SQL Anywhereの パフォーマン関連自動機能

Jason
Hinsperger
Product
Manager SAP

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

SQL Anywhere の設計ゴール

容易な管理

　　包括的なツール

out-of-the-box （箱から出した状態ですぐ発揮される）パフォーマンス

「埋め込み可能」な機能 ⇨ 自動チューニング

　　多くの環境ではDB管理者がいない

クロスプラットフォームのサポート

相互運用性

➔自律的なデータベース管理への総合的なアプローチ

自律的なデータベース管理

自動管理/自動設定

自動チューニング/自動適応

自動ヒーリング

- 問題のモニタリング、修正・アドバイス

自動プロテクション

管理の容易性

- ゴール: ゼロ (マニュアル) 管理

自動化の設計

管理ツール

- インデックスコンサルタント、アプリケーションプロファイラー …

自己管理はなぜ重要か?

複雑性

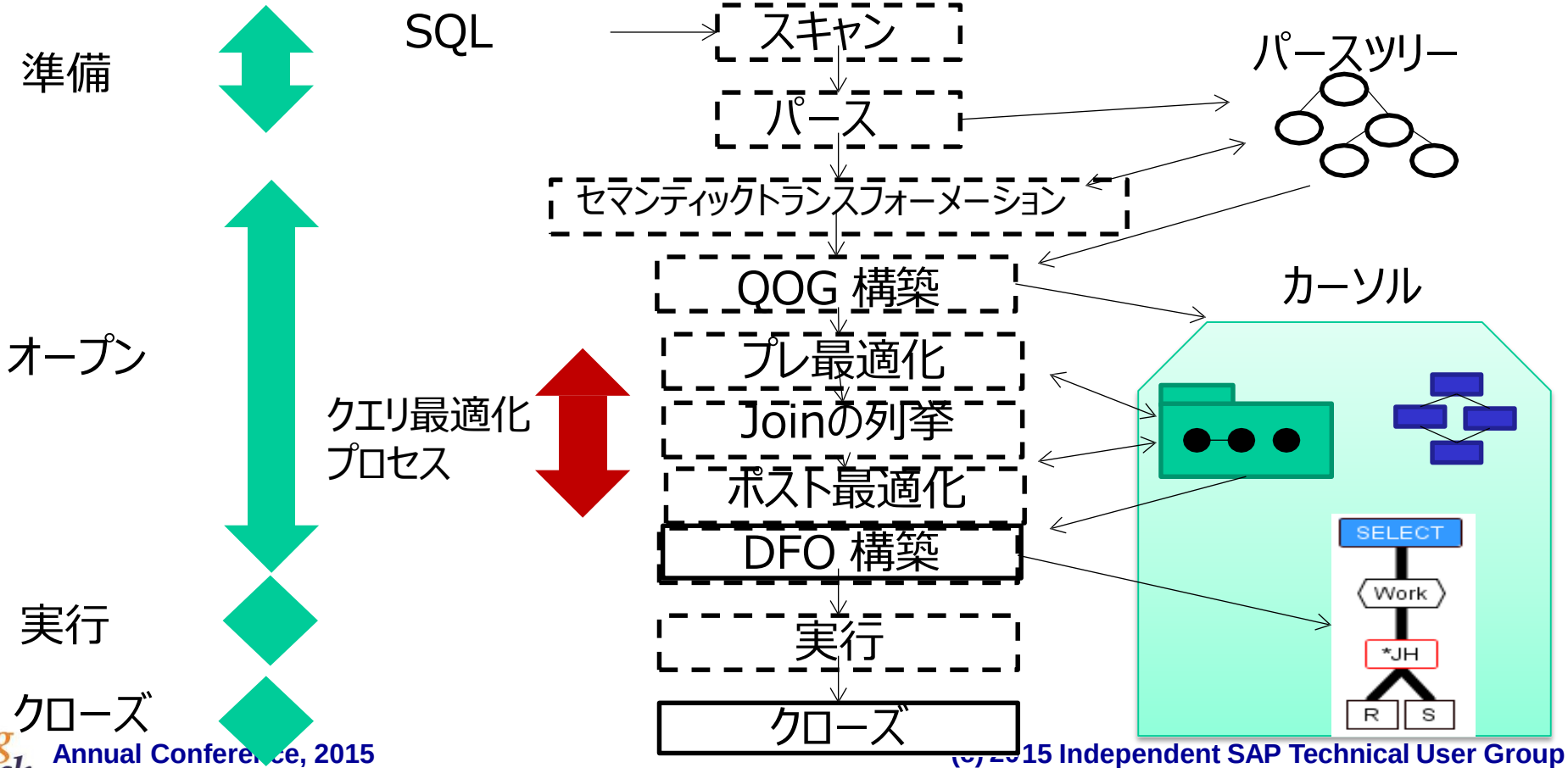
アプリケーション開発は、より複雑になってきている。ORMツールキット、データベースレプリカ間のシンクロナイゼーション、分散コンピューティング、などの新しい開発パラディアムが起こっている。
データベースは、ITでは今やユビキタスであり、多岐にわたる難しい問題を解決している。しかし、ほとんどの企業では、様々な異なるDBMSを所有して管理し続けており、管理コストの増加につながっている。

ユビキタス性は、いくつかの方法でスケールをもたらす。
TCO をコンスタントにキープするためには、各開発者の生産性を向上させる必要がある。

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

SQL Anywhere におけるクエリ処理



SQL Anywhere のパフォーマンス

SQL Anywhere は、ほとんどチューニングなしにパフォーマンスを発揮するよう設計されている

- 多くの自動チューニングと自動管理機能は以下に適用するよう設計されている
  - 自動管理バッファープール: サイズとコンテンツ
  - マルチプログラミングレベルの動的チューニング
  - 自動統計情報収集、モニタリング、ヒーリング
  - 自動チューニングクエリ最適化
    - シンプルな文のためのクエリ最適化バイパス
    - イントラクエリ並列処理
    - 適応型クエリの実行
    - 特定のオペレーションのおけるIO インテリジェンス
  - スタートアップ時のキャッシュウォーミングと定常状態

SQL Anywhere パフォーマンス

しかし…

自動チューニングと自動管理の機能は、以下に適応するように設計されている

- ハードウェア – CPU, I/O, マシンのメモリ
- リクエストされるクエリ
- アプリケーションロジックと同時接続の属性

SQL Anywhere は、異なるディプロイメント環境に適応

- しかしながら: 特定のアプリケーションによっては、パフォーマンスが発揮されない可能性もある
  - 例. 低メモリ実行戦略

→ アプリケーションとデータベースのインタラクションのパフォーマンスを向上させるための多くのことは、**開発時**に行うことができる

- 容量計画、パフォーマンス分析と向上、スケーラビリティテスト

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - データの読み込み - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

SQL Anywhereの速度はどの程度なのか?

パフォーマンスに関する典型的な会話:

お客様:「で、SQL Anywhereはどれくらい速いの?」

SQLA:「あの…お客様のデータベース設計や、同時ユーザー数、ハードウェア、サーバーキャッシュコテンツ、などの様々な要素によるので…」

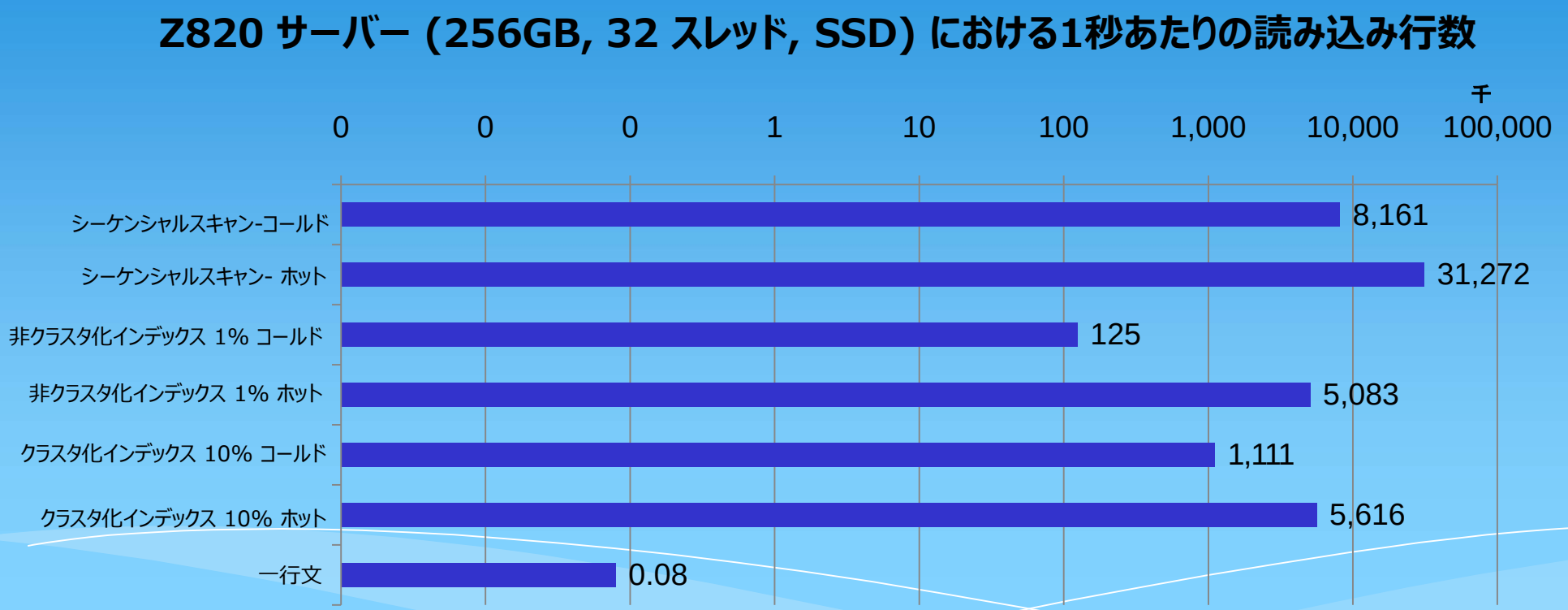
お客様:「いや、それらが重要だということはわかってる。サーバーが一秒間に何行フェッチできるかだけでも教えてくれないか…」

SQLA :「あの…私どものテストでは、1秒間に80から3000万行フェッチできます」

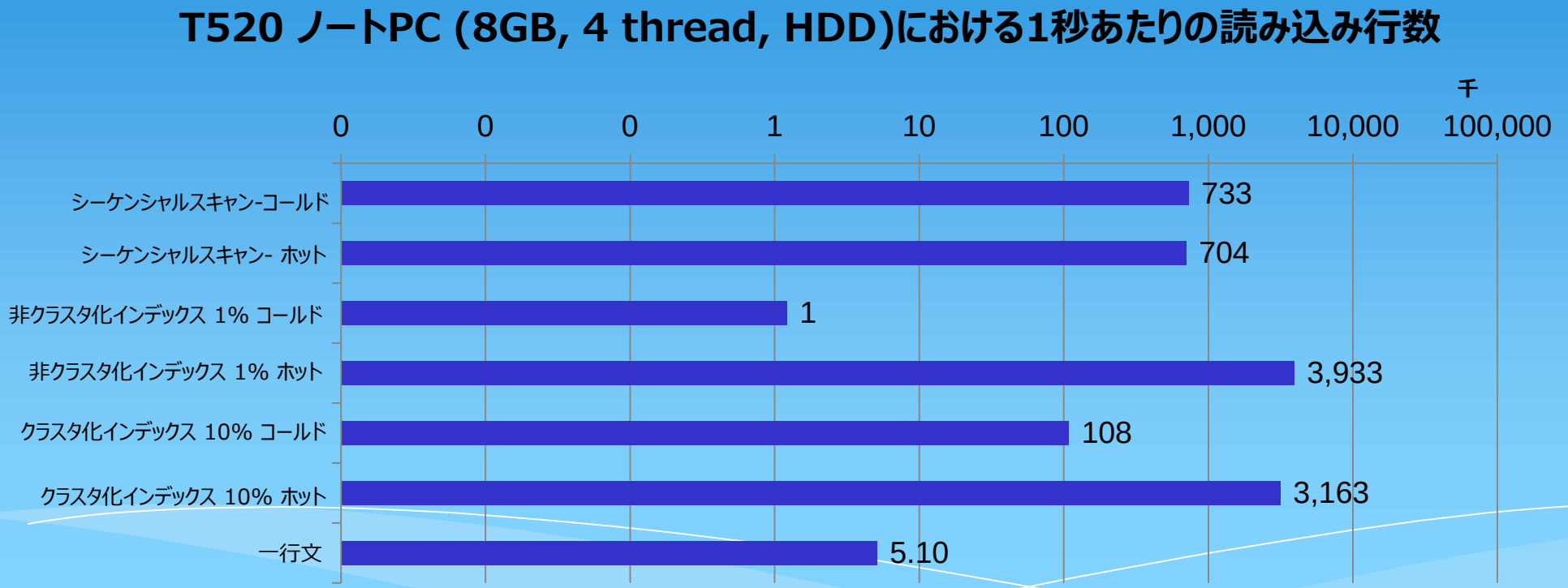
お客様:「は? それはおかしい。うちのアプリケーションは1秒間に3000万行レベルのフェッチはできない。何かバグがあるに違いない。直してくれ」

SQLA :「ですからあの…いろいろな条件に依存するので…」

1秒間に何行読み込み可能なのか?



1秒間に何行読み込み可能なのか? (続)

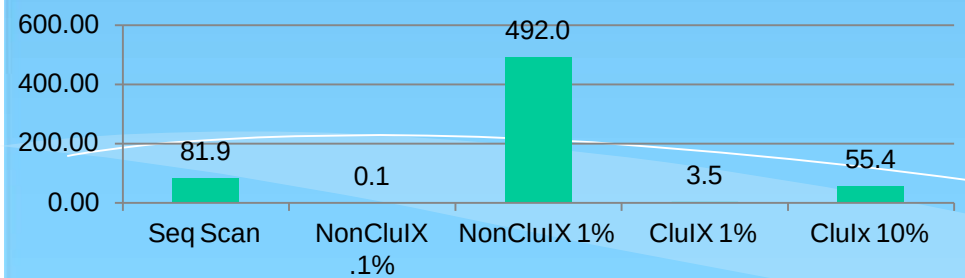


2つのホスト上のコールドキャッシュ パフォーマンス

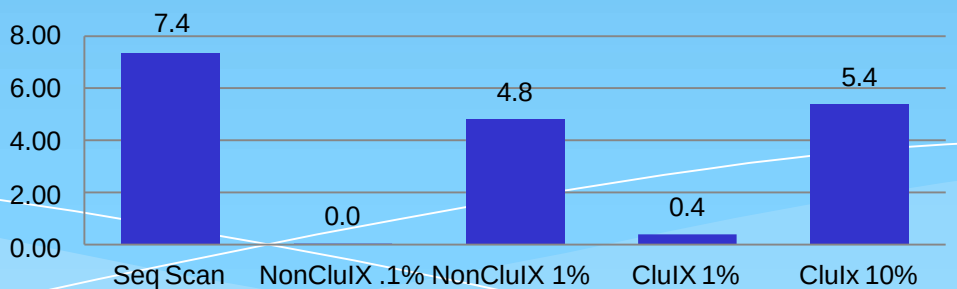
コールドキャッシュのパフォーマンスはI/Oで決まる
HDDでは、シーケンシャルの方が高速
インデックスのクラスタリングは、非常に重要
バッファサイズは再読み込みされるページ数に影響

SSD の方が高速
素晴らしいスループットとランダムシーク
CPU スピード/数は、I/Oが高速なときに重要

T520: ノートPC, 8GB, 4-スレッド, HDD



Z820: サーバー, 256GB, 32-スレッド, SSD



2つのホスト上のウォームキャッシュ パフォーマンス

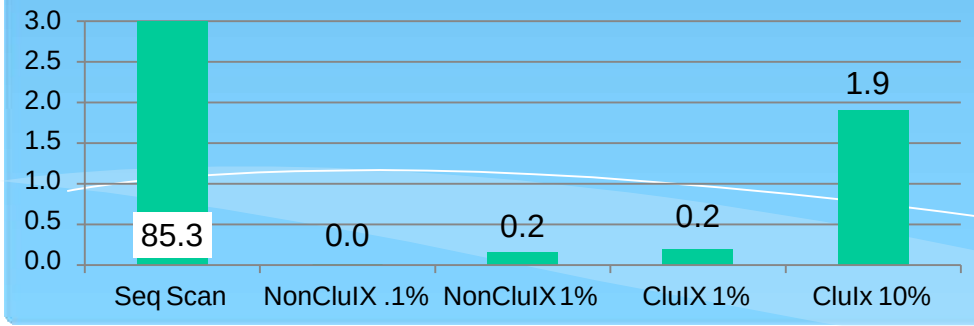
データがキャッシュ上にある場合には、CPUが主な要因

並行処理も可能だけれどもオーバーヘッドがある

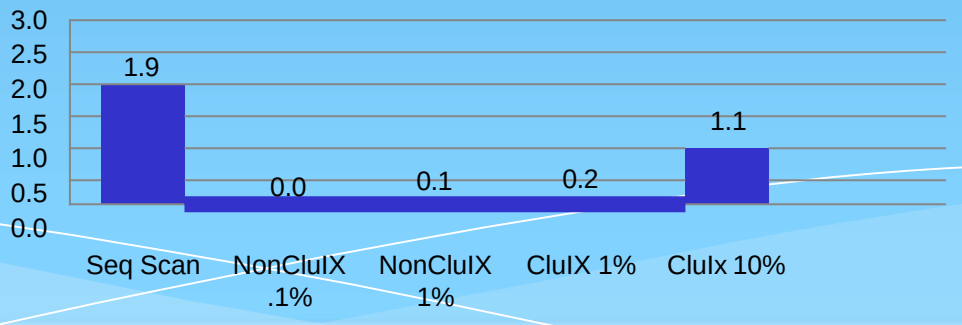
クロックスピードが重要な要因

バッファープールコンテンツに大きなインパクトがある

T520: ノートPC, 8GB, 4-スレッド, HDD



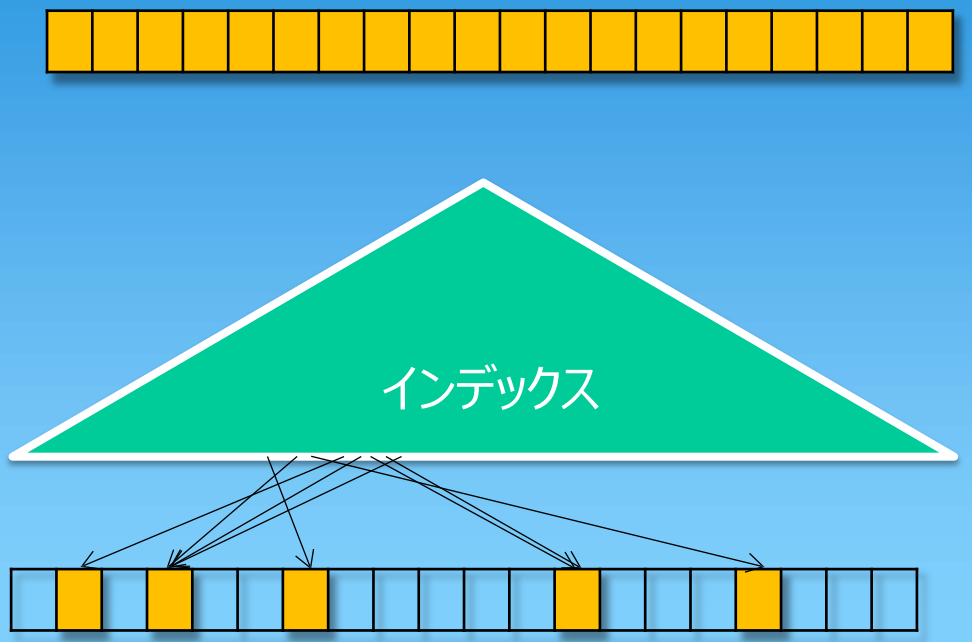
Z820: サーバー, 256GB, 32-スレッド, SSD



アクセス方法

フルテーブルスキャン

インデックススキャン



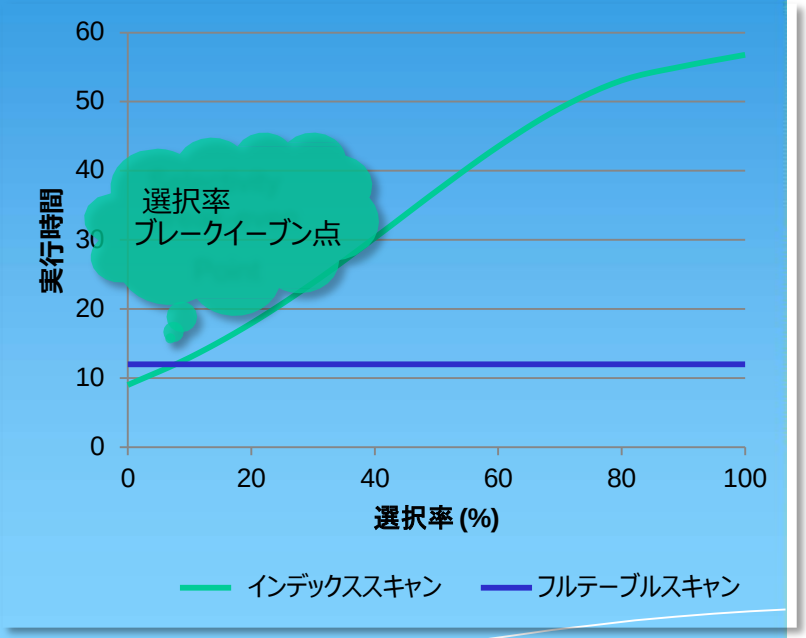
アクセス方法を決定する

フルテーブルスキャン

- テーブル内の全ページの読み込み => 不必要な I/O ☹
- 全行の処理 => より多くの CPU ☹
- しかし、シーケンシャル I/Oによるメリット ☺

インデックススキャン

- 必要なページのみの読み込み ☺
- 必要な行のみ処理 => より少ない CPU ☺
- ランダム I/O なのでたいへん ☹
- テーブルページに追加してインデックスページを読む必要あり ☹
- 同一のテーブルページを再フェッチする必要の可能性あり ☹
- 選択性が大きい場合には、全テーブルページを読む必要がある可能性あり ☹



インデックスとテーブルスキャンを選択する要素

選択性

選択性が大きい → テーブルスキャン

選択性が小さい → インデックススキャン

行サイズ (ページ毎の行数)

より大きな行サイズ→ブレークイーブン点を右へシフトさせる（インデックススキャンの方がうまく機能する）

キャッシュコンテンツ

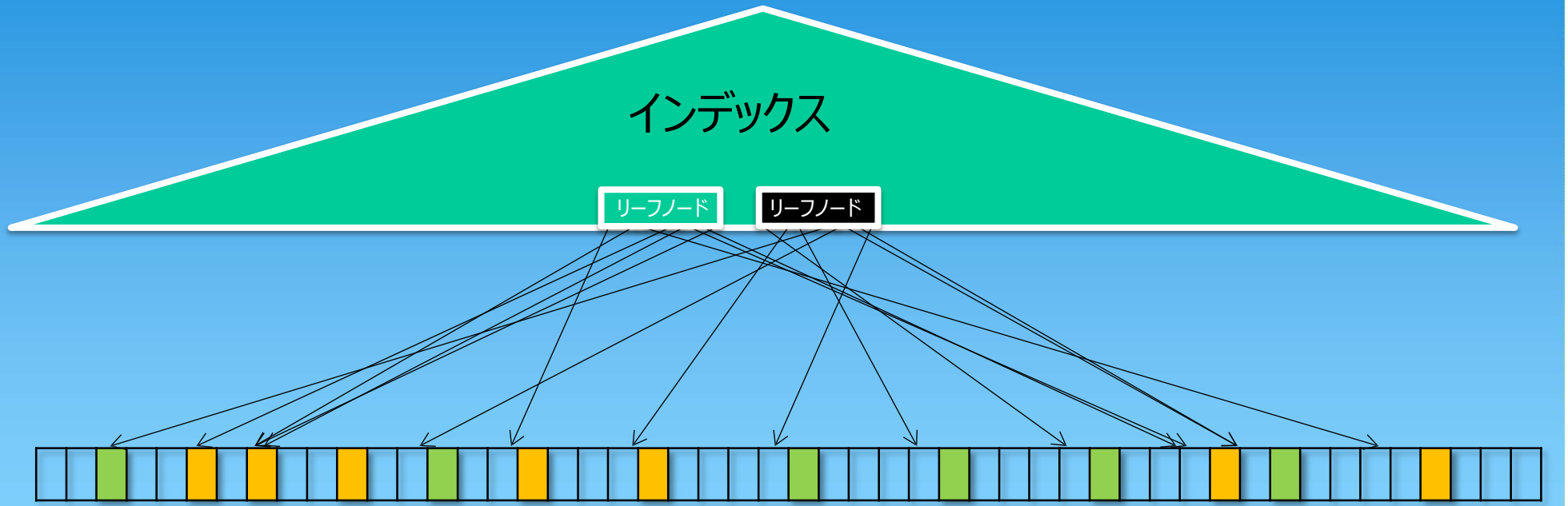
キャッシュにあるテーブルでより大きくなる。より多くの読み込みは、キャッシュによる

利用可能なメモリ

より大きな利用可能なメモリ → ブレークイーブン点を右にシフト (インデックススキャンの方がうまく機能)

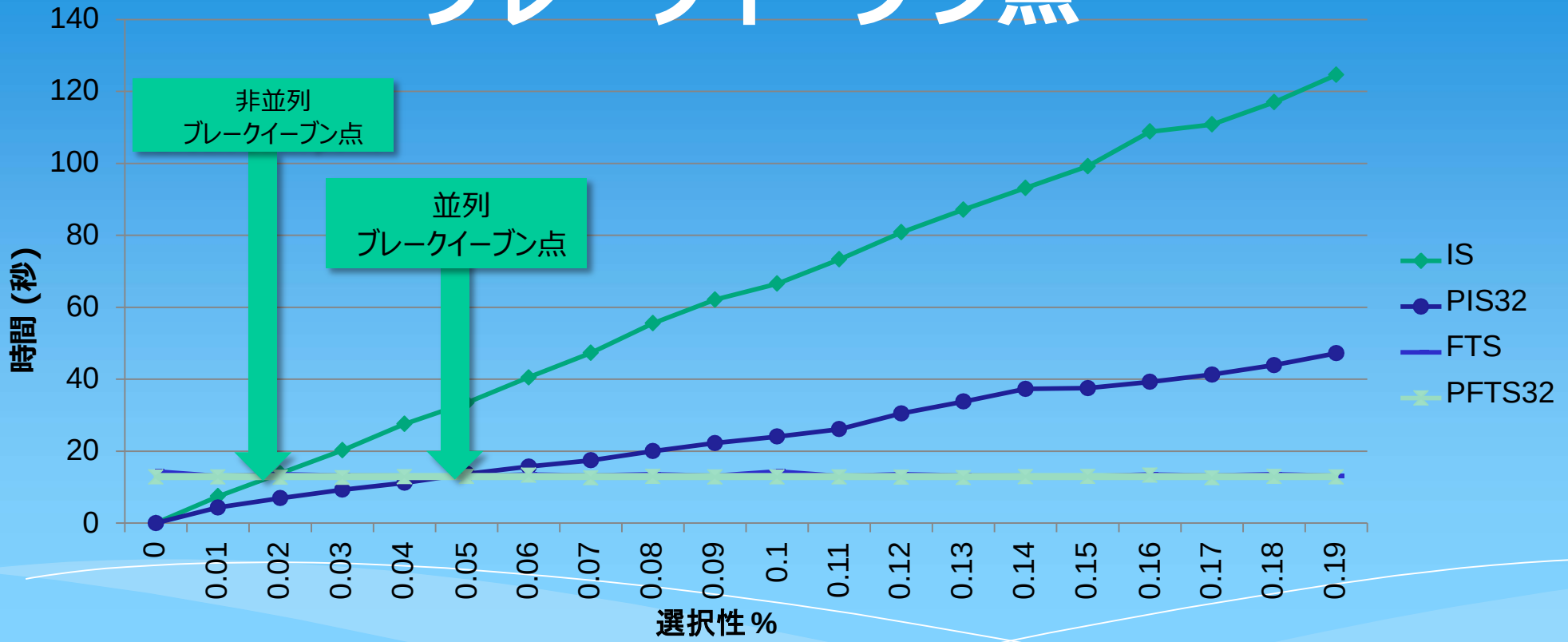
I/O の並列は?

SAP SQL Anywhereにおける並列インデックススキャン

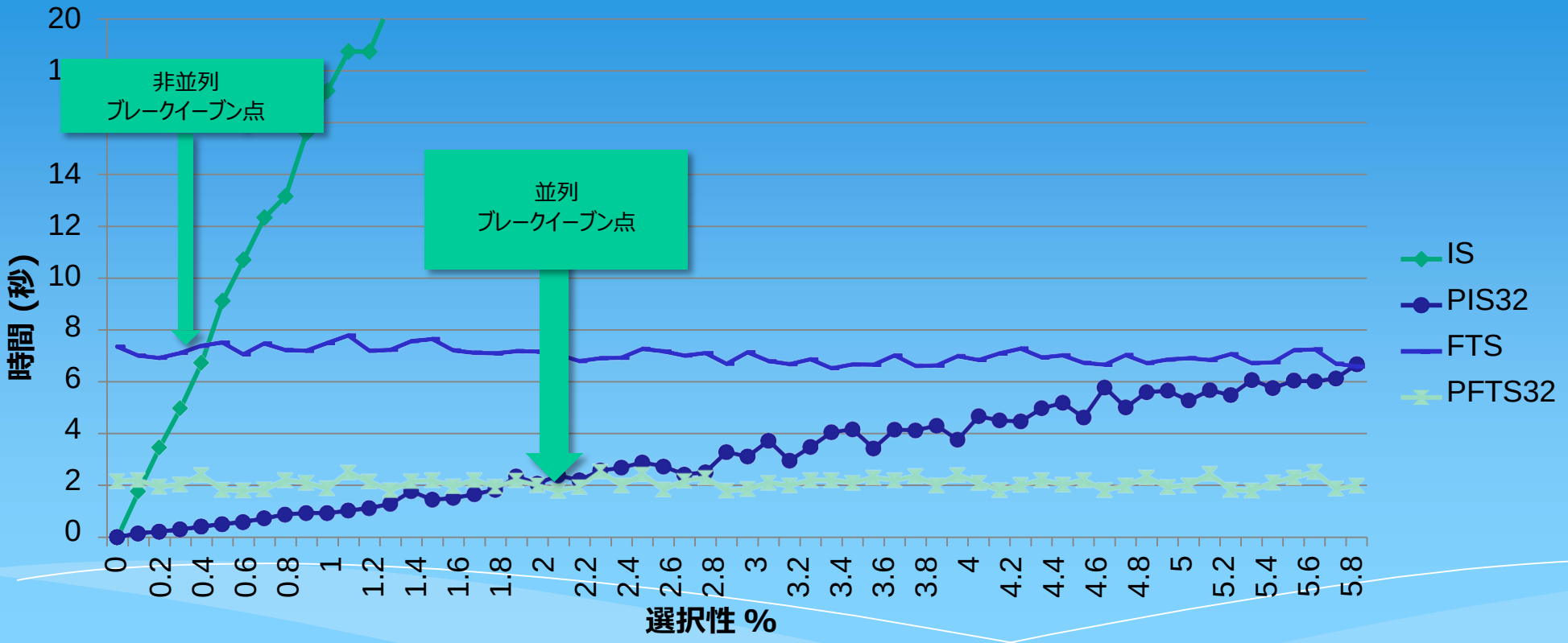


HDD – 並列インデックススキャンでは、ブレークイーブン点が移動

ブレークイーブン点



SSD – ブレークイーブン点はさらに右に移動



ブレークイーブン点のシフト

	NP-HDD	P-HDD	NP-SSD	P-SSD
RPP=1	0.55%	1.4%	8%	48%
RPP=33	0.02%	0.05%	0.4%	2.1%
RPP=500	0.0045%	0.005%	0.15%	0.5%

SSD では、選択性のブレークイーブン点のシフトは非常に高い

→そのため、クエリオプティマイザは、並列I/Oのインパクトについて認識している必要がある

さもなくば、最適ではない実行プランでは最適なプランと比較して最大20倍遅い

→ ALTER DATABASE CALIBRATE SERVER を検討する

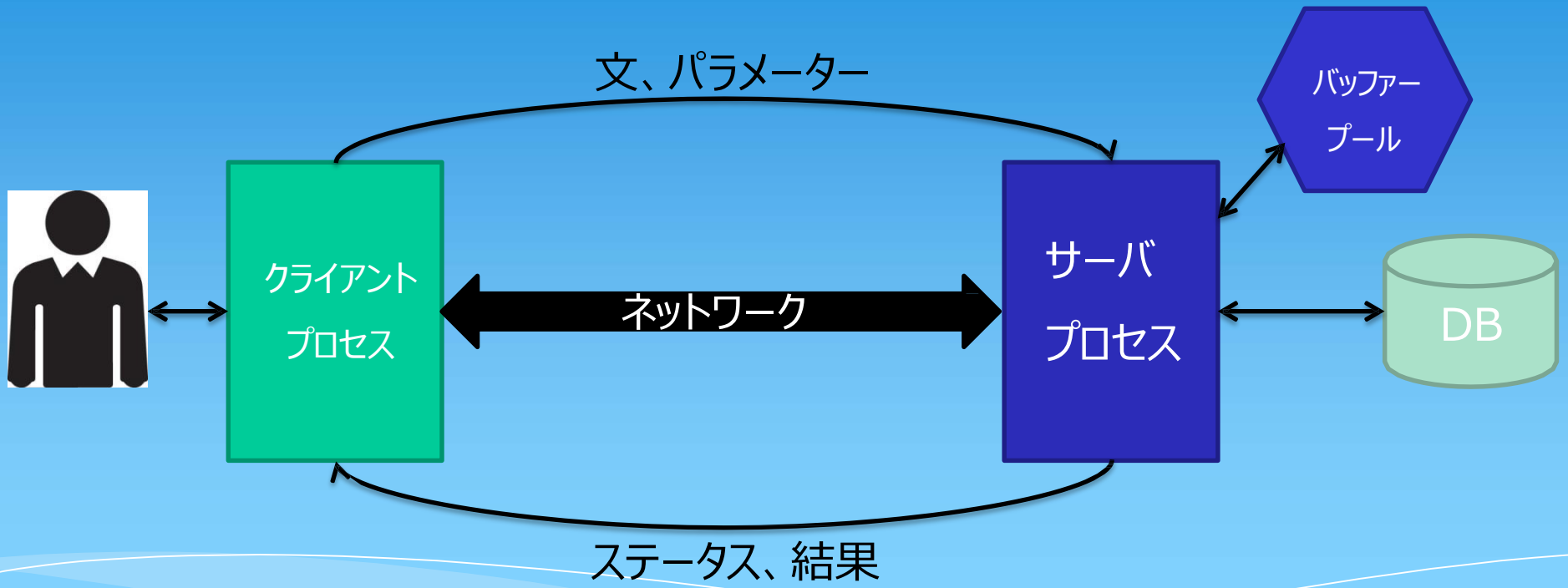
データベースが1つの設定を実行するのであれば、キャリブレーションを検討する

ディスクの設定を制御できるのであれば、デフォルトのキャリブレーションがベスト

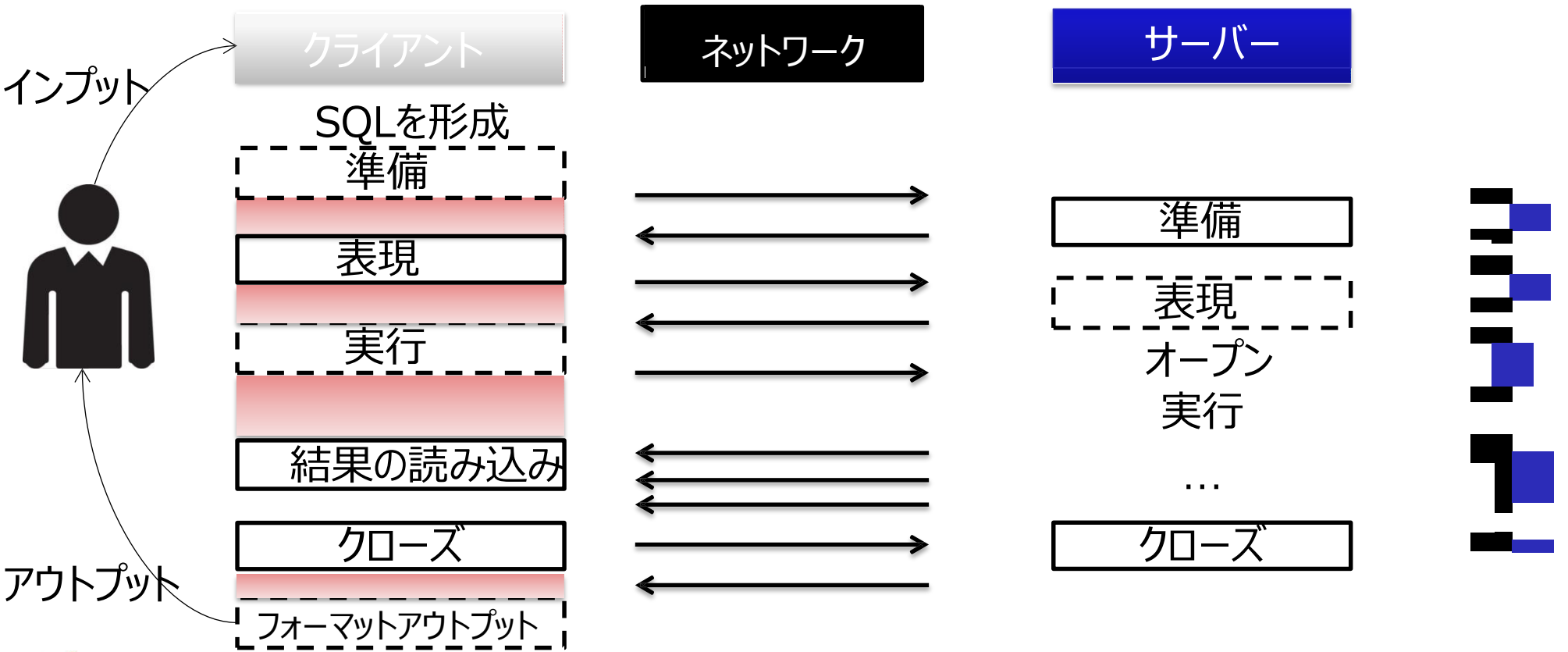
アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

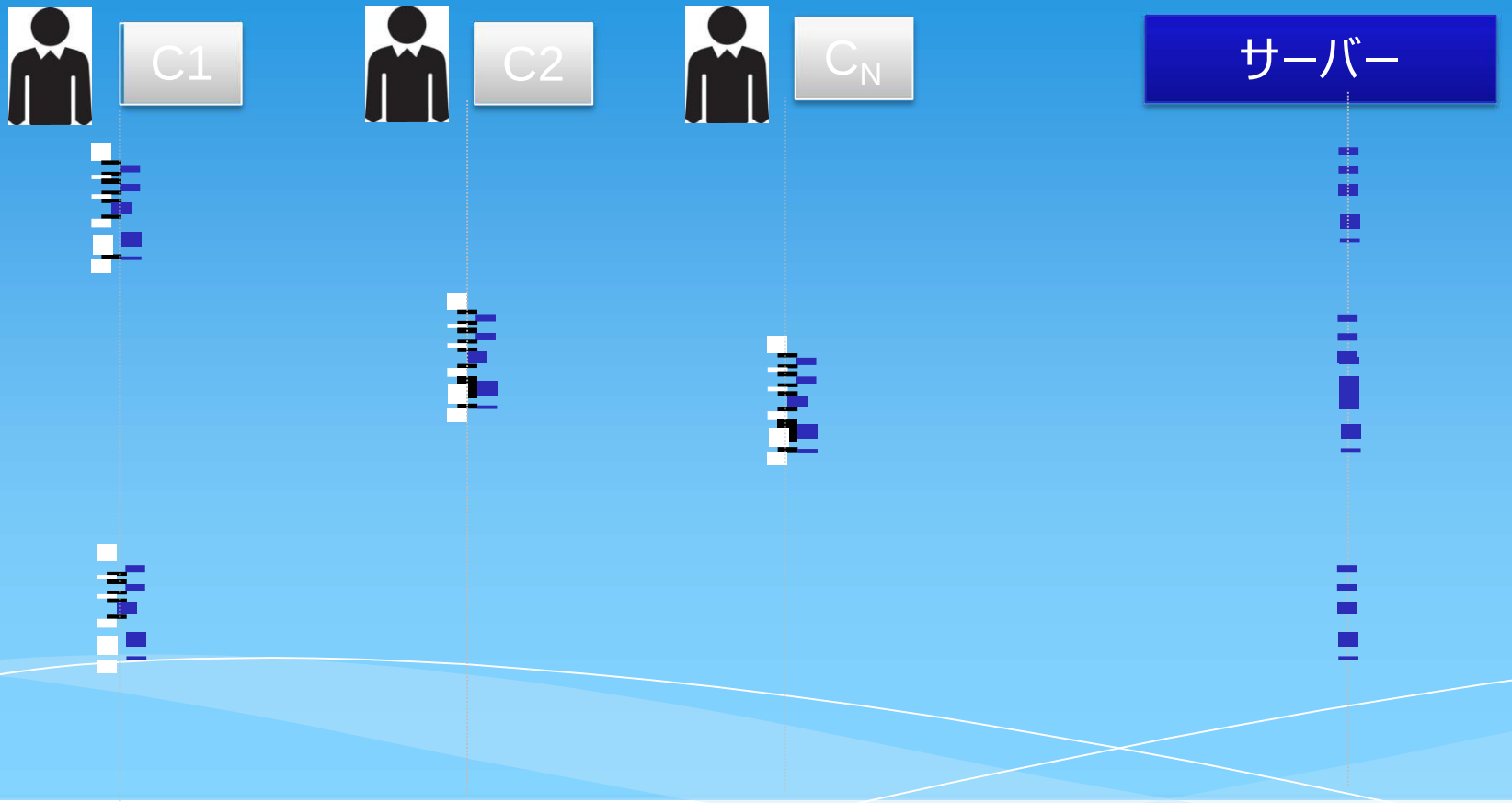
データベースシステムの要素



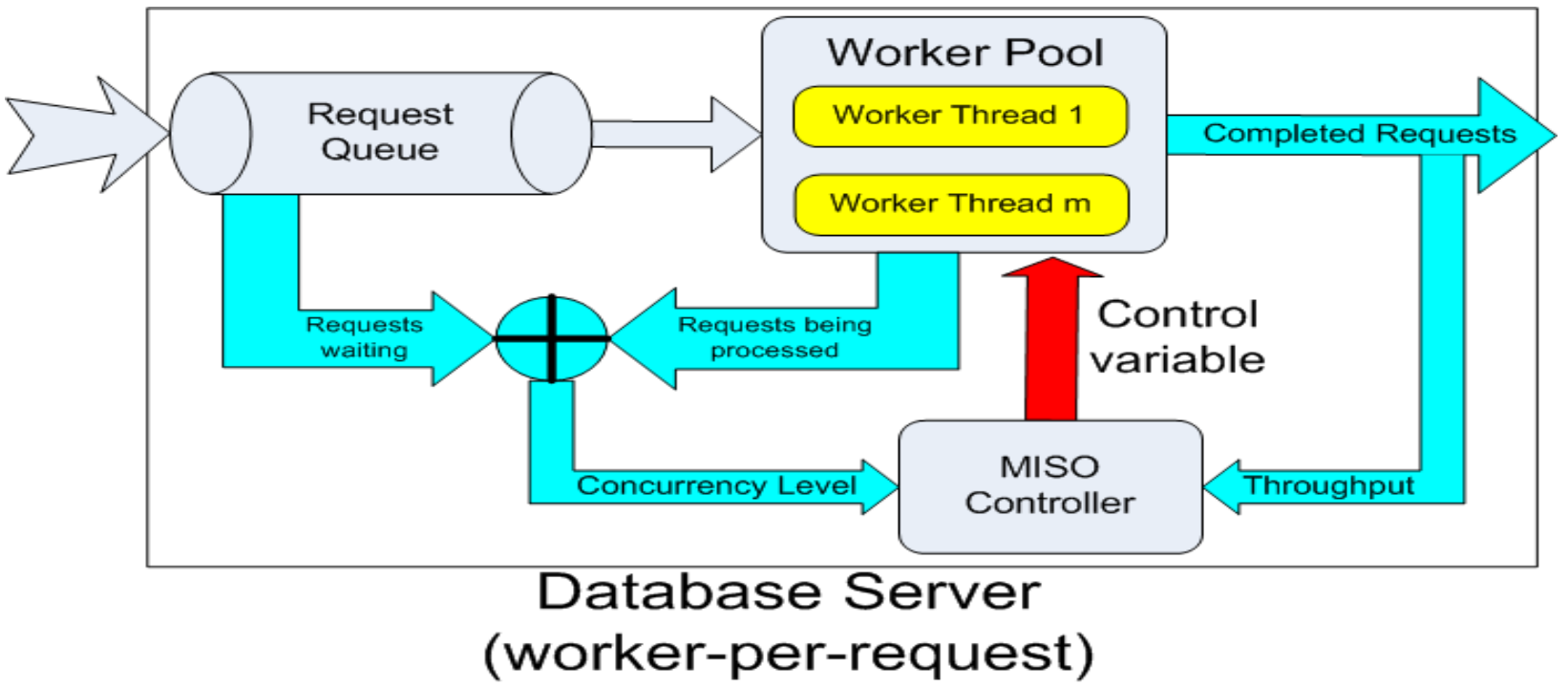
単一文の解剖



同時リクエスト



SQL Anywhere スケジューラー



タスク スケジューリング

- データベースサーバーに a worker-per-requestがある。
  - ワーカープール1つとリクエストキュー1つ
    - それぞれのワーカーが1つのリクエストをピックして完了させる
    - 同じのワーカーが同じ接続をサーブするかどうかの保証はない
- 小さなワーカーのプールがリクエストを実行
  - スケジューラーは、ワーカーに対して動的にワークをアサイン – 協調的なマルチタスキング
  - アサインされていないリクエストは、利用可能なスレッドを待つ
  - サーバーは、動的イントラクエリパラレリズムをサポート
    - パラレリズムの程度は、利用可能なリソースによって変化
- プールサイズがマルチプログラミングレベルを確立
  - SQLAのデフォルト: 20
- プライオリティは、接続時に設定可能
  - あらゆる任意のリクエストが得るタイムスライスに合わせて調整

Worker-per-Request のアーキテクチャー

ワーカープールのサイズの選択方法?

- 大きめのワーカープール:
  - サーバーの同時接続レベルが増加
  - サーバーリソースのコンテンションが増加
  - サーバーのワーキングセットサイズが増加
- 小さめのワーカープール:
  - ハードウェアリソースのユーティライゼーション
  - ワークロードの同時接続レベルを制限
  - リクエストのハンドリングをするワーカーがなく、サーバーがハングアップ

動的ワーカープール管理

ワーカープールのサイズを動的に調整

- ワークロードのスループットのモニタリングとペンディング中のリクエストをベースに行う

動的 MPL の利点：

- DB 管理者が気にするパラメーターが1つ減る
- 異なるワークロードのサーバースループットを改善
- ワークロードトランザクションがミックスされた環境でも、より柔軟に変更に対処

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

動的メモリ管理

SQL Anywhere サーバーは、以下の 2 つの**両方**に対応するため、必要に応じてバッファープールを増やしたり縮ませたりできる。

- データベースサーバーのロード
- 他のアプリケーションの物理メモリの要求

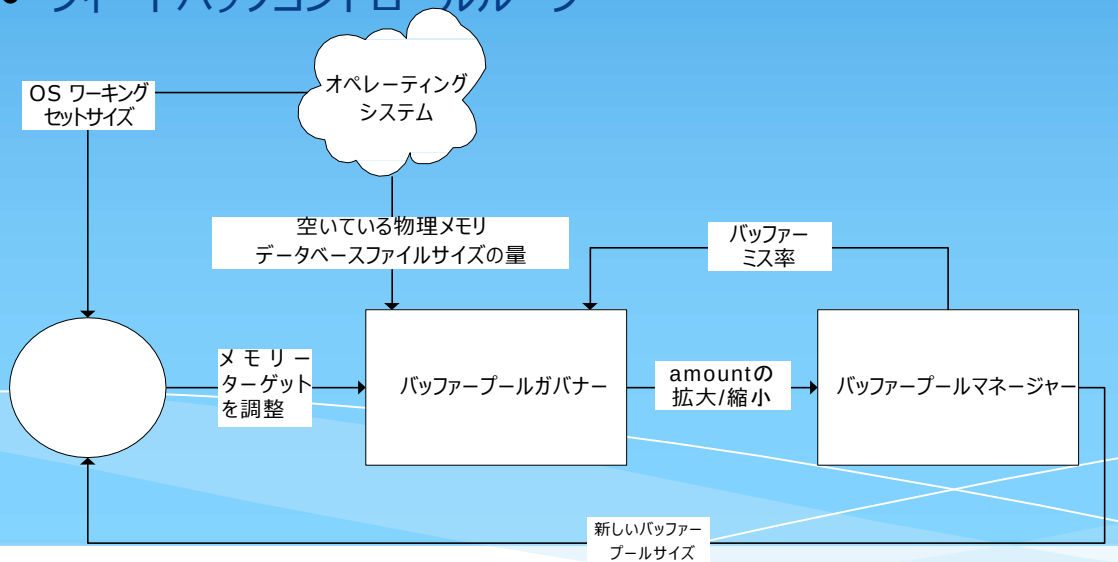
全サポート OS において、デフォルトで有効に設定

- ユーザーは、より低い、またはより高いバンドを最初のサイズとして設定可能

動的メモリ管理 – バッファープールサイズに適応

基本的な考え方: オペレーティングシステムにプラスしてOSのフリープールによって決定されたSQL Anywhere のワーキングセットにバッファープールサイズを合わせる

- フィードバックコントロールループ



SQL Anywhere のメモリ管理

事前に定義された制限がほとんどない単一のヘテロジニアスなバッファープール

バッファープールの構成要素

- テーブルとインデックスデータページ
- チェックポイントログページ
- ビットマップページ
- ヒープページ (クエリ実行プランのためのデータ構造、最適化グラフ、接続構造、ストアドプロシージャー、トリガー)
- フリー (利用可能な) ページ

全ページのフレームは、同じサイズ

完全に含まれるメモリーマネージャー

- 自己管理されるメモリーフットプリント

キャッシュウォーミング

スタートアップキャッシュウォーミング

「スタートアップ期間」に参照されたページを記録

これらのページを将来のスタートアップでも読み込み

最初のいくつかのリクエストのために必要なデータをすぐにロードするため

定常状態キャッシュウォーミング

キャッシュの平常状態の近似値を記録

スタートアップウォーミングが行われた後、バックグラウンドでは、ロードアップページが必要であると期待される

- V17より

キャッシュ コンテンツ見積もり

全テーブルとインデックスが、キャッシュにあるページ数をメンテナンス

- ページが、読み込み/evictされている場合、これはインクリメント/デクリメントされている

コストモデルは、ディスク読み込みがどれだけ必要か見積もり

- プランによって参照される明確に異なるページ数を見積もり
- すでにバッファープールにどれだけありそうか見積もり
- これらのうち複数回読み込みがバッファープールに残るのか見積もり

→パフォーマンス評価の場合には、バッファープールコンテンツを検討

- 状態を安定化させる実験を行う前に、キャッシュのフラッシュまたはウォーミングを検討

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

SQL Anywhere クエリオプティマイザー

SQL Anywhereは、実行されるリクエストを毎回最適化

- サーバーのコンテキストを考慮
- 最適化プロセスには、ヒューリスティックとコストベースの再書き込みの両方が含まれる
- ハードの制限はなし – 単一のブロックで500の quantifier でテスト

メリット

- プランは、サーバー環境、バッファープールコンテンツ/サイズ、データスキューに反応
- 「パッケージ」を管理する必要なし (事前最適化された SQL)

最適化努力は予測されるクエリコストと最適化のメリットに適応

- 単一の文がオプティマイザーをバイパス
- チープながら複雑な文はプランキャッシュを使用
- 予測されるメリットに応じて、オプティマイザーは、複数のジョイン enumeration アプローチを考慮

クエリオプティマイザーのバイパス

「複雑性」がない単一のテーブルクエリでは、オプティマイザーをバイパスさせる：

- 特定のフォーム (select * from T where pk = value) がある場合、単一の「バイパスキャッシュ」プランを使用
- 理にかなったプランが1つしかない場合 (WHERE 句がユニークな行を特定する)、ヒューリスティックをバイパス
- “bypass costed” は、代替のインデックスとシーケンシャルスキャンを比較
  - サブセットは、 術部オプティマイゼーションやセマンティック変換の試みをスキップできる「bypass costed simple」
- バイパスのオプティマイザーが　> 5 秒のプランを発見すると、フルのオプティマイザーで再最適化

プランキャッシングと自動パラメータ化

ストアドプロシージャー/トリガー/イベントにあるクエリのためのアクセスプランは、将来の実行のためにキャッシュされ、再使用される

- プランは、「トレーニング期間」を得て、プランの分散（偏差）を決定
- 分散（偏差）がない場合は（変数の値がない場合でも）、プランはキャッシュされ、再使用される
- クエリは、対数スケールで定期的に再最適化され、プランが次善にならないようにする
- V16 と v17 における改善では、パフォーマンスをデグレードさせる場合のプランキャッシングはしない

→ max_plans_cached=0 を設定しないように

適応型クエリ処理

実際の中間結果サイズの見積もりがよくない場合は、代替のアクセスプランを実行

- サーバーは実行時に代替のプランに自動的にスイッチ

バッファープールの利用が高い場合には、低メモリ戦略を使用

それを行うことで有利になるのであればアクセスプランを並列化

- 並列化の程度は、enumeration 処理間のコストで決定

ワーカープールサイズのワークは独立してパーティション化

- 並列化の程度に関して、プランの大半はセルフチューニング
- 一定期間、利用可能なワーカー数が最適な数よりも少ない場合には、クエリフラグメントの枯渇を防止

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

自動統計情報管理

- 自動チューニングカラムヒストグラム
  - ベーステーブルとテンポラリーテーブルの両方
  - 統計情報は、自動的にオンザフライで更新
- 最適化プロセス中の中間結果分析のためのジョインヒストグラム構築
  - パーシステンスではない
- サーバーは最適化中、パーシステンスのインデックス統計情報をリアルタイムにメンテナンス
- 最適化中にインデックスをサンプリング
  - ヒストグラムがない、または「信頼がない」と報告する場合
  - 1つ以上の術部をカバーしているインデックスがない場合 (単一 - のカラム見積もりを組み合わせるよりも良い)

カラムヒストグラム

術部の見積もりと更新の結果とともにリアルタイムに更新
- DML 文
  - デフォルトでは、統計情報は毎DMLリクエスト中に計算

ヒストグラムは、自動的に LOAD TABLE または CREATE INDEX 文で計算される
- 必要な場合、明示的に作成/ドロップ
- アンロード/リロードに対してデフォルトで保持

セルフヒーリング統計情報へのモチベーション

自動チューニングした統計情報の品質は、恣意的にデグレードすることが可能

- ロールバックに直面した場合には、同期しないことも可能
- 統計情報の生成は、データを一度見て、度を越している
- セルフチューニングでパーフェクトにしないことがゴール
- 自動チューニングはビジーなサーバーでは「高速を継続」できない可能性がある

→システムは自らモニターして修正する必要がある

セルフヒーリング統計情報

バックグラウンドサーバー処理の内部システム

- エンジンとクエリ実行の低オーバーヘッド

統計情報ガバナー

- QP間の見積もりエラーをカテゴライズして記録
- 統計情報の「品質」をセルフモニター
- 「良くない」統計情報をセルフヒーリング
- 「悪い」統計情報を取り除く

SQL Anywhereのソリューション

統計情報フラッシャー

- メモリーから使用されていない統計情報をアンロード
- カラム統計情報のヘルスをアドバイス
- カラム統計情報の使用方法をアドバイス
- 統計情報を作成またはドロップするかどうかをアドバイス
- 30分毎に実行

統計情報クリーンナー

- フラッシャープロセスによってトリガーされ、修正できない統計情報を修正
- 悪い統計情報が見つかったテーブルIDをトラッキング
- バックグラウンドプライオリティーで実行

統計情報の修正

統計情報の品質を改善するためにいくつかの方法が使用されている

ユーザーのクエリをPiggyback-offする
- テーブルの大部分を見るアクセスプランを利用
- クエリ実行の間インラインの統計情報収集を実行
- *in-situ* をリプレースまたは修正

インデックスから再作成
- ピギーバッグのためのフォールバックメカニズム
- シャローインデックスを使用して、ヒストグラムを再作成

サンプル化されたテーブルスキャンを実行
- テーブルカラムがインデックスを持たない場合、統計情報を得るためにはテーブルをスキャンしなければならない
- 少ない数のテーブルページのランダムサンプルを読む

異常状況を検出し、セルフヒーリングを防止、またはさらにヒストグラムをドロップ

アジェンダ

- レビュー
  - SQL Anywhere の設計ゴール
  - 自己管理はなぜ重要か
  - SQL Anywhere におけるクエリ処理
- SQL Anywhere パフォーマンス
  - シーケンシャルスキャン vs インデックススキャン
  - マルチプログラミングレベル
  - キャッシュ管理
  - 適応型クエリの実行
  - 統計情報管理

まとめ

SQL Anywhereの速さはどのくらい?

「状況による」が、正しい答え！

オプティマイザーは、**その時点で**可能な最高のパフォーマンスを提供/維持するために、複数のソースからの変更データを、リアルタイムに、コーディネート

しかし、これはパーフェクトではない!

特別な方法

ALTER DATABASE CALIBRATE SERVER を検討

パフォーマンスを評価する際は、バッファープールコンテンツを検討

max_plans_cached=0 は設定しない

Questions?

Jason Hinsperger

jason.hinsperger@sap.com