に対して並行して処理を行うためにできるかぎり多くのスレッドを使用可能にする。読取りは本質的に非同期であるため、スレッドはI/Oの待機中に停止されません。
2. ストレージ構成を最大限に活用するためにマルチブロック読取りを利用する。

InfiniDBは多くの異なるサーバー構成またはストレージ構成で実行されることが予測されるため、これらのパラメータを調整すると所定の環境のパフォーマンスに影響を及ぼす可能性があります。デフォルト設定によって、様々な構成で高パフォーマンスが得られる必要があります。

この動作を制御するCalpont.xml構成ファイル内のパラメータを次に示します。

```
ColScanReadAheadBlocks
```

デフォルト設定は、512(データのブロック数)です。

使用例:8バイトのデータ型に、スキャン操作をサポートするために読み取られる8192個のデータのブロックが含まれています。一度に128個以上のブロックを読み取るときにスループットが最大になるストレージサブシステムでは、`ColScanReadAheadBlocks`パラメータに128、256または512以上の値を指定すると、ストレージサブシステムの最良のパフォーマンスを実現できます。パラレル読取りの実行に8つのコアを使用可能なサーバーでは、1024、512または256以下の値を指定すると、(最大で)8つすべてのコアで並行して読取りを実行できます。

多くのサーバー構成またはディスク構成では、類似する結果をもたらす有効な設定が複数存在する可能性があります(これらの設定では、次の両方の要件が満たされるため)。

1. 並行して読み取るのに十分なスレッドを使用できる。
2. 帯域幅を最大化するためにマルチブロック読取りを十分に利用する。

追加の列の読取りのチューニング

追加の列の読取りに対する物理I/Oのパフォーマンスは、前の列に適用されたフィルタの特定のカーディナリティまたはフィルタチェーンによって異なります。エクステントから1つの行のみが必要な場合は、どのストレージシステムでも、そのブロックのみを読み取ると時間が短縮されます。いくつかのブロック、ほとんどのブロックまたはすべてのブロックが必要な場合は、一部の個々のブロックが問合せのサポートで参照されない場合でも、マルチブロック読取りを利用すると時間が短縮される可能性があります。

追加の列の読取り操作で使用可能な読取り方法は2つあります。

1. 既知のアドレスに基づいて個々のブロックを読み取る。
2. `ColScanReadAheadBlocks`に設定されている数のブロックを一度に読み取るマルチブロック読取りを使用する。

読取りが必要な場合、これら2つの操作の選択はPM内で行われます。実際には、同じ列に対する以前の読取りから統計が収集され、パラメータと比較されます。

この動作を制御するCalpont.xmlファイル内のパラメータを次に示します。

```
PrefetchThreshold
```

デフォルト設定は、5(問合せで使用されたブロックの割合)です。

デフォルト設定の5は、以前読み取られたブロックの5%を超えるブロックが問合せを満たすために実際に必要だった場合に、マルチブロック読取りが継続して使用されることを示します。問合せを満たすために必要なブロックの割合が5%を下回った場合は、後続の読取りで個々のブロックの検索が使用され始めます。

このパラメータは、行の数ではなく、フィルタで必要なブロックの数に関連していることに注意してください。行の5%を返す述語には、90%を超えるブロックが必要な場合があります。ただし、データ値が高度にクラスタ化されている場合、ブロックの数は5-10%まで下がることがあります。

実際の動作は、多くの項目の複雑な相互関係に関連しています。

1. 問合せで使用される特定のフィルタまたはフィルタチェーン。
2. フィルタで使用される列の値の頻度。
3. ブロック内でのこれらの値の実際の分布。
4. その時点でのデータバッファキャッシュ内のブロック。
5. 個々のブロックを読み取った場合と総ブロックを読み取った場合のストレージシステムの相対コスト。
6. ストレージシステムに対する同時実行問合せの影響。

通常、このパラメータはほとんどの問合せに影響を及ぼしません。これは、多くのInfiniDBの機能および動作に基づいています。

1. 複数の最適化に基づいたI/Oの大幅な削減によって、すべての状況でストレージシステムのボトルネックの発生を防止できます。
2. 多くの問合せでI/O要件が軽減されるため、より多くの問合せをキャッシュによって処理できます。また、グローバルなスケーラブルデータキャッシュを使用できます。
3. 通常、ほとんどの分析的な問合せは、エクステント内の多くのブロックにアクセスし、境界条件に該当しません。

同時実行および問合せ

InfiniDBでは、同時実行の管理は、特定の問合せに割り当てるスレッドの数を調整したり、スレッドの優先順位を割り当てるのではなく、バッチプリミティブ操作のリクエストがUMからPMに発行される割合を管理することによって行われます。

処理の流れは次のパラメータによって制御されます。

`MaxOutstandingRequests`

デフォルト値は5(エクステント)です。

前述のとおり、UMから発行されるバッチプリミティブリクエストは、1エクステント(各列に1つのエクステント)内の行の範囲に対して特定の操作を実行するリクエストです。この処理では、次の一般的なフィードバックループが行われます。

- UMは、`MaxOutstandingRequests`に設定されている数まで、バッチプリミティブを発行します。
- PMは、データブロックを処理し、個々の応答を戻します。
- UMは、すべての未処理バッチプリミティブからのブロック応答を個別に受信し、次のエクステントに対して次のバッチプリミティブを発行するタイミングを決定します。

  たとえば、パラメータで5(エクステント)を設定した場合、5つのエクステントを処理するバッチプリミティブリクエストが発行されます。ブロック応答が戻されると、UMは、未処理リクエストの総数がパラメータの設定に達するように、新しいエクステントに対して1つの追加バッチプリミティブを発行します。戻されたブロックのサイズの合計が1エクステントのサイズに達すると、次のエクステント用に別のバッチプリミティブが発行されます。

このパラメータの設定では、前述のパラメータ設定を使用してどの時点でも5つ相当のエクステントがアクティブに処理されようにすることが目標として定義されます。

事実上、この処理によって、利用可能なすべてのリソースを大規模な問合せで使用でき(他で消費されていない場合)、小規模な問合せを最小限の遅延で実行できます。各列の1つのエクステントに対して個々のバッチプリミティブを実行するのに必要な時間は非常に短く、ディスクから多数の列を読み取る場合にかかる時間は最大で1秒ほどです。

負荷が大きい場合に小規模な問合せに対する遅延を最小限に抑えるには、小さい整数値を設定することをお薦めします。

大規模な問合せを優先させるには、`MaxOutstandingRequests`の値を少し大きくします。完全または完全に近いCPU使用率でPMが動作できる状態にシステムが安定するまで、少しだけ時間がかかる場合があります(1秒ほど)。多くの場合、デフォルト値によってすべてのPMが常に動作している状態に保たれるため、キュー内の追加リクエストによって経過時間が変わることはありません(開始時間は短縮されません)。

通常、単独で実行される大規模な問合せが迅速に開始されて利用可能なすべてのリソースが使用されるように、パフォーマンスモジュールの数を基に`MaxOutstandingRequests`の設定を調整する必要

があります。同時実行のワークロードシナリオでは、複数問合せのサポートによってキューが一杯になるため、このパラメータの設定にかかわらず、通常は完全なシステムの使用率が実現します。

システムのテストおよびベンチマークの結果では、単独で実行される大規模な問合せを迅速に開始できる各スケールアウト構成に対する適切な開始値は次のとおりです。

PMs	MaxOutstandingRequests
1-2	3
3-4	5
5-6	7
7-8	9
9 or more	PM count + 1

InfiniDB の複数結合のチューニング

InfiniDBの結合処理モデルでは、ネステッドループ操作は行われず、かわりにハッシュ結合操作が実行されます。比較的少ない数の行の結合（索引によってサポートされている場合）では、ハッシュ構造を構築する必要がないため、ネステッドループ操作の方が高速になる可能性が高くなります。一方、ハッシュ結合は、数千以上の行を結合する場合に一般的にパフォーマンスが高く、数百万、数十億という行を結合する場合はネステッドループ操作よりもかなり高速に実行されます。

InfiniDBエンジンには、ハッシュ結合操作を構築する多数の方法があります。これらの方法では、結合操作に関連する最大の表のハッシュ構造を作成する必要がないように、操作が優先的に順序付けられます。かわりに、最大の表（ファクト表）をマテリアライズする必要がないように、このファクト表には必要なすべてのフィルタ、結合および集計操作が行われます。ファクト表のマテリアライズを回避するような問合せの構築は、InfiniDBの最適化によって自動的に行われます。

InfiniDBが実行するハッシュ結合は、UMが管理するバッチプリミティブステップの範囲内で実行されます（したがって、PMが実行するバッチプリミティブ操作によって実行されます）。バッチプリミティブステップは問合せの実行計画のステップであり、結合の図ではノードとして表すことができます。実際の実行は、いくつかのパラメータ設定および結合される小さな各表のカーディナリティによってわずかに異なる場合があります。

簡単な 2 つの表の結合の詳細

特定の2つの表に対する結合操作の場合、一方の表が大きい表と判断され、他方が小さい表と判断されます。この判断は、表内のブロック数および述語のカーディナリティの推定値に基づきます。この結合は、2つのバッチプリミティブステップ（BPS）を使用して実行されます。小さい表をスキャンするBPSは、UMにアクセスしてデータを戻すときに、利用可能な任意のフィルタを適用します。これは、BPS（小）と表すことができます。

次の条件付き動作は、大規模な結合の実行に必要なネットワーク通信またはプロセス間通信を最小化する（データの送信コストを最小限に抑える）ために行われます。データがUMに戻されると、データセットのサイズが判断されます。この測定値は、次のパラメータと比較されます。

```
PmMaxMemorySmallSide
```

デフォルト値は64M（サイズはMB単位）で、最大で4Gです。

小さい表の100万程度の行が、大きい表の数十億（あるいは数兆）の行に対して結合される場合には、このデフォルトで十分です。実際のカーディナリティは、結合データ型と、問合せに含まれる小さい表の追加列によって異なります。データセットのサイズが`PmMaxMemorySmallSide`よりも小さい場合は、データがPMに送信されて分散ハッシュマップが作成されます。そうではない場合、UMで作成されます。

UMまたはPMで小さい表のハッシュマップがインスタンス化されると、最大の表に対してBPS（大）が発行されます。小さい表のハッシュマップがPMに送信された場合、結合操作および集計操作は完全分散で行われます。小さい表のハッシュマップがUMにある場合、結合および集計はUMで行われます。どの場

合も、列および表のフィルタは、PM上で、完全にマルチスレッド化および分散された方法で同様に適用されます。

実質的に、2つの表の結合は、2つのバッチプリミティブステップによって実行されます。

BPS(小) -> BPS(大)

データウェアハウスの用語を使用すると、この簡単な問合せは次のように言い換えられます。

BPS(ディメンション) -> BPS(ファクト)

複数結合の詳細

単一のBPSで複数の小さい表と1つの大きい表を結合する場合にも、この結合をサポートするために前述の処理モデルが使用されます。

1つの大きな表が2つの小さい表と結合される場合、操作は次のようになります。

```
BPS(小_1)
          \
BPS(小_2)  ----> BPS(大)
```

これは、大きい表を処理する前に、小さい表の2つのステップで実行されます。この処理モデルは、小さい表のハッシュマップがUMまたはPMのどちらでインスタンス化されたかにかかわらず、同じであることに注意してください。実際に、前述の図のステップは、各オブジェクトのサイズにかかわらず実行できます。実行時の各オブジェクトの正確なカーディナリティやサイズに応じて、両方の結合をPMまたはUMで実行したり、任意の組合せに分割して実行することができます。

実際、この結合処理モデルは、任意の数の結合操作を処理できます。この際、結合操作に含まれる小さい各表に対して1つのBPSが使用され、最大の表に対して1つのBPSが使用されます。

スタースキーマのデータモデルの場合は、この方法で、自己結合を使用しない多数の問合せに対応することができます。

```
BPS(ディメンション_1)     \
BPS(ディメンション_2)      \
BPS(ディメンション_3)        > BPS(ファクト)
   中略                    /
BPS(ディメンション_20)    /
```

これによって、スレッド間の同期を最小限に抑えた状態で、完全にマルチスレッド化および分散(オプション)された処理を実行できます。また、ファクト表から行または集計を戻す処理は、可能なすべてのフィルタが適用されるまで保留されます。

その他の結合の組合せは、連鎖結合演算子および複数結合演算子の様々な組合せを使用して行われます。たとえば、スノーフレークスキーマでは、BPS(ファクト)表用のハッシュマップを構築するための

前提条件である追加の結合操作が発生します。内部結合操作および外部結合操作の両方がこれらの連鎖結合演算子および複数結合演算子でサポートされています。

結合の最適化

InfiniDBオプティマイザは、統計を使用してグローバル結合ツリー内で最大の表を判断し、その表をストリーム表として使用できるように操作を順序付けます(データの転送コストを最小限に抑えます)。グローバルで最大の表によってまだ設定されていない追加のサブツリーでも、大きい表または小さい表のどちらであるかをローカルで判断できます。実際のカーディナリティが類似している場合、表の選択はパフォーマンスに影響しないことがあります。大きい1つのファクト表と小さい複数のディメンション表を含むデータセットの場合、準最適なものが選択される可能性はかなり小さくなります。オプティマイザによって行われた判断が準最適である場合に問合せをチューニングできるようにするには、結合操作内で最大の表を設定するためのヒントを利用できます。

`INFINIDB_ORDERED`ヒントには、`From`句の最初の表はグローバルで最大の表として処理され、可能な場合はその表の結合結果のマテリアライズが除外されるように結合が順序付けられることが示されます。

このヒントは、グローバルで最大の表を結合対象の最後の表として設定します。問合せ内でこのグローバルで最大の表に結合された表により、その最後の操作に含める表(または結合サブツリー)が決定されます(`From`句内の表の順序ではありません)。InfiniDBでは、複数の結合操作が1つのBPSで実行されます。そのため、1つの大きい表とn個の小さい表の結合では、ほとんどの場合、小さい表の結合の順序によって問合せのI/O特性が変わることはありません。

たとえば、このヒントを次のように使用すると、オプティマイザは実際のカーディナリティに関係なく、region表を結合ツリー内で最大の表として処理します。

```
select /*! INFINIDB_ORDERED */ r_regionkey
from region r, customer c, nation n
where r.r_regionkey = n.n_regionkey
 and n.n_nationkey = c.c_nationkey;
```

ネステッドループ操作は行われないため、この場合も、ヒントによって駆動表が指定されたり、1つずつ結合する表の順序が決定されることはありません。これらの概念は、少なくとも従来の使用方法ではInfiniDB内に存在しません。

主要なチューニングパラメータ:PmMaxMemorySmallSide

`PmMaxMemorySmallSide`パラメータによって、PMに送信される小さい表の単一ハッシュマップのサイズに対する上限が設定され(最大4G)、結合を完全分散で実行するかどうかが決定されます。

単一サーバーのインストールの一般的なチューニングガイドライン

`PmMaxMemorySmallSide`は、小さい表の結合について予測できる最大サイズをサポートできる大きさに設定します。ただし、利用可能なメモリーを上限とします。単一サーバー内でも、データの送信コストを考慮する必要があり、先に(PMで)フィルタを実行すると、サーバー内部のデータ送信コストが削減されます。

複数 PM のインストールの一般的なチューニングガイドライン

複数PM構成の`PmMaxMemorySmallSide`のチューニングは、サーバーで利用可能なメモリーの量、データバッファキャッシュ用に必要なメモリーの量、同時に発生することが予測される同時結合操作の数、およびこれらの結合操作のサイズに関係します。

次のようにチューニングすることをお薦めします。

> (同時実行されるPMの小さい表のハッシュマップの数)*(各PMのハッシュマップの平均サイズ)
> 前述の値が次より小さくなる必要があります
> (サーバーのメモリーの合計)-(データバッファキャッシュのサイズ)

このパラメータを設定することによってメモリーが明示的に消費されるわけではないことに注意してください。このパラメータでは、PMでインスタンス化できるハッシュマップの最大サイズのみが制限されます。

16GBのメモリーを備えたサーバーで、同時実行はそれほど多くなく、2番目に大きい表の結合カーディナリティが最大で1000万である場合は、`PmMaxMemorySmallSide`を512Mに設定して、データバッファキャッシュを14GBに設定すると適切です。

同時実行がかなり多い場合でも、各PMのハッシュマップの平均サイズによっては、512Mの値で有効な場合があります。ただし、サーバーのメモリー利用率が100%に近づいた場合、またはこの値を越えてスワップが開始された場合は、`PmMaxMemorySmallSide`の設定を小さくすると、サーバーのメモリーが削減される可能性があります。

メモリー管理

主要なチューニングパラメータ：NumBlocksPct および TotalUmMemory

各パフォーマンスモジュール処理内のデータバッファキャッシュ専用のメモリーの量は、次のCalpont.xmlパラメータによって設定します。

PMのメモリーパラメータ：
`NumBlocksPct`

デフォルトは提供形態によって異なります（単一サーバーのデフォルトは50で、複数サーバーのインストールのデフォルトは70になります）。値が86の場合、データバッファキャッシュは、サーバーで利用可能なメモリーの86%まで消費できます。

ユーザーモジュールとパフォーマンスモジュールが同じサーバーで実行される単一サーバー構成の場合は、NumBlocksPctの設定を低くする必要があります。これは、ユーザーモジュールが、一時データセットを管理するために一時領域を著しく必要とする場合があるためです。単一サーバーのインストールの場合、`NumBlocksPct`と`TotalUmMemory`で、サーバーで利用可能な合計メモリーの75%を超えないようにすることをお薦めします。

`TotalUmMemory`パラメータは、ユーザーモジュールでの結合、集計および集合操作の中間結果の管理に利用可能なメモリーの最大量を制限します。これは、メモリーを専用に確保するではなく、単一結合のハッシュマップのサポートで消費可能なメモリーの最大量を制限します。通常は、必要に応じてこの設定を引き上げることで、何億もの行が含まれることのある小さい表の臨時ハッシュマップを最小限の影響で使用できます。

PM処理とUM処理を異なるサーバーに分離すると、小さい表の平均の結合カーディナリティが100Kより小さい場合、キャッシュをPMの合計メモリーの95%に設定しても差し支えありません。この場合、PMのメモリー利用率は、様々な同時実行のシナリオに対して95-97%の間で変動します。

InfiniDBでは、大きい表の数十億または数兆の行を小さい表の任意の数のハッシュマップに結合することがサポートされており、大きい表のサイズは、どのパラメータ設定でもまったく制限されないことに注意してください。`TotalUmMemory`パラメータは小さい表のマップにのみ適用されます。

スケーラビリティ

スケーラビリティ：データのサイズとパフォーマンス

特定のシステムが構成され、問題となる問合せが最適化されて効率的に実行されたら、パフォーマンスモジュールの規模を調整することによってパフォーマンスを向上することができます。また、システムの規模を調整することで、大容量のデータを一貫したパフォーマンスで分析できる場合があります。

スケーラビリティ：ユーザーモジュール

ユーザーモジュール上でCPU使用率が増加している場合、その負荷を分散するためにユーザーモジュールを追加できます。InfiniDBの目標の1つはできるかぎり多くの作業を分散することであるため、多くの問合せがPM上で99%のCPUサイクルを実現できます。

ツールおよびユーティリティのチューニング

問合せのサマリー統計：calgetstats

以前の操作のサマリー結果を表示する問合せの後に、`select calgetstats()`コマンドを実行できます。次の問合せの例では、3つのフィルタ（そのうち2つは結合で表されている）を使用して、ファクト表に59億9000万行が含まれているスタースキーマベンチマークのデータセットに対して4つの表の結合が実行されています。最後の集計は約2000万行に対して行われ、この例では8PM構成が使用されています。

結果の詳細は、出力例のとおりです。

```
     select  d_year, lo_tax, p_size, s_region,  count(*)
       from  dateinfo, part, supplier, lineorder
      where  s_suppkey = lo_suppkey
        and  d_datekey = lo_orderdate
        and  p_partkey = lo_partkey
        and  lo_orderdate between 19980101 and 19981231
        and  s_nation = 'BRAZIL'
        and  p_size <> 23
     group by  1,2,3,4
     order by  1,2,3,4;

+--------+--------+--------+----------+----------+
| d_year | lo_tax | p_size | s_region | count(*) |
+--------+--------+--------+----------+----------+
|   1998 |   0.00 |      1 | AMERICA  |    47607 |
|   1998 |   0.00 |      2 | AMERICA  |    47194 |
... results abbreviated ...
|   1998 |   8.00 |     48 | AMERICA  |    47846 |
|   1998 |   8.00 |     49 | AMERICA  |    47394 |
|   1998 |   8.00 |     50 | AMERICA  |    47030 |
+--------+--------+--------+----------+----------+
441 rows in set (3.66 sec)

mysql> select calgetstats();
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------+
| calgettats()
|
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------+
| Query Stats: MaxMemPct-4; NumTempFiles-0; TempFileSpace-0MB; ApproxPhyI/O-0;
CacheI/O-1363254; BlocksTouched-1363254; PartitionBlocksEliminated-2637824;
MsgBytesIn-811MB; MsgBytesOut-0MB; Mode-Distributed| 1256555629 961957 |
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------+
1 row in set (0.02 sec)
```

`calgetstats()`関数の出力は次のとおりです。

- MaxMemPct-4:このフィールドには、ユーザーモジュール(UM)の結合、グループ化、集計、重複行の削除などの操作を行うために使用されているUMのメモリー利用率が示されます。
- NumTempFiles-0:このフィールドには、ユーザーモジュール(UM)の結合、グループ化、集計、重複行の削除などの操作を行うために使用されているUMの一時ファイル使用数が示されます。
- TempFileSpace-0MB:このフィールドには、ユーザーモジュール(UM)の結合、グループ化、集計、重複行の削除などの操作を行うために使用されているUMの一時ファイル使用サイズが示されます。
- ApproxPhyI/O-0:このフィールドには、問合せに対するストレージからの読取りが示されます。状況によっては、実際とは少し異なる場合があります(通常は0.1%未満)。
- CacheI/O-1363254:このフィールドには、リクエストされた個別の物理I/O読取りの数が削減されている、問合せで必要なブロックアクセスが示されます。
- BlocksTouched-1363254:このフィールドには、問合せで必要なブロックアクセスが示されます。
- PartitionBlocksEliminated-2637824:このフィールドには、エクステントマップの最小値または最大値に基づいて発生したパーティションの除外によって排除されたブロックが示されます。フィルタが適用されるまで、列ストレージまたは遅延している列の読取りでのI/Oの削減はレポートされないことに注意してください。
- MsgBytesIn-811MB:データ移動を処理するプロセスの測定単位。
- MsgBytesOut-0MB:データ移動を処理するプロセスの測定単位。
- Mode-Distributed:問合せ結合処理がInfiniDB内で処理されたか、従来のMySQLの結合機能で処理されたかを示すインジケータ。大規模なデータに対して最高のパフォーマンスを実現するには、この処理を分散することをお薦めします。

問合せの詳細統計:calgettrace

InfiniDBのSQLトレース機能を有効にすることでさらに詳細なレベルでの追加情報を表示できます。
SQLトレースの使用手順は次のとおりです。

1. コマンド`select calsettrace(1);`を発行して、新しいトレースを有効にします。
2. 問合せを実行します。
3. コマンド`select calgettrace();`を発行して、トレースの結果を表示します。

たとえば、新しいトレースを有効にした後、次のような問合せを実行して分析できます。

```
  select  d_year, lo_tax, p_size, s_region,  count(*)
    from  dateinfo, part, supplier, lineorder
   where  s_suppkey = lo_suppkey
     and  d_datekey = lo_orderdate
     and  p_partkey = lo_partkey
     and  lo_orderdate between 19980101 and 19981231
     and  s_nation = 'BRAZIL'
     and  p_size <> 23
group by  1,2,3,4
order by  1,2,3,4;

mysql> select calgettrace();
+-------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| calgettrace()
|
```

```
+-------------------------------------------------------------------------------------------
-----------------------------------------------------------+
|
Desc Mode Table       TableOID ReferencedOIDs       PIO    LIO    PBE    Elapsed Rows
BPS  PM   part        3513     (3521,3514)          1370   1372   0      0.281   1371884
BPS  PM   dateinfo    3558     (3559,3563)          12     10     0      0.126   2556
DSS  PM   supplier    3528     (3539)               0      1      -      0.000   1
BPS  PM   supplier    3528     (3533,3529,3540)     2449   3551   0      0.263   40078
HJS  PM   supplier    3528     -                    -      -      -      -----   -
                     -supplier
BPS  PM   lineorder    3493     (3499,3497,3498,3508) 132060 131298 266240 5.553   1489347
HJS  PM   lineorder   3493     -                    -      -      -      -----   -
                     -dateinfo
HJS  PM   lineorder   3493     -                    -      -      -      -----   -
                     -part
HJS  PM   lineorder   3493     -                    -      -      -      -----   -
                     -supplier
TAS  UM   -           -        -                    -      -      -      5.374   441
TAS  UM   -           -        -                    -      -      -      5.374   441
+-------------------------------------------------------------------------------------------
-----------------------------------------------------------+
1 row in set (0.02 sec)
```

`calgettrace()`の出力には次のヘッダーが含まれています。

- Desc：実行された操作。
- Mode：UM内またはPM内のいずれで実行されたか。
- Table：列をスキャンまたは投影できる表。
- TableOID：スキャンされた表のオブジェクトID。
- ReferencedOIDs：問合せで必要な列のオブジェクトID。
- PIO：問合せで実行された物理I/O（ストレージからの読取り）。
- LIO：問合せで実行された論理I/O。ブロックアクセスとも呼ばれます。
- PBE：除外されたパーティションブロック（PBE）では、エクステントマップの最小値または最大値によって除外されたブロックが識別されます。
- Elapsed：表示されているステップの経過時間。
- Rows：返された中間行

トレース出力には、操作の順序の他、ブロックアクセス、経過時間および行カーディナリティの操作コストがレポートされます。最初のフィルタはsupplier表に対して適用されます。

```
DSS  PM   supplier    3528     (3539)          0      1      -      0.000   1
```

= 'Brazil'のフィルタは、複数の行で1つの文字列値を共有できるディクショナリ構造内に格納されている可変長フィールドを参照します。これは、ディクショナリシグネチャステップ（DSS）という個別のステップで行われます。

```
BPS  PM   supplier  3528  (3533,3529,3540)  2449   3551   0  0.263   40078
```

このBPSは、後続の結合でsupplierから他の列を読み取ることを示します。

```
HJS  PM   lineorder 3493  -                -      -      -  -----   -
          -supplier
```

このハッシュ結合ステップ(HJS)は、supplierの投影とDSSフィルタを関連付けます。これは、実際には前のBPSの一部であり、時間には常に0(ゼロ)がレポートされることに注意してください。Modeフィールドは PM結合がここで実行されたことを示しています。

```
BPS  PM   dateinfo  3558  (3559,3563)       12     10     0  0.126   2556
```

このBPSは、後続の結合でdateinfo表の列を読み取ることを示します。

```
BPS  PM   part      3513  (3521,3514)        1370  1372   0  0.281   1371884
```

このBPSは、後続の結合でpart表の列を読み取ることを示します。

```
BPS  PM   lineorder  3493 (3499,3497,3498,3508) 132060 131298 266240 5.553   1489347
```

このBPSは、lineorderからスキャンおよび投影を行い、supplier、dateinfoおよびpartに対して結合操作を実行することを示します。すべてのフィルタおよび結合が終了した後の出力カーディナリティは 21,340,320行です。

```
HJS  PM   lineorder 3493  -                -      -      -  -----   -
          -dateinfo
```

HJSは、前のBPS内で発生するハッシュ結合操作のインジケータです。Modeフィールドの値PMまたは UMは、UMまたはPMにハッシュマップが作成されたかどうかを示します。ハッシュ結合操作は実際には前のBPS内で発生するため、ここにレポートされる経過時間は常に0(ゼロ)になります。

```
TAS  UM   -         -    -                -      -      - 5.374   441
```

タプル集計ステップ(TAS)では、UMが入力BPSから最初の中間集計結果を受信してから必要な集計を完了するまでの経過時間がレポートされます。ほとんどの集計シナリオで、この時間は前のBPS操作と密接に対応します。この操作では、レポートされるModeは常にUMとなりますが、基本機能は常に2 フェーズ操作(まずPMでローカル集計が実行され、UMで最終集計が実行される)で行われます。

多くの操作が並行して行われるため、個々の経過時間の合計が問合せの経過時間よりも長くなることに注意してください。前述の例では、BPSステップに5.553秒、タプル集計ステップ(TAS)に5.374秒かかっていますが、実際にはパイプライン操作として行われており、BPSは部分的に集計されたデータをTAS に入力します。したがって、BPS(lineorder)とその後のTASの両方を実行する場合の経過時間はレポートされる5.553秒により近くなります。

キャッシュのフラッシュ:calflushcache

InfiniDBでは、すべてのPM間のデータバッファキャッシュからデータをフラッシュする物理I/Oの簡単なテストを実行できる開発ユーティリティまたはテストユーティリティが提供されています。これは、次のコマンドを発行することで実行されます。

```
select calflushcache();
```

これは、開発ユーティリティまたはテストユーティリティとして使用することのみを目的としています。キャッシュのフラッシュおよびPIOの増加によるメリットは報告されていません。それどころか、通常は問合せの実行時間が長くなります。

パラメータの読取りまたは変更：configxml.sh

getconfigまたはsetconfigによってCalpont.xmlパラメータファイル内の値の読取りまたは設定を行うことができるユーティリティが提供されています。

使用方法：`/usr/local/Calpont/bin/configxml.sh {setconfig|getconfig} section variable set-value`

このマニュアル内で説明されているパラメータを更新する構文の例：

```
/usr/local/Calpont/bin/configxml.sh setconfig HashJoin PmMaxMemorySmallSide 640M
/usr/local/Calpont/bin/configxml.sh setconfig JobList MaxOutstandingRequests 7
/usr/local/Calpont/bin/configxml.sh setconfig DBBC NumBlocksPct 90
```

エクステントマップの表示：editem

`editem`というユーティリティを使用すると、指定した列にデータが移入されているかどうかを判断できます。このユーティリティではオブジェクトIDが必要です。オブジェクトIDは、トレース出力またはシステムカタログのいずれかから取得できます。

警告：`editem`ユーティリティは、検査用の値の表示には安全に使用できます。データが破損する恐れがあるため、editemを他の目的に使用する場合は、Calpontサポートの指示に従って使用するか、隔離されたテスト環境でのみ使用してください。

システムカタログからの選択の例：

```
select columnname, objectid from calpontsys.syscolumn where tablename =
'lineorder' and columnname = 'lo_commitdate';
```

`editem`のヘルプでは、提供されている他のすべてのユーティリティと同様に、-hフラグを使用することで（例：`/usr/local/Calpont/bin/editem -h`）、使用可能なコマンドのリストが提供されます。

`editem -o <objectid>`コマンドによって、指定した列の最小値および最大値が表示されます。最小値が存在しない場合はそのデータ型に対して可能な最大値で表され、最大値が存在しない場合はそのデータ型に対して可能な最小値で表されることに注意してください。

列データストレージの相違点

列ストレージのメリットとトレードオフを次に示します。トレードオフを回避したり、列ストレージ機能を最大限に活用する方法についても説明します。

- **挿入のトレードオフ**:列ストレージでは、個々の行を挿入する際のパフォーマンスプロファイルが異なります。任意の行ベースのシステムに1行挿入する場合、アクセスする可能性があるのは1つの表ブロックおよび複数の索引ブロックです。列ストレージのDBMSに挿入する場合は、列ごとに1つ以上のブロックにアクセスしますが、索引ブロックに対する追加コストがありません。このコストは、バルクロード(`cpimport`)、LOAD DATA INFILEまたはバルク挿入操作では変わることに注意してください。何千行も挿入する場合、関連するブロックアクセスは数千行に集中し、行DBMSの表および索引の組合せと比較すると、列ストレージは数千を超えるとより効率的になります。
    - **InfiniDBの方法**:`cpimport`を使用します。InfiniDBの`cpimport`バルクロードユーティリティを使用すると、列ストレージ(索引を使用しない)での挿入に必要なブロックは、行ベースのDBMSの場合より少なくなります。また、索引を使用しないと、通常、より高速のロードパフォーマンスを実現でき、大規模なデータのロード時間の一貫性が向上します。InfiniDBでは、表が空でも、表に100億行が含まれていても、一連の行のロードは同じ操作です。
- **削除のトレードオフ**:列ストレージでは、個々の行の削除のパフォーマンスプロファイルが異なります。任意の行ベースのシステムから1行削除する場合、アクセスする可能性があるのは1つの表ブロックおよび複数の索引ブロックです。列ストレージのDBMSから削除する場合は、列ごとに1つ以上のブロックにアクセスしますが、索引ブロックに対する追加コストがありません。特定の範囲の行から数千行を削除する場合は、関連するブロックアクセスが集中し、行DBMSの表および索引の組合せと比較すると、列ストレージはより効率的になります。

  削除操作のパフォーマンスは、どのDBMSでも、削除する行がブロック間で分散されている状態によって異なる場合があります。たとえば、8Kブロックごとに100行を格納する行ベースのDBMSから1000行を削除する場合は、10個(最適な分散)から1000個(最悪の分散)の範囲のブロックにアクセスする可能性があります。列ストレージでは、10列の表(各列がブロックごとに1024-8192行を格納する)の場合、最良のシナリオは10個のブロック(最適な分散)ですが、最悪の場合は10,000個のブロック(最悪のシナリオ)です。
    - **InfiniDBの方法**:バッチで削除します。たとえば、100万行を指す100個のキーを含むIN句に基づいた行の削除は、100個の個別の削除文より大幅に短時間で実行されます。
- **更新のメリット**:列ストレージの更新では、ほとんどのシナリオで大きなメリットが得られます。1000行の行DBMSの更新の最良のシナリオは、100個のブロック(列ごとに100行と想定)です。列DBMSの最良のシナリオは1個のブロックです。索引も更新する必要がある場合は、パフォーマンス上のメリットが増えます。
    - **InfiniDBの方法**:すでに最適化されているため、通常は更新が非常に高速に実行されます。

データのロード速度およびリアルタイムに近いロード

データのロード速度は、バッチ処理の状況、基礎となるストレージ機能、表定義、データ型および値によって異なります。ただし、一般的な目安として、`cpimport`バルクロードユーティリティのロード速度は、1秒当たり数十万から数百万行になる場合があります。LOAD DATA INFILEのロード速度は、1秒当たり数千行の場合があります。個々の挿入のロード速度はさらに遅く、1秒当たり数十行です。どの場合も、データは読取り一貫性動作が確保されるモデルで使用可能になります。

`cpimport`バルクロード機能を使用すると、一度に1000行、数百万あるいはそれ以上のデータをロードできます。一度に10万行以上をロードするときに速度がピークになるインポートでは、スタートアップ時間に約1秒程度かかります。基本的な操作は変わらないため、このロード速度は以降のロードでは一貫しています。

どの方法でデータをロードする場合も、行の挿入によってエクステントが表に追加される可能性があります。この際、エクステントがストレージ内で全体に連続して配置されるようにするため、通常は5-20秒の遅延が発生します。これによって、マルチブロック読取りのメリットを最大化し、以降の選択操作で最大限のパフォーマンスが確保されます。

処理の優先順位付け

デフォルトでは、インポート処理と問合せ処理の優先順位は同じです。この優先順位は、InfiniDB構成ファイルCalpont.XMLにエントリを追加することによって変更できます(Linuxのみ)。これは、同時実行の問合せとインポートが定期的に実行され、結果としてインポート速度が低下する場合に役立つことがあります。このエントリは次のとおりです。

```
<ExeMgr1>
   <Priority>###</Priority>
</ExeMgr1>

<PrimitiveServers>
   <Priority>###</Priority>
</PrimitiveServers>

<WriteEngine>
   <Priority>###</Priority>
</WriteEngine>
```

###の値は次のとおりです。

- 1-40:1は最も低い優先順位で、40は最も高い優先順位です。前述の3つのプロセスのデフォルトの優先順位は21です。

ExeMgr1およびPrimitiveServersの優先順位を変更すると問合せの優先順位が変更され、WriteEngineの優先順位を変更するとインポートの優先順位が変更されます。

InfiniDB のパフォーマンスの目安

熟練したチューニング担当者が行ベースのDBMSシステムでの経験に基づいて適切と感じる多数の目安があります。InfiniDBではこの従来の目安のいくつかは大幅に変わっており、実際、より適切にデータウェアハウスの複雑さに対処できる多数の新しい方法を示している場合があります。

- 5%未満のデータを問い合せるときには索引が役立ちます。この目安は、表をスキャンしない(全表スキャンを行わない)列ストレージでは変わります。たとえば、類似する20列を含む表の場合、1列をスキャンするコストが表をスキャンするコストの5%になるため、目安は0.25%未満のデータとなります。索引の有用性の低下、ランダムなI/O動作および多大な維持コストを理由として、現時点ではInfiniDBで索引は実装されていません。
    - InfiniDBの新しいパラダイム:I/Oがより効率的であるマルチスレッド操作や分散操作による処理が利用されます。

- 大きい表をモデリングするときは、アクセス頻度が低い列を含めないでください。行ベースのDBMSシステムの場合、表の一部として作成される追加の列によって、多数の行を含む(また、Covering Indexまたはデータのその他の複製やマテリアライズで処理できない)すべての問合せの速度が低下する可能性があります。
    - InfiniDBの新しいパラダイム:問合せで参照されない列は無視されるため、追加のストレージおよび負荷に対するコストのみで、不定期の分析に追加のデータを使用できます。

- 列のデータ型はそれほど重要ではありません。多くの行ベースのDBMSシステム(特に、すべての列を可変サイズのデータ型に格納するシステム)では、データ型に基づくパフォーマンスの違いがない場合があります。データに単一の文字値のみが含まれる場合は、行ベースのDBMSのchar(2)をchar(1)に変更すると、索引なしの列をスキャンするコストが変わる可能性がありますが、その程度は1%以下です。列ストレージシステムの場合、列のスキャンで必要なブロックは半分であるため、効率が向上して列ごとの検索が高速化されます。
    - InfiniDBの新しいパラダイム:索引をチューニングするのではなく、列のデータ型をチューニングして検索を高速化できます。

- 冗長データに利点はありません。冗長データによって、表にアクセスするすべての問合せのコストが増えるだけでなく、更新または一貫性のロジックに関連する問題が必ず発生します。ただし、データのロードに一度だけ書込み可能な方式を使用して実装されるデータウェアハウスのシナリオでは、更新または一貫性の問題は発生しない場合があります。行ベースのストレージでは、表にアクセスするすべての問合せに対して追加のI/Oコストが依然として必要となります。
    - InfiniDBの新しいパラダイム:追加の列によってデータへの新しいアクセスパスが提供される場合があります(追加の列がフィールドの主要な部分か、複数のフィールドの連結であるかは関係ありません)。

- 有用なキャッシュにするには、キャッシュをファクト表（またはアクティブなパーティション）より大きくする必要があります。行ベースの独自のDBMSのほとんどでは、2回目のスキャンがキャッシュによって処理されるようにするには、全表スキャンまたは表内のアクティブなパーティションの全体スキャンがメモリーに完全に収まる必要があります。
    - InfiniDBの新しいパラダイム：列ストレージでは、問合せで参照されない列が削除され、列へのアクセスが個別に行われるため、キャッシュのサイズが表のサイズのほんの一部にすぎない場合でもキャッシュのメリットが得られます。

- ディスクからの読取りは、キャッシュからの読取りに比べて20-40倍低速です。実際の比率は多数の要因によって異なりますが、通常、ディスクから読み取るときはパフォーマンスが大幅に低下します。20-40倍の比率は、問合せコストの95%-97.5%がディスクからの読取りに直接関連することを示します。
    - InfiniDBの新しいパラダイム：キャッシュにより2-10倍改善します。InfiniDBのI/Oの効率性（列ストレージ、パーティションブロックの除外、フィルタ後までのI/Oの遅延）により、ほとんどの問合せの絶対I/Oコストが大幅に減ります。結果として、キャッシュにより相対コストが2-10倍速くなります。

- パーティション化はデータウェアハウスのパフォーマンスにとって重要です。これは、実際に今も当てはまります。従来の独自のDBMSシステムでは、DBAは1つまたは2つのレベルのパーティション化方法を宣言して、この1または2列に対して適切なフィルタを使用して問合せのブロックアクセスを減らすことができます。
    - InfiniDBの新しいパラダイム：列ストレージに基づいて、自動パーティション化を使用できます。ロード方法に基づいて、データの昇順化やその他のクラスタ化が行われているすべての列に対して自動パーティション化を使用できます。これは1または2列に限定されず、適切な特性を持つすべての列に使用できます。
    - InfiniDBパーティションは必ずしも実行する必要はありません。データがランダムにロードされる場合は、パーティションをなくすことによるメリットはない可能性があります。

- 結合には適切な索引が必要です。どのネステッドループ結合操作でも、ループ内の検索が索引によってサポートされると、パフォーマンスが向上します。多くの場合、索引を使用しないと結合のパフォーマンスは大幅に低下します。また、ネステッドループ操作を反転した場合は、別の索引が必要になります。
    - InfiniDBの新しいパラダイム：ハッシュ結合機能を利用して索引の必要性をなくし、ネステッドループ処理に関連する大量の行単位処理のコストをなくします。

- 条件付きのパフォーマンス期待値。適切にチューニングし、チューニングによって定義されている境界内に問合せが存在する場合は、行ベースのDBMSシステムをこれらの問合せ用にチューニングできます。ただし、索引を使用しない、または明示的に宣言された1つまたは2つのパーティション列によって解決されないその他の問合せのパフォーマンスは非常に低い場合があり、パフォーマンスの低下が100倍になることもあります。索引が完全にキャッシュされる場合と、ランダムなアクセスを処理するために何度も再ロードされる場合とでは、索引のパフォーマンス特性が大きく異なります。結合の順序によってネステッドループ操作のコストが大幅に変わる可能性があります。
    - InfiniDBの新しいパラダイム：より一貫したパフォーマンスが実現します。10億のレコードに対して列が1、2、4または8バイトの表の場合はスキャン率にいくらかの違いがありますが、桁違いになることはありません。結合の順序は、データの送信コストを最小限にするように自動的に処理されます。

- すべての列が必要なわけではない場合、「Select *」に利点はありません。
    - InfiniDBの従来どおりのパラダイム：すべての列が必要なわけではない場合、従来どおり利点はありません。