Tableau Server のスケーラビリティ

サーバー管理者のためのテクニカルガイド

Neelesh Kamkolkar
データ & パフォーマンス部門プロダクトマネージャー

tableau

目次

エグゼクティブサマリー

多くの組織で、Tableau は欠かせない存在となっています。企業では、さまざまな部門でデータからインサイトを見つけて大きな価値を引き出すようになっており、IT 部門はビジネス部門と連携を取って、全社的な分析のプラットフォームとして Tableau の導入を進めています。このように企業が Tableau の導入を開始する際、現在だけでなく将来にわたってビジネスにおける分析ニーズに応えていくためには、エンタープライズアーキテクトや IT リーダーが、データやコンテンツ、ユーザーに合わせて Tableau Server がスケールする方法、および多様で異種混合エンタープライズ IT プラットフォーム全体に Tableau Server を導入して統合する方法を理解していることはきわめて重要です。

このホワイトペーパーでは、エンタープライズアーキテクトおよび IT リーダーを対象として、Tableau のアーキテクチャ、およびワークロードの増加に応じたスケーリングについて詳しく説明します。

Tableau では、Tableau Server 10 のスケーリング、および従来のバージョンと比較した結果について、調査と試験を行いました。ユーザーからのリクエストに応えて、スケーラビリティテストでは、ユーザースケーラビリティに加え、バックグラウンドワークロードもテスト範囲に含めています。

Tableau Server のパフォーマンスやスケーラビリティに影響を及ぼす可能性があるシステム上の要素は多数あります。システムの重要な可変要素としては、たとえば、ワークブックのデザイン、サーバーの構成、インフラストラクチャの調整、データ環境、計算処理能力、ネットワークなどが挙げられます。これらの要素は、使用プロファイルや導入状況によって大きく変わります。これらの可変要素を調整したり変更したりすると、スケーラビリティテストの結果が異なります。可変要素を隔離して影響を緩和するために、ラボの閉鎖的なネットワーク環境で物理マシンを使ってテストを実施しました。このテストの目標は、外部システムからの影響による測定値の変動を最小限に抑えることでした。それにより、Tableau Server の実際の利用状況をモデル化して、スケーラビリティ指標を測定することが可能になります。

この目標に向けて、まず、Tableau Server の実際の稼働環境におけるピーク利用時の分析を行い、自動化されたテスト環境でその利用状況をモデル化しました。このように 2 段階の手順を踏むことで、非常に現実に即したワークロードを再現しています。この方法では、実際のユーザーやバックグラウンダーワークロード (主にデータ抽出やユーザー通知など) によるシステムの使用状況がどのように変動するかもシミュレーションします。ここで言う変動には、ユーザーがビジュアライゼーションを操作する際の待ち時間の長さや、スケジュールに従って実行されるバックグラウンダージョブの数などが含まれます。

実際の稼働状況を模倣するために、テストではこのような変動性をモデル化しました。Tableau Server クラスタにワーカーノードを追加すると、それに比例した直線的なスケーリングが見られました。この実験では、実稼働環境の測定結果を超えるピーク時の負荷条件をサーバーに課しています。実稼働環境では、平日は日中を通じて短時間の使用量のピークが頻繁に発生する傾向があります。ここでは、負荷をスループットで表して測定しています。スループットとは、一定時間内のサーバーの処理量のことで、簡単に言えば 1 秒当たりのトランザクション数です。下図のように、この実験では、1 秒あたりのトランザクション数が 4 から 18 になり、スループットが増加しています。それぞれの棒は、見出しに記載されているトポロジ (スタンドアロンサーバー、プライマリ + ワーカー 1 など) で実行された実験の結果を示しています。

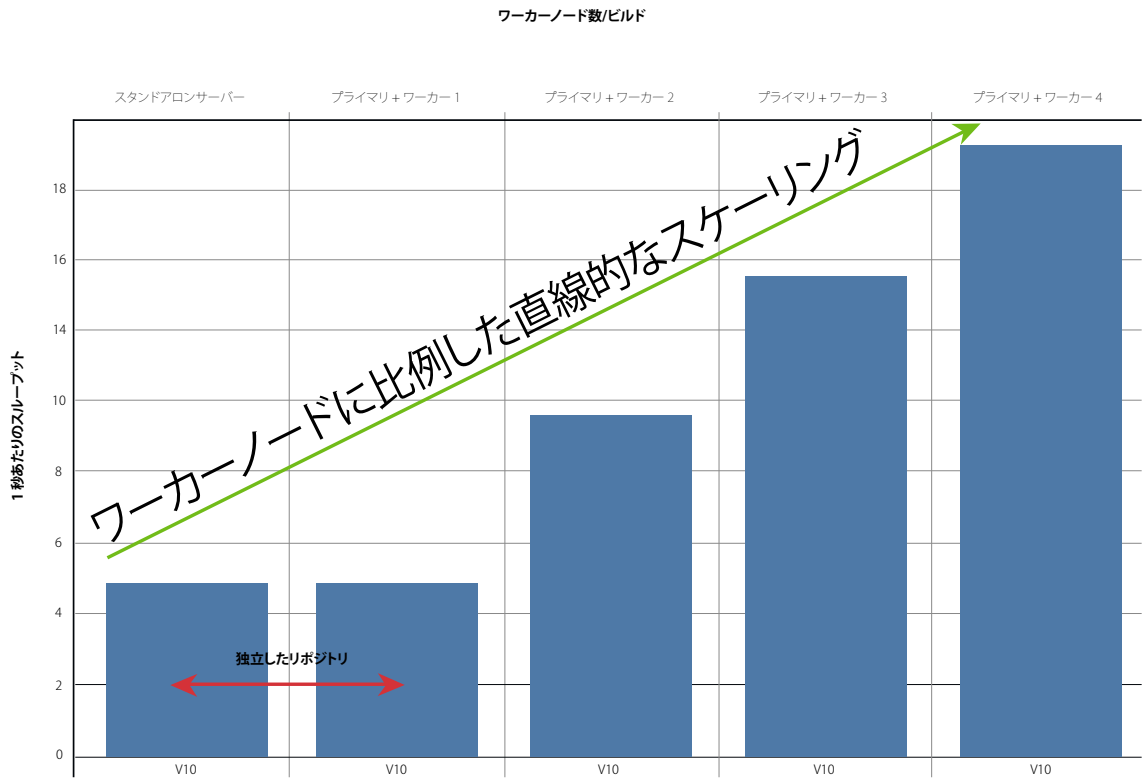

図 1: シナリオの 1 秒あたりのスループット

実験では、物理コアを 8 コア搭載した単一のサーバーで対応可能なアクティブユーザーの負荷の最大数を特定しました。アクティブユーザーの負荷は、Tableau Server に対して実行される一連の操作を自動化したものです。アクティブユーザーの負荷にどのような操作が含まれているかは、このホワイトペーパーの後のセクションで詳しく説明します。物理 8 コアを搭載した Tableau Server で対応可能なアクティブユーザーの最大数を測定した後、そのアクティブユーザー数を基準の単位として、均質 8 コアのワーカーノードをクラスタに追加することで、負荷を直線的に増加させることができるかどうかをテストしました。

その結果、8 コア搭載のサーバー 1 台で、ピークが持続する現実的なモデルの負荷をかけて、アクティブユーザー 112 人まで対応することができました。これをベースとして、それぞれ 8 コアの均質ワーカーノード 4 つと、リポジトリおよび基本的なインストールのみのプライマリコントローラノードを追加して、直線的に拡張していったところ、アクティブユーザー 448 人まで対応することができました。これはシステム上でユーザーが同時に作業を行った場合なので、すべてのユーザーが常時システムを使用しているわけではないと予測すると、8 ～ 32 コアのクラスタ構成で 500 ～ 10,000 ユーザー程度は対応できると考えられます。後ほど詳しく見ていきますが、このような推論はさまざまなパラメーターに依存します。他の可変要素がすべて同じであれば、ユーザー数に対応したサーバーのサイジングは、分析の利用状況とデータ条件に左右されます。このテストにより、Tableau のアーキテクチャでは、Tableau Server クラスタにワーカーノードを追加することで、対応可能なユーザーベースを直線的に拡大し続けることができると判明しました。

このテストでは、サーバーに負荷をかけて、エラーやタイムアウトが発生するポイントを確認しました。この負荷テスト (ストレステスト) により、サーバーのスケーラビリティとパフォーマンスの限界を示す上限値が明らかになりました。これらの上限を数値化しながら、実稼働環境における負荷特性の域を超えて、ピーク負荷を維持できるよう

に設計しました。実際には、もっと安定した状態であることが普通なので、このホワイトペーパーに書かれている以上の使用状況に対応できると考えることができます。Tableau Server のアーキテクチャは、直線的なスケーリングが可能なので、現在のニーズにとどまらず、将来的な成長にも対応できるスケーラビリティを備えています。

各トポロジおよび使用事例を、下記の表にまとめました。「リスクプロファイル」列は、自動フェールオーバー、ハードウェア故障の可能性、急激なピークに対応できる能力などのさまざまな要素に基づいて、そのトポロジにどの程度のリスクがあるかを示しています。

展開の構成	ユースケース	リスクプロファイル
スタンドアロンサーバー	シンプルな単一のサーバー展開	可用性のリスクは高、リソースの競合は高、ピーク時の負荷でパフォーマンスが求められる場合の余力はなし
プライマリ 1 + ワーカー 1	2 ノードの展開	可用性リスクは高、リポジトリとのリソースの競合は低、ピーク利用時の余力はなし
プライマリ 1 + ワーカー 2	3 ノードの展開	可用性のリスクは中程度、リソースの競合は低、水平方向のスケーリング向上、ピーク利用時の余力はごくわずか
プライマリ 1 + ワーカー 3	4 ノードの展開	可用性のリスクは中程度、リソースの競合は低、スケーリング向上、ピーク利用時の余力は中程度
プライマリ 1 + ワーカー 4	5 ノードの展開	可用性のリスクは低、スケーリング向上、ピーク利用時の余力あり[1]

表 1: 導入トポロジと使用事例

[1] ピークに対応できる能力は、さまざまな要素に左右されるもので、保証されるものではありません。

これらの数字は、テスト方法やテスト環境を考慮したうえで理解することが重要です。そのため、ここからはアーキテクチャに焦点を当て、従来の BI テクノロジーと比較した Tableau アーキテクチャの独自性について、また、テスト方法やスケーラビリティのテスト結果の分析について説明していきます。

古いパラダイムを打ち破る VizQL

従来のビジネスインテリジェンス (BI) ソリューションに慣れている方や Tableau を初めて使う方は、従来の BI と比較した Tableau の主な違いを知っておくと役立ちます。Tableau は、従来の BI で前提となっていた「最初にクエリ、次にビジュアライゼーション」という考え方に挑戦しました。私たちは、リアルタイムにデータを探索し、疑問を追及する中でインサイトが得られるものだと考えます。Tableau には、10 年以上前から VizQL™ という特許取得済みのテクノロジーが組み込まれており、これによって、クエリとビジュアライゼーションが 1 つのプラットフォームに融合されています。強力な

機能が組み合わさったことで、エンドユーザーは、データを視覚化しながら、クエリ、フィルター、分析を行って、データに関する疑問をどこまでも追及していくことができます。VizQL™ は、ユーザーの質問や操作を表現し、社内やクラウド上のデータセットに対して実行可能なクエリに変換する、Tableau の基本的な言語です。10 年以上にわたってエンジニアリングへの投資を行ってきたことでテクノロジーとして成熟し、さらに次世代のデータソースや分析に関するニーズにも対応できるよう進化を続けています。

あらかじめ定義された固定的な要件に沿って設計・開発される従来の BI レポートとは異なり、Tableau のビジュアライゼーションは、対話的な操作やコラボレーションが可能な設計になっています。ユーザーは複雑な SQL クエリの記述や結合を行わずに、データに関する疑問を解き明かすことができます。疑問の答えを得るために、これまでのように、ソフトウェア開発のサイクルが一通り完了するまで待つ必要はありません。すでにあるビジュアライゼーションに対してさらに分析を続けて、ビジネスやプロジェクトに関する新たな疑問の答えを見つけることができます。

従来の BI では、最適化済みの対象システムに対して実行するように設計されたクエリを使用する静的なレポートで、特定のサービスレベル契約 (SLA) を満たす負荷テストを実施することがよくありました。静的なレポートには、固定されたスコープ、固定のクエリセットがあり、多くの場合、開発者が何週間もかけて 1 つずつ最適化しています。静的なレポートに対する SLA の定義や設定はそれほど難しいものではありませんが、レポートに変更を加えると、開発サイクル全体がリセットされます。このような制約は、通常、SLA では考慮されません。

これに対して、Tableau のビジュアライゼーションでは、ユーザーがさらにデータを調べるためのアクションを実行すると、新しいクエリが再生成され、送信されます。ローカルブラウザのキャッシュと VizQL の最適化によってデータをすばやく取得できるので、ユーザーはクエリの結果が返されるまで待つことなく、分析をスムーズに行うことができます。

VizQL という言語は、VizQL サーバープロセスに組み込まれ、他の数多くの分散されたサーバープロセスと連動することで、スケーラビリティ、可用性、安全性、信頼性に優れたビジネス分析プラットフォームを実現しています。

Tableau のアーキテクチャ

Tableau の柔軟なアーキテクチャは、スケールアップにもスケールアウトにも対応できるよう設計されています。Tableau は、エンタープライズスタンダードのプラットフォームとしても、クラウド分析を実現するプラットフォームとしても利用できます。Tableau Server は、きわめて複雑なエンタープライズ環境のインフラストラクチャ要件にも対応できるだけでなく、シンプルな部門ごとの導入や、ワークグループレベルでの導入にも対応します。

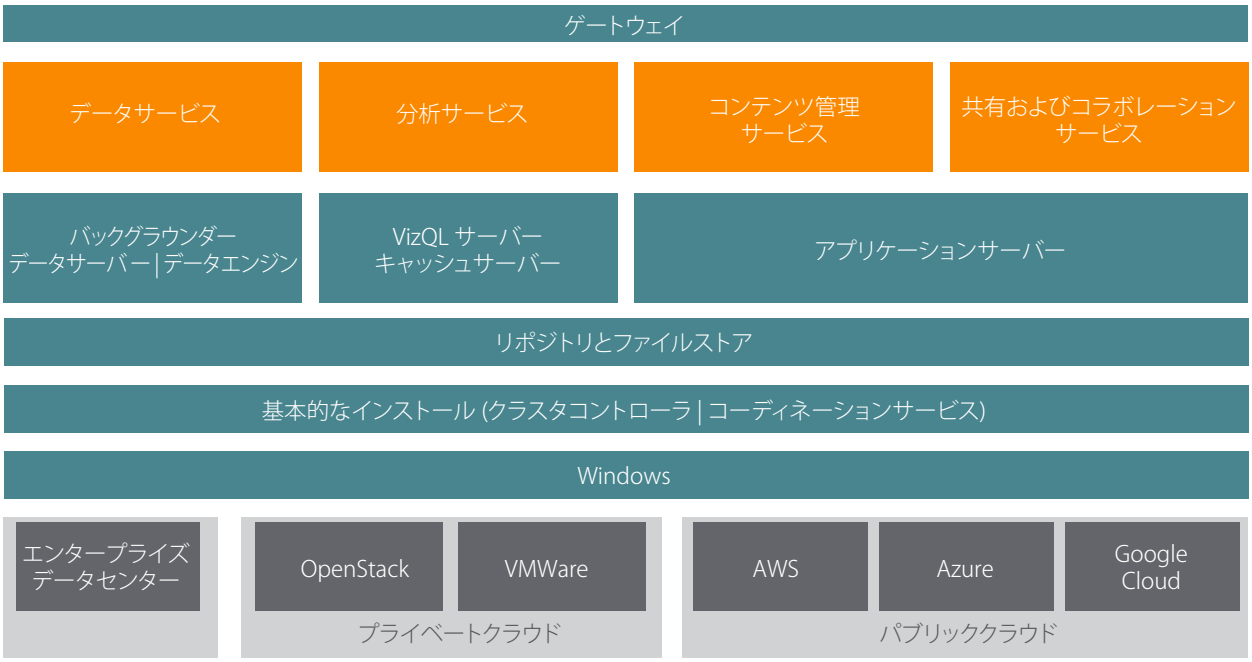

図 2: Tableau Server のアーキテクチャ

Tableau Server のインストールおよび構成のプロセスはシンプルです。インストールすると、複数のサーバープロセス (図 2 のブルーの部分) で多層的にさまざまなサービスが提供されます。

ゲートウェイプロセスは、すべての Tableau クライアントからのトラフィックをクラスタ内の利用可能なサーバーノードにリダイレクトするコンポーネントです。

データサービスは、最新データの取得、共有メタデータの管理、データソースの管理、インメモリのデータ利用を可能にする複数のサービスを論理的にグループ化したものです。その下には、データサービスを支えるプロセスとして、バックグラウンダー、データサーバー、データエンジンがあります。

分析サービスは、VizQL とキャッシュサーバープロセスで構成され、ユーザーとのインターフェイスとなるビジュアライゼーションおよび分析サービス、また、キャッシュサービスを提供します。

コンテンツ管理サービスと、共有およびコラボレーションサービスの下には、アプリケーションサーバープロセスがあります。ユーザーのログイン、コンテンツ管理 (プロジェクト、サイト、セキュリティパーミッションなどの管理)、管理タスクなどの Tableau Server の中核をなす機能は、アプリケーションサーバープロセスによって実現されます。

前述のすべてのサービスが利用し、依存しているのがリポジトリプロセスで、パーミッション、ワークブック、データ抽出、ユーザー情報、メタデータなどの構造化されたリレーショナルデータはこのリポジトリにあります。ファイルストアは、クラスタ間でのデータ抽出ファイルの冗長性を実現するプロセスで、これにより、すべてのクラスタノードでローカルの抽出を利用できるようになります。負荷が高い場合、クラスタ全体にわたりローカルの抽出ファイルを利用することにより、処理やレンダリングを高速に行うことができます。

Tableau のアーキテクチャは柔軟性が高く、プラットフォームを実行できる場所に制限がありません。Tableau Server は、オンプレミスでも、プライベートクラウドやデータセンターにも、Amazon EC2™、Google Cloud Platform™、あるいは Microsoft Azure™ にもインストールできます。Tableau の分析プラットフォームは、VMware ESXi™ や Microsoft Hyper-V™ などの仮想化プラットフォームで実行することもできます。Tableau Server から最大限のパフォーマンスを引き出すためには、各仮想化プラットフォームのベストプラクティスに従うことをお勧めします。

個々のサーバープロセスの詳細については、Tableau Server の管理者ガイドをご覧ください。

インメモリとライブが統合されたアーキテクチャ

Tableau は、異種混在プラットフォームに対応でき、さまざまなベンダーによる 50 以上の主要なデータソースに接続できます。また、インメモリで高速に分析を実行するか、ライブデータストアに直接接続して分析を行うかを、柔軟に選択することができます。どちらを選択しても、ビジネスのニーズやデータソースの変化に応じてライブ接続とインメモリの分析を切り替えることも簡単にできます。

Tableau ではインメモリでの分析がサポートされており、Tableau データ抽出 (TDE) という独自のカラム型ストアにデータを抽出し、メモリマップドファイルに読み込むことで高速なアクセスを実現しています。データ抽出は、Tableau Desktop を使用して作成するか、外部のビジネスプロセスから Tableau API を使って作成することができます。Tableau Server アーキテクチャには、管理者が制御する最適なスケジュールに従って、ユーザーの作成した抽出を更新し、最新の状態にするサポートが組み込まれています。

データに接続して Tableau Server に読み込む方法にかかわらず、データ分析、キャッシュ、抽出の更新、その他関連する処理に使用できる十分なメモリリソースを確保しておく必要があります。

純粋なインメモリツールとは異なり、Tableau の使用するメモリはクラスタ全体にまたがり、単一のサーバーだけに保持されることはありません。外部で共有されるクエリキャッシュにはキャッシュサーバーが、セッションレベルのキャッシュの保存にはその他のプロセスが使用され、クラスタ全体に分散する形でメモリが使用されます。そのため、負荷の上昇によるメモリへの影響は、ワークロードに応じてクラスタ全体に広がります。

テストの手順と方法

対象のアプリケーションをブラックボックスとして扱う、従来の多くの負荷テストプロジェクトとは異なり、Tableau Server の負荷テストは、そのアーキテクチャを十分に理解したうえで実施する必要があります。Tableau Server は、オンプレミスでもクラウドでも、どこでも実行できるように作られています。あらゆる規模のチームや組織に対応できる設計です。そのため、Tableau Server は既定のインストール方法でほとんどの標準的な導入に対応できます。Tableau Server を拡張して企業全体に導入していく場合は、さまざまなワークロードがどのように処理されるのか、また、導入のシナリオに合わせてほんの少し構成を調整することでどのような成果が得られるかを理解しておく必要があります。

今回は、バージョン 10 のスケーラビリティの目標に関して、次の質問に答えます。

1. 次の2 つの一般的なシナリオで、ハードウェアを追加することで、Tableau Server 10 の直線的なスケーリングが可能かどうか
   - エンドユーザーワークロードの増加
   - バックグラウンダーワークロードの増加

2. バックグラウンダープロセスを専用のワーカーノードに移すべきタイミングはいつか

3. Tableau Server 10 のパフォーマンスは、以前のリリースと比較してどうなっているか

以前のバージョンのホワイトペーパーでは、特にエンドユーザーのスケーラビリティに焦点を当てていましたが、お客様からバックグラウンダーワークロードのスケールに関して多くの質問をいただきました。バックグラウンダーワークロードでは、データの新しさ (抽出の更新) および分析が利用される範囲 (サブスクリプションの通知) を管理しています。

テストの方法

パフォーマンスとスケーラビリティの向上が、Tableau Server 10 リリースの主たる目標でした。そのため、実稼働環境に対応した、エンタープライズレベルのテスト方法を開発することが求められました。バージョン 10 のプレリリースバージョンについてこのセクションに記載したテスト方法を実施し始めました。アジャイル開発のプロセスを繰り返す中で、この方法により、20 件を超えるスケールおよびパフォーマンスに関するバグを発見し、修正しました。バージョン 10 のリリース後もテストを続け、10.0.1 メンテナンスリリース (MR) でバグを修正しています。このホワイトペーパーでは、全体を通じて基本的に Tableau Server バージョン 10 のリリースを対象としていますが、テスト結果は、10.0.1 MR に基づいています。バージョン 10 のスケールやパフォーマンスに関するメリットを十分に引き出すためには、10.0.1 MR またはそれ以降を使用してください。

Tableau では、スケーラビリティに関する手法を継続的に進化させ、お客様による実際の利用シナリオを再現するワークロードを収集し、テストしています。導入のスケーラビリティを測定するにはさまざまな要素がありますが、実際の導入について計画を練るうえでは、次の要素が重要になります。

• ユーザーによる影響 - セルフサービスの利用状況と、ユーザーによる受け入れ状況。分析を利用するユーザー数、情報に基づく意思決定を行うためにユーザーが分析を実施する頻度はどの程度か、ユーザーが作成するビジュアライゼーションの複雑さはどの程度か、など。

• データによる影響 - データの新しさ、サイズ、場所。データのサイズ、データの保存場所、ビジネス上の意思決定を正しく行うために必要なデータの更新頻度など。

図 3: 分析の利用状況とデータ更新頻度のマトリックス

これらの両方についてテストするために、エンドユーザーを増やしてサーバーへの負荷を次第に上げていくとともに、バックグラウンダーワークロードを増やしていくテストを組み込む必要がありました。このような方法で、エンドユーザーのサービス品質に対するバックグラウンダーワークロードの影響を調査しました。低・中・高の分析利用度およびバックグラウンダーワークロードの分離による影響をモデル化するために、ラボ環境でさまざまなワークロードを使用して 400 回を超えるテストを実施しました。システムのスケーラビリティについて調査するとともに、高負荷の状況でのみ明らかになるバグの修正も行いました。

現実の世界に関連するテスト結果を得るために、テストを実施するうえで適切なワークロードを確かめることから始めました。

実際のワークロードの特徴

バージョン 10 について、実稼働環境の Tableau Server が利用率のピーク時にどのように使用されているか観察を行い、その特徴を明らかにしました。

どのワークロードをモデル化するか、テストに適したワークロードの特徴はどのようなものかを判断するために、3,000 人のユーザーの実稼働環境から Tableau Server のログファイルを入手して分析しました。利用頻度の高いビジュアライゼーションとワークブックを特定し、それらのワークブックの利用分布を計算して、利用状況の特徴を分析しました。たとえば、リクエストの送信間隔 (ユーザーの思考時間) を調べました。実稼働サーバーのワークロードをモデル化するために使用した具体的な手順を以下に示します。

テストモデル作成のステップ

1. 使用状況のピーク時の実稼働サーバーのログを取得

2. サーバー上の時間の加重平均から、上位 N 位のワークブックビューを特定

3. 加重平均 = 平均応答時間 * リクエスト数

4. 上位 N 位のワークブックビューの相対的重みを計算

5. 選択したワークブックビューごとに
   a. refresh=y でビジュアライゼーションの読み込みのパーセント値を算出 (最新のデータを取得するために積極的にデータを更新したユーザーの数を確認するため)
   b. 重み付けにより上位 N 位の操作を特定
   c. 操作と操作の間の思考時間の平均を算出

6. モデルの検証のために
   a. ワークブックビューのブートストラップ間の平均時間 (テスト間隔) を確認
   b サーバー上の加重平均時間による上位 N 位のバックグラウンダータスクを確認

7. バックグラウンダーのトラフィックをタスクごとに分類 (サブスクリプションと抽出の更新)
   a. 上位のバックグラウンダータスクの平均時間を確認し、モデルに反映
   b. さまざまなスケジュールでにサブスクリプションの数を確認
   c. 抽出の更新のサイズと種類を確認 (データサーバーへの抽出のパブリッシュ、ワークブックの抽出)

上述のすべてのデータを使って、利用のピーク時に実際のユーザーがどのように実稼働サーバーを利用していたのかを示す、現実に即したワークロード構成をモデル化しました。最後に、実稼働環境のログを分析して、バックグラウンダーの抽出の更新およびサブスクリプションのワークロードベースのモデルを生成しました。

次の表は、ワークロードの組み合わせをまとめたものです。

ワークロード名	説明	以前のバージョンとの比較
実稼働環境サーバーのワークロード	IT 部門が管理するミッションクリティカルなアプリケーションとして、組織内の 3,000 ユーザーに提供される、実稼働環境の Tableau Server の利用状況を分析した特徴に基づくワークロード。	ここに記載の通り、このホワイトペーパーに記載されている結果データでのみ、9.3 と 10.0 の比較が可能。 テスト方法が大きく異なるので、9.0 のホワイトペーパーも含め、 以前のホワイトペーパーと結果を比較することはできない。
実稼働環境サーバーのワークロード + 新機能	上記のワークロードに、バージョン 10 の新機能を利用したワークブックを組み合わせたワークロード。	9.3 ではバージョン 10 の機能を利用できないので、9.3 と 10.0 の比較はできない。このホワイトペーパーの結果は、バージョン 10 のスケーリングに関する特徴を確認するのに最適。
バックグラウンダー混在	実稼働環境のワークロード分析に基づき、実稼働環境を反映させて、実際のワークブックおよびスケジュールのモデル化を行ったワークロード。	10.0 のスケーラビリティテストのホワイトペーパーで導入された新しい構成。以前のホワイトペーパーの結果との比較はできない。

表 2: ワークロードの組み合わせの詳細

次に、これらの組み合わせのそれぞれを取り上げて、物理マシン上の隔離されたスケーラビリティラボで別々に実行し、エンドユーザーの負荷およびバックグラウンダーの負荷を徐々に大きくしていきました。クラスタの処理能力の上限に達したところで、ワーカーノードを 1 つずつ追加して、負荷をさらに拡大し続けました。このような実験で、段階ごとにシステムの動作を観察しました。テストの実行ごとに、応答時間、スループット、エラー率などの KPI を記録しました。また、JMX を使用してシステムの指標およびアプリケーションサーバーの指標も記録しました。各テストにおいて、得られたデータを相関させ、ワークロードが増加していくとシステムがどのように動作するかを分析しました。また、このプロセスを通じて、スケーラビリティのバグの検出と修正も行っています。これは Tableau のアジャイル開発プロセスの一部でもあります。

バックグラウンダーに関する方法

バックグラウンダーサーバーでは、システムレベルおよびユーザーレベルのバックグラウンドジョブが処理されます。定期的なリポジトリのメンテナンスタスクなどの、システムレベルのジョブは、バックグラウンダーによって実行されます。ユーザーレベルのジョブとは、ユーザーがシステムに送信した可能性がある、システムがユーザーのために実行する必要があるジョブです。たとえば、ユーザーが抽出をサーバーにパブリッシュし、その抽出のデータをスケジュールに従って定期的に更新するように構成することができます。これにより、更新のジョブが作成されます。バックグラウンダーがジョブリストを確認して、ユーザーに代わってジョブを実行します。ユーザーは管理部門によってデータが更新されるのを待つ必要がなくなるため、効果的なセルフサービスを実現するためにこれは重要な機能です。ただし、Tableau Server の管理チームが、このタイプの負荷に対応する容量を確保していないと、エンドユーザーがサービス品質の問題に直面する可能性があります。サーバーのサイジングとプランニングにおいて、バックグラウンダーサービス用にどれだけ最適化する必要があるかは、重要な要素です。ワークロードを切り離すために、バックグラウンドサービスを分離して別のコンピューターに移すかどうかを検討する必要があります。

このホワイトペーパーの後半では、ワークロードの処理を時間帯に分けるための、いくつかのシンプルなベストプラクティスについて紹介します。ただし、バックグラウンダーを分析サービスと同じコンピューターで実行していると、負荷が高くなったときに、サーバー上のリソース共有の制約から、エンドユーザーのサービス品質に影響が出ることがあります。

このため、バックグラウンダーが分析サービスと同じコンピューター上で実行されている場合に、バックグラウンダーがエンドユーザーのスケーリングに与える影響を調べる必要がありました。さらに、バックグラウンダーを専用のハードウェアに隔離した場合に、負荷の増加に対してどのように対応できるのか、数値化する必要がありました。

そのために、バックグラウンダーのみを実行するための、物理コアを 4 つ搭載したコンピューターを用意しました。そのコンピューター上では、他の Tableau Server プロセスは一切実行しませんでした。同じ実稼働環境をモデル化したワークロードを Tableau Server のバージョン 9.3 と 10.0 で実行しました。このワークロードには抽出の更新とサブスクリプションを含め、ユーザーレベルのジョブにフォーカスできるようにしました。ワークロードには、8 つのスケジュールに従って 400 回のサブスクリプションが含まれていました。サブスクリプション通知の正常な完了と、Tableau がすべてのサブスクリプションの処理を完了するまでにかかった時間を調査しました。

標準化した隔離環境

私たちは、自社のパフォーマンスラボで、次のような仕様の同一の物理マシンを使って、これらのスケーラビリティテストを実行しました。

サーバータイプ	Dell PowerEdge R620
オペレーティングシステム	Microsoft Windows Server 2012 R2 Standard 64 ビット
CPU	2.6 GHz x 8 コア (物理コア)、ハイパースレッディング有効 (論理コア 16 コア)
メモリ	64 GB

表 3: テスト環境の各ノードのハードウェア仕様

この表には物理コア数を記載していますが、ハイパースレッディングは有効にしておくことをお勧めします。一貫性を保つために、ここでの記述を除いて、このホワイトペーパーでは物理コア数を使用し、物理マシンでは必ずハイパースレッディングが有効になっているものとします。

導入トポロジ

クラスタは、1 つ以上のプライマリ (コントローラ) ノードと 1 つ以上のワーカーノードで構成されます。このテストでは、ワーカーノードのプロセス構成プロファイルは同じものを使用しています。

プロセス	プライマリ tsperf-212.perf.dev.tsi.lan	ワーカー 1 tsperf-213.perf.dev.tsi.lan	ワーカー 2 tsperf-215.perf.dev.tsi.lan	ワーカー 3 tsperf-216.perf.dev.tsi.lan	ワーカー 4 tsperf-217.perf.dev.tsi.lan
クラスタコントローラ	✓	✓	✓	✓	✓
ゲートウェイ	✓	✓	✓	✓	✓
アプリケーションサーバー		✓	✓	✓	✓
VizQL サーバー		✓ ✓	✓ ✓	✓ ✓	✓ ✓
キャッシュサーバー		✓ ✓	✓ ✓	✓ ✓	✓ ✓
検索とブラウズ		✓	✓	✓	✓
バックグラウンダー		✓	✓	✓	✓
データサーバー		✓ ✓	✓ ✓	✓ ✓	✓ ✓
データエンジン		✓	✓	✓	✓
ファイルストア		↻ 同期	↻ 同期	↻ 同期	↻ 同期
リポジトリ	✓				

更新ステータス　✓ アクティブ　↻ ビジー　✓ パッシブ　⚠ ライセンスなし　✕ ダウン　□ ステータス不明

図 4: プロセス構成

ここに記載されているワーカープロセスの構成は、既定の構成です。その環境で構成したプロセスの数や種類、および利用シナリオによって、得られるスケーラビリティは異なる場合があります。

プライマリノードは、クラスタコントローラ、ゲートウェイ、リポジトリのプロセスを含む、基本的なインストールで構成されています。コアライセンススキームで導入する場合、このタイプのプライマリノード構成はコアとしてカウントされないことに注意してください。

負荷生成ツール (TabJolt) を使用して、前述のユーザーワークロードのシミュレーションを行うことで、ワークロードをスケールしました。TabJolt は、Tableau Server 9.0 以降のために特別に設計された、「ポイントアンドラン」で利用できる、負荷およびパフォーマンスのテストツールです。次の図は、テストの実行環境を論理的に示したものです。

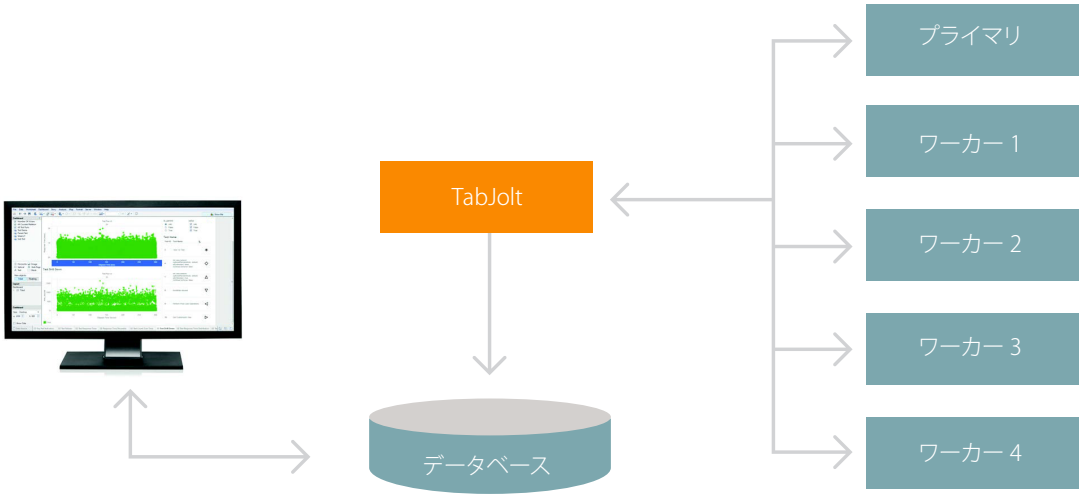

図 5: テスト環境の論理図

このテストを繰り返し実行することで、分析するデータを収集しました。ここで、その結果を説明する前に、いくつかの指標と定義を確認しておきましょう。

測定とレポート作成

ハードウェアのパフォーマンスおよびスケーラビリティを理解するために、CPU、メモリ、ディスクのシステム指標を含む、いくつかの指標を測定しました。また、応答時間、スループット、エラー率、実行時間などの、パフォーマンスおよびスケーラビリティに関する指標も測定しています。

このホワイトペーパーで説明するデータをよりよく理解していただけるよう、いくつかの定義を簡単にまとめました。

シナリオ

シナリオとは、サーバーで実行される最上位のユーザーアクティビティです。以前のバージョンのホワイトペーパーでは、サーバーでゲストアクセスが有効になっている場合のビジュアライゼーションの読み込みと操作の時間に焦点を当てていました。今回のバージョンでは、ワークロードを拡張して、アプリケーションサーバーサービス (ログインなど) およびその他のサービスも含めています。エンドユーザーは、実稼働環境のワークロードのモデル化に基づく一連のステップを実行します。カスタマイズした TabJolt を使用してシミュレーションを行いました。

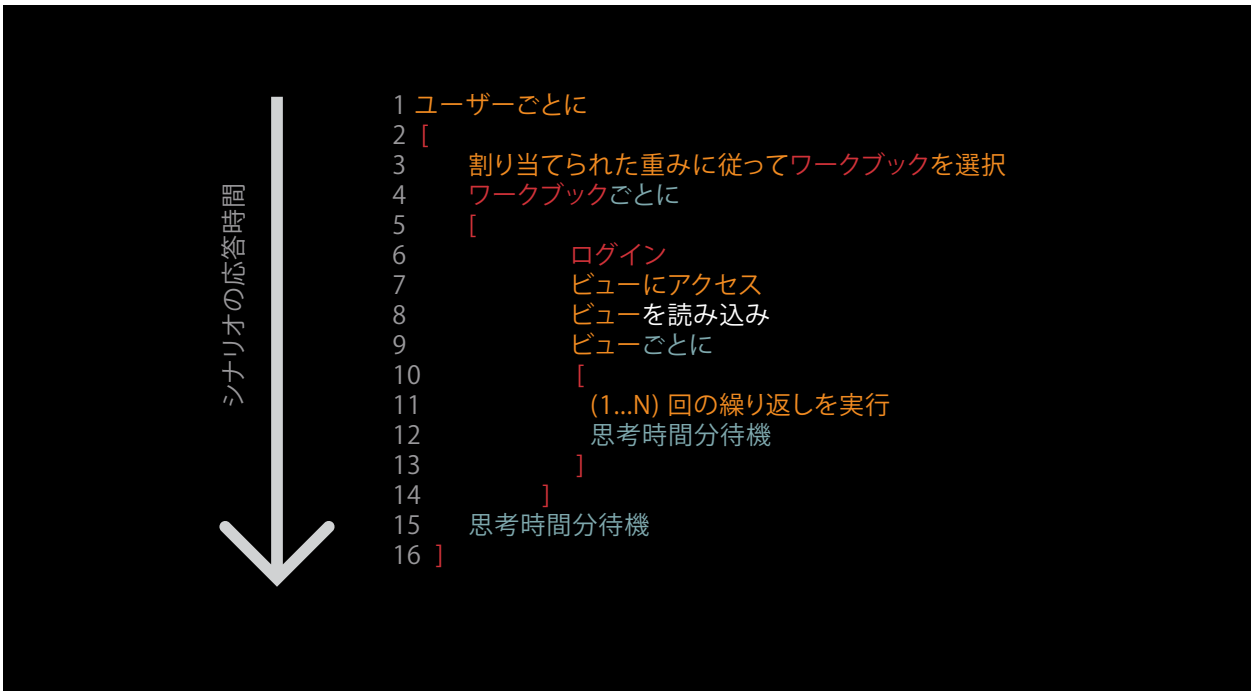

図 6: シナリオをテストするアルゴリズム

このテストモデルではシナリオの実行に必要な時間が含まれているため、シンプルな読み込みと操作のモデルと比較すると、このホワイトペーパーに記載されている応答時間は長くなっています。クライアントの視点から見たトランザクションの応答時間を測定し、記録しています。つまり、遅延などのネットワーク環境による変動も含めた、実際のエンドユーザーの視点から見たシナリオとなっています。

応答時間

応答時間は、サーバーがエンドユーザーのリクエストに応答するために要する秒数として測定されています。たとえば、ユーザーがサーバーにサインインし、ビジュアライゼーションを開き、そのビジュアライゼーションのフィルターを変更し、ビジュアライゼーションが更新されてレンダリングされるのを待ち、ビジュアライゼーションを分析する (思考時間) とします。この場合、ユーザーがサーバーにログインしてからユーザーの思考時間が終了するまでの実際の合計時間が、そのシナリオ 1 回分の応答時間として記録されます。

シナリオのスループット

シナリオのスループットは、1 秒間にシナリオが正常に完了した回数 (TPS) です。特定のトポロジで、システムの処理能力を限界まで使って 1 時間シナリオを実行することで、シナリオの TPS を計算しました。シナリオの合計数を 1 時間の秒数 (3,600 秒) で割ったものが TPS となります。

たとえば、8 コアの Tableau Server 10 でテスト時間いっぱいシナリオを実行したところ、16,372 回シナリオの実行が完了したとします。この場合の TPS は約 4.5 となります (16,372/3,600)。この実験では、TPS が 1 未満だった、実稼働環境で観察したよりもはるかに高い負荷をサーバーにかけました。

実験データでの大幅な増加は、対象のハードウェアおよびスケールユニットのパフォーマンスの上限を理解するために、サーバーの限界まで負荷をかけたことが原因です。

ただし、実稼働環境への導入では、使用状況のピーク時に負荷の急激な増加が発生しており、ユーザーは通常、自動的なテスト実行のフレームワークよりもずっと動作が遅くなります。この違いを考慮すると、たとえば Tableau Public ではかなりの規模に対応でき、1 秒あたり約 12 までのビジュアライゼーションの読み込み (「インプレッション」と呼びます) が可能で、これは、1 週間あたり 700 万インプレッション以上の Tableau のビジュアライゼーションに対応できることになります。

アクティブユーザー

アクティブユーザーは、ピーク時の 1 時間の枠内に Tableau Server を同時に使用するユーザー数を測定する指標です。シナリオには、ログイン、ビジュアライゼーションの読み込み、ユーザーの操作、検索、その他のアクションが含まれます。ここでは、ピーク時の 1 時間の枠内にこれらのいずれかのアクションを実行しているユーザーをアクティブユーザーと定義します。

導入したサーバーで対応可能なアクティブユーザー数を確認するために、まず、応答時間の低下、2% を超えるエラー率 (Tableau Server の HTTP エラー)、または 80% を超える CPU 使用率を発生させることなく、単一のサーバーで維持できるユーザー数を特定しました。

実稼働環境のワークロードの特徴に基づいて、特定のワークブックに重みを割り当てました。割り当てられた重みに基づいてワークブックをランダムに選択するスレッド (仮想ユーザー) を生成し、前述のシナリオ全体を完了しました。シナリオの最後に、スレッドは、指定された思考時間の分だけ待機状態になります。この時間が経過すると、繰り返し実行が完了し、そのスレッドは終了します。このテストでは、さまざまな指標の中でも特に、シナリオのスループット、応答時間、エラー率、CPU およびメモリの使用状況を測定しました。CPU 使用率が 80% 未満で、エラー率が 2% 未満、かつ応答時間が低下しない範囲でアクティブなスレッド数を増やし続けました。このいずれかのしきい値に達した場合、そこが、そのトポロジで合理的に対応できるアクティブユーザー数のスイートスポットであると考えました。

サーバーの直線的なスケーリングが可能であることを確認するために単一マシンのスイートスポットを特定し、次に、仮想ユーザー数を直線的に増加させて、新しく負荷がかかっても事前に設定した条件が満たされることを確認しました。

結果

テストの実行方法、使用した導入方法、および指標を確認したところで、次は結果を見てみましょう。

Tableau Server 10 は直線的なスケーリングが可能

まず確認したかったことは、Tableau Server 10 のスケーラビリティです。ユーザーの負荷が増大している状態で、クラスタにワーカーノードを追加することで、負荷に対して Tableau Server 10 が直線的にスケールすることを観察しました。下のグラフは、ワーカーを増やすことで 1 秒あたりに完了できたユーザーシナリオの数を示しています。

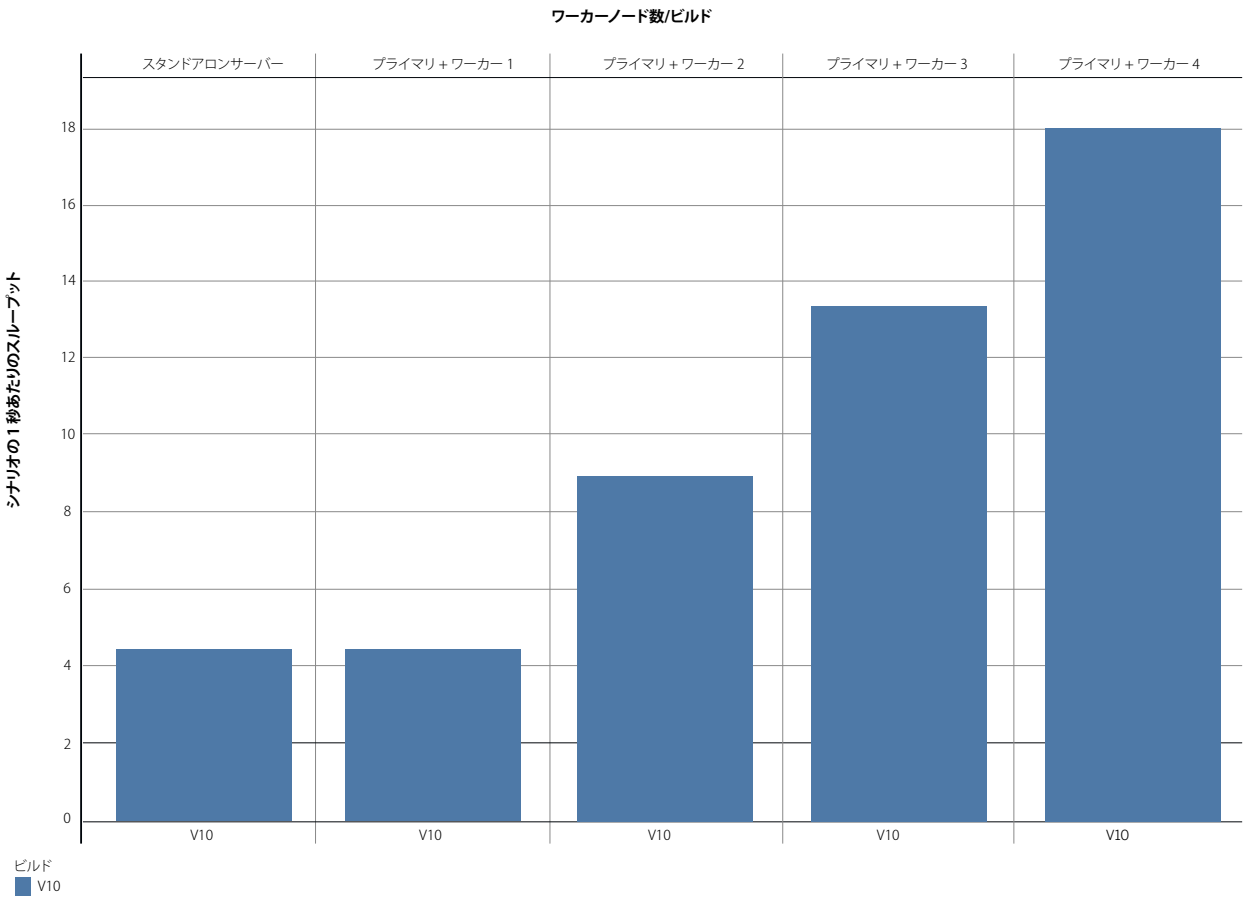

図 7: シナリオの 1 秒あたりのスループット

各列はクラスタのトポロジに対応しています。左端は、単一サーバーの構成です。2 つ目は、プライマリ/ワーカーのクラスタ構成で、ワーカーは 1 つです。残りの列では、前述の通り、ワーカーノードを 1 つずつ追加しています。棒の高さは、シナリオの 1 秒あたりのスループット (TPS) の平均を示しています。TPS は、サーバーが対応できる処理量を表します。ご覧のように、ワーカーノードを追加することで TPS が直線的に上がっています。クラスタに対する負荷を上げて観察したところ、クラスタ全体での CPU 使用率は平均でおよそ 80% でした。次のグラフは、負荷を上げた場合のクラスタの CPU 使用率と 95% 信頼区間を示しています。

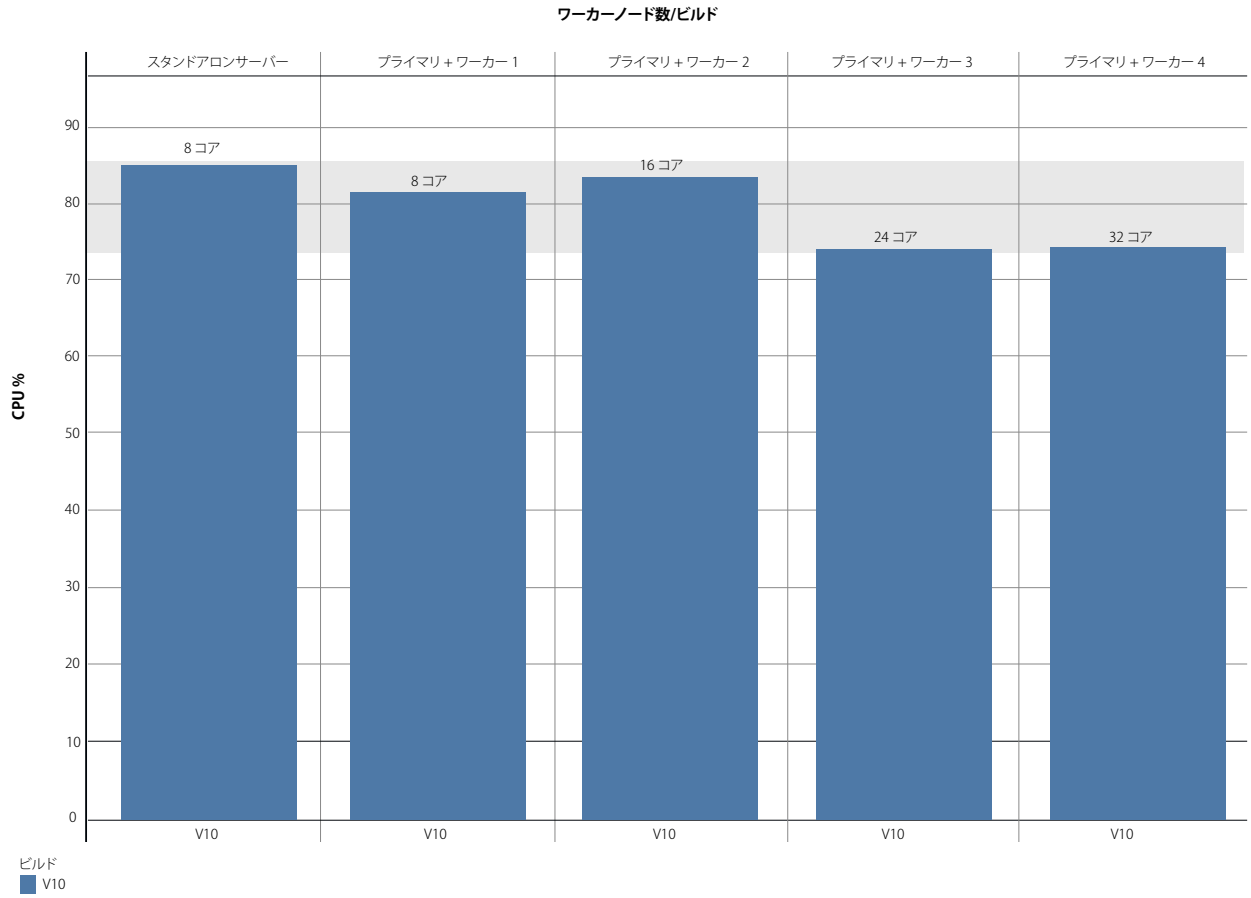

図 8: 負荷を上げた場合のクラスタ全体の CPU 使用率

図 8 が示すように、ワーカーノードを増やしてクラスタ内の CPU 負荷を分散させると、負荷の急激な増加にも対応できる余裕ができ、システムが最適化されます。クラスタ内で構成しているワーカーノード数が少ない場合、ワーカー数が多いクラスタと比較すると CPU 使用率が高くなります。ワーカーノード数が少ない場合、計算処理を要求するプロセスが限られたリソースを奪い合うことになります。このテストでサーバーのエラー率を観察したところ、このテスト方法で設定した 2% という目標に収まっていました。クラスタのマシン数または容量あるいはその両方が限られている場合、ワークロードによっては、エラー率が高くなる (サービス品質が低下する) ことがあります。

次に確認したかったことは、同じテスト方法で Tableau Server 10 と Tableau Server 9.3 を比較するとどうなるかということです。

それぞれのバージョンで同じ条件のテストを実施した結果、Tableau Server 10 のスループットは、Tableau Server 9.3 と比較して向上しました。

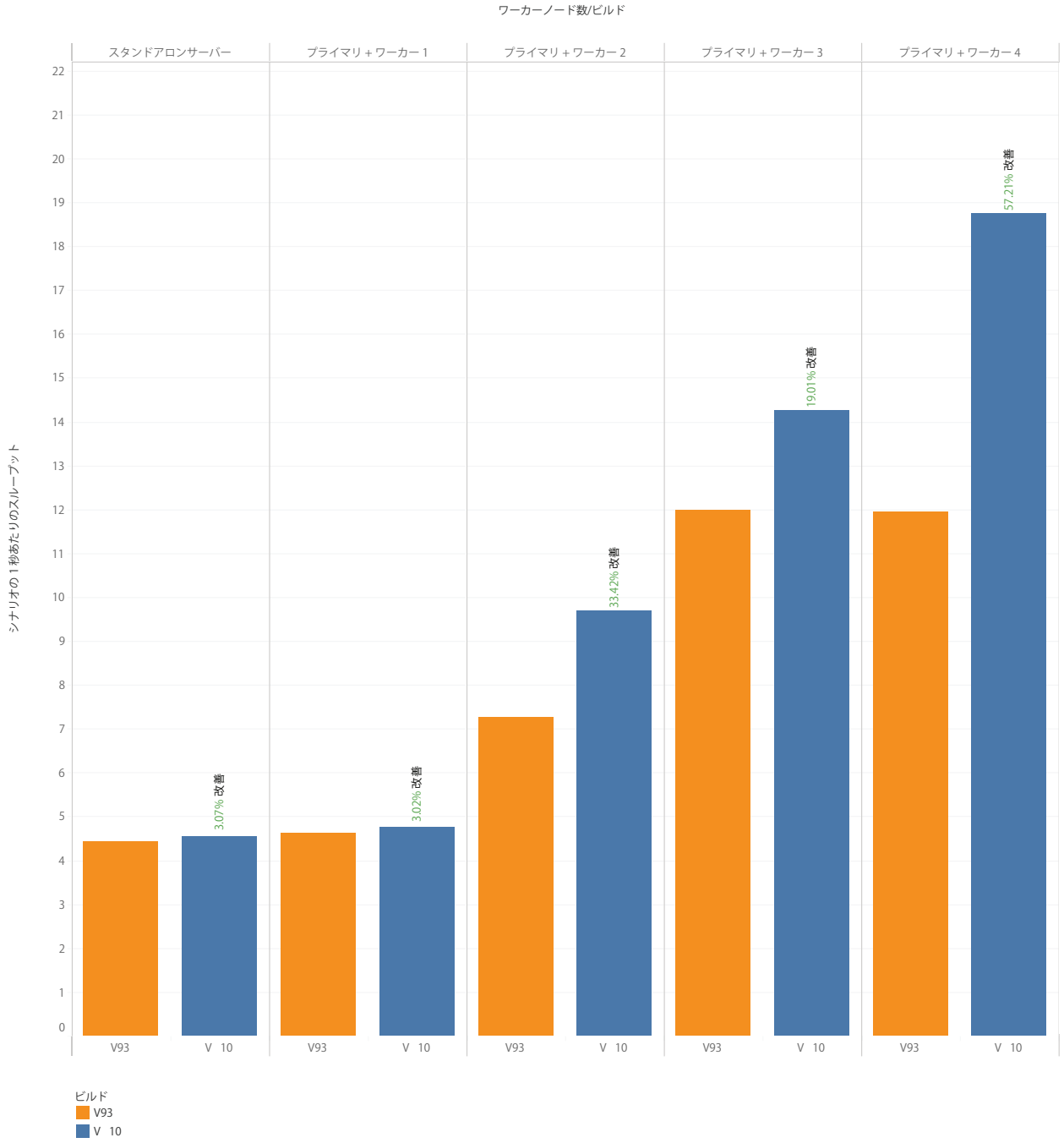

図 9: シナリオの 1 秒あたりのスループットの比較: 9.3 と 10.0.1

図 9 では、Tableau Server 9.3 のシナリオのスループット (オレンジの棒) と、Tableau Server 10 のシナリオのスループット (青の棒) を比較しています。各ペインには、列の見出しに説明されるクラスタトポロジの構成が表示されています。クラスタにノードを追加すると、Tableau Server 10 は直線的なスケーリングが可能であることは前述の通りですが、さらに、Tableau Server 9.3 と比較してシナリオのスループットも向上しています。

バージョンを比較する際、適切な比較結果を得るためには、現実的なユーザーシナリオを使用する改善されたテスト方法で 2 つのバージョンを比較するしかありません。この理由で、Tableau Server 10 の結果を以前のホワイトペーパーの結果と比較するのは、テスト方法が大きく変わっているため、適切な比較となりません。

スループットを観察することも重要ですが、エンドユーザーの応答時間について、テストシナリオ全体を通じて適切なパフォーマンスを発揮し、負荷が上がってもパフォーマンスが低下しないことを確認したいと考えました。

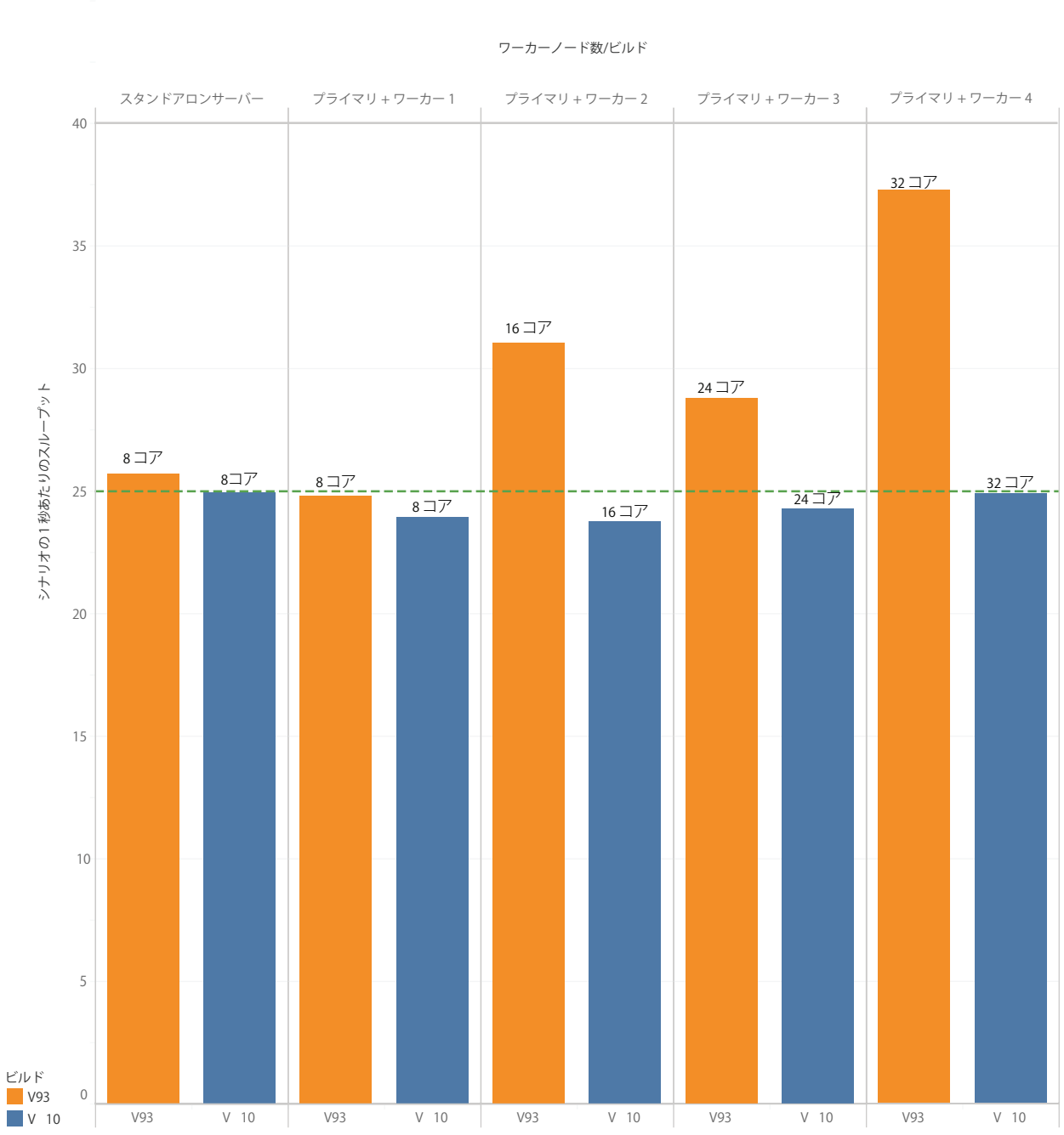

図 10: シナリオの応答時間 (秒) の平均値の比較: 9.3 と 10.0.1

図 10 では、オレンジの棒の高さが、バージョン 9.3 のシナリオの応答時間の平均を示しています。青の棒の高さは、バージョン 10.0.1 での同じ指標を示しています。バージョン 9.3 をホストしているクラスタでエンドユーザーの負荷を高くしていくと、応答時間も順次長くなっていきました。これに対して、Tableau Server 10 では、システムに仮想ユーザーを追加し続けても、比較的一定した応答時間が得られました。これは、同じワークロードで Tableau Server 9.3 と比較した場合に、Tableau Server 10 では、エンドユーザーにとってのパフォーマンスと応答性が向上していることを示しています。

負荷を増加させても、比較的一定で安定した応答時間が得られるなど、Tableau Server 10 で確認できたいくつかの改善点は、キャッシュコンポーネントに対して加えられた改善の成果です。具体的には、バージョン 10 では、ブートストラップの応答のシナリオ向けにキャッシュが最適化されています。ブートストラップのシナリオは、初期化してユーザーセッションのデータをキャッシュするために行う最初の呼び出しです。以前のバージョンの Tableau Server では、キャッシュがなかったので、後続のリクエストが同一であったりきわめて近いものであったりしても、毎回ブートストラップデータを計算し、保存していました。バージョン 10 では、このような場合にスマートなキャッシュ処理が実行されます。このような改善によって、適切な応答時間を維持しながら、より多くのユーザー負荷を処理することができるようになっており、Tableau Server 10 の効率性が向上しています。ただし、サーバーの処理量は増えており、先ほど確認したように、スループットが増加しています。

ここまでに確認してきた結果はすべて、エンドユーザーによる分析の利用が増加した場合に Tableau Server のスケーラビリティがどうなるかを示すものでした。サーバー上のエンドユーザーのワークロードが増加した場合、エンドユーザーに質の高いサービスを提供するためには、クラスタにワーカーノードを追加して、十分な容量を確保する必要があります。

次に、バックグラウンダーをどこにどのように構成するかによって、ユーザーによる影響がどうなるかという質問があります。具体的には、バックグラウンダープロセスを分析サービス (VizQL サーバー) と同じコンピューターでホストした場合と、別のコンピューターにバックグラウンダープロセスを分離した場合を比較して、ユーザーによる影響を調べました。次のセクションで、この結果について説明します。

バックグラウンダーに関するテスト結果

まず、既定の単一サーバーのインストール環境でバックグラウンダーを実行した場合の影響を数値化しました。このシナリオで、エンドユーザーに対する影響と全体的なスケーラビリティはどうなるかを調べました。単一の Tableau Server 導入環境でワークロードの実験を実行し、TPS を記録しました。さらに、ワーカーを追加してクラスタトポロジを構築し、クラスタでワークロードのテストを行いました。

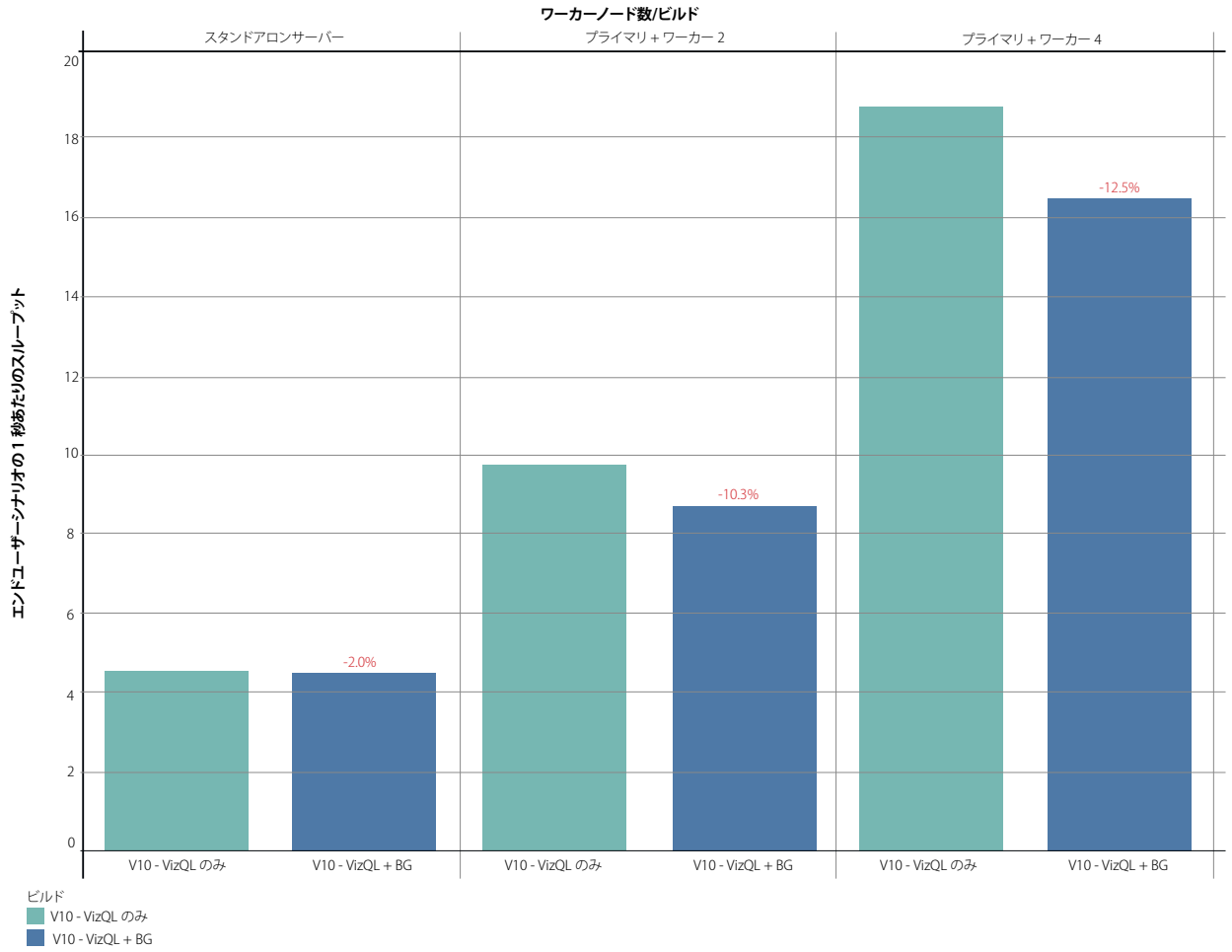

図 11: バックグラウンダーと VizQL サービスの共存による、エンドユーザーのスケーラビリティへの影響

図 11 は、2 つのテストシナリオの結果を示しています。緑の棒は、VizQL のみのワークロードを実行したテストシナリオを示しています。VizQL のみのワークロードでは、エンドユーザーのワークロードをシミュレートしています。青の棒は、同じクラスタで VizQL のワークロードに、さらに固定量のバックグラウンダーワークロードを追加したテストシナリオを示しています。この 2 つのシナリオで、エンドユーザーのワークロードの TPS の変化を記録しました。VizQL サーバーとバックグラウンダーのどちらも大量のコンピューティング処理が必要なワークロードであるため、このシナリオでは影響が見られることはわかっていましたが、ここでの意図は、特定のワークロードについてこの影響を測定することでした。

エンドユーザーシナリオの全体的なスループットには、2 ～ 12% の低下が見られました。この低下は、バックグラウンダーのワークロードを処理するコストにより、クラスタがエンドユーザーにサービスを提供する能力が影響を受けたことを示しています。このデータから推定すると、クラスタで単位時間に 100 のエンドユーザーシナリオを処理していた場合、一定数のバックグラウンダー負荷を追加すると、同じ単位時間に処理できるユーザーシナリオ数が 88 に減少することがわかります。このスループットの低下により、ワークロード、急激な負荷の上昇、ハードウェアの制限、インフラストラクチャにおける可変要素など、さまざまな要素によっては、エンドユーザーのスケーラビリティまたはサービス品質、あるいはその両方に大きな影響が及ぶ可能性があります。

このテストは、Tableau Server のワークロード計画に応じて、適切なハードウェアを用意することの重要性を示しています。性能が低い、あるいは限られたハードウェアで Tableau Server を実行すると、パフォーマンスの問題として、スループットの低下、バックグラウンダージョブの失敗、サブスクリプション通知の遅延、エンドユーザーのエラーなどが生じる可能性があります。このような現象が発生した場合は、クラスタを拡張して、バックグラウンダーのワークロードを専用のハードウェアに移すことを検討してください。計算処理を必要とするプロセスを同じコンピューター上で共存させると、同じリソースを奪い合うことになりますが、バックグラウンダーサービスを分離することで、そのプロセスを競争から解放することができます。

それでは、バックグラウンダーが VizQL サーバーと共存している場合と、専用のマシンに分離した場合のサブスクリプションのワークロードへの影響を見ていきましょう。

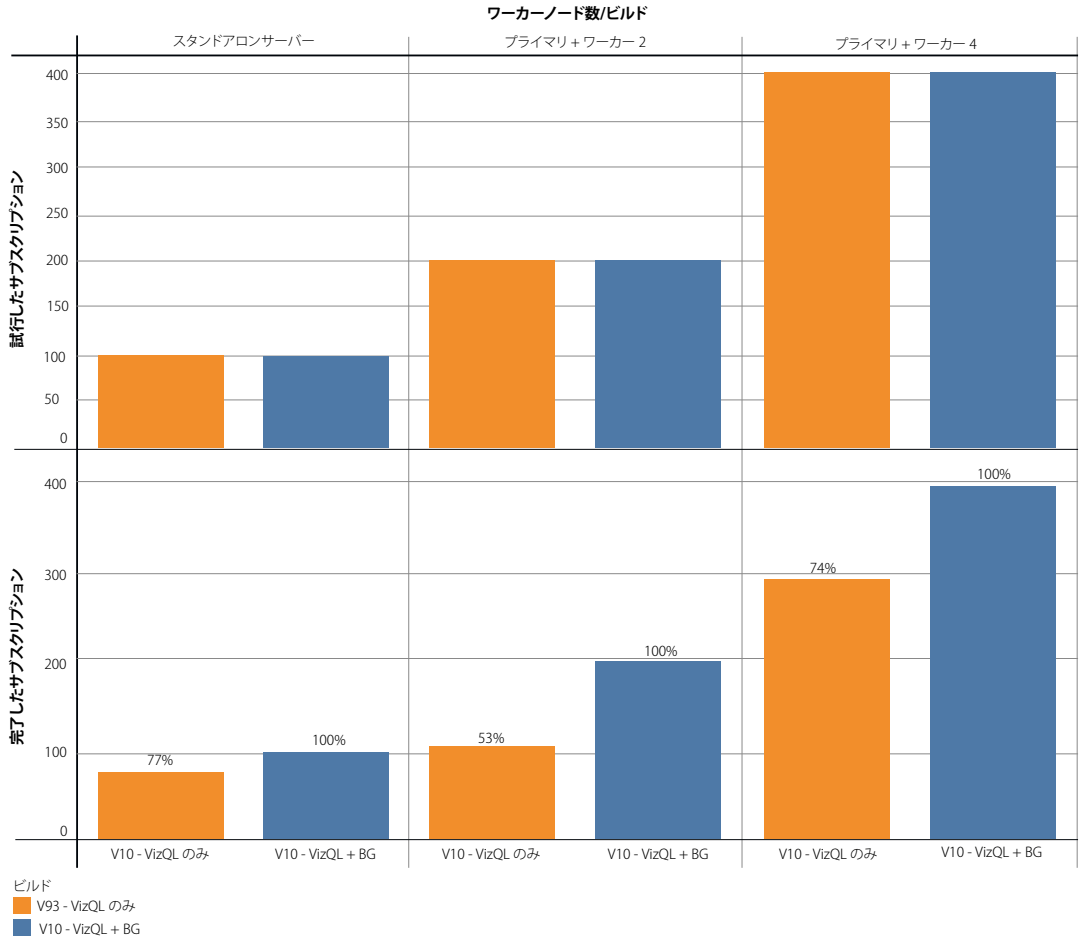

図 12: サブスクリプション通知における機能強化

図 12 の各ペインは、Tableau Server クラスタのトポロジごとに分かれており、各棒は、サブスクリプション通知の試行数と、正常に完了したサブスクリプション通知の数を示しており、オレンジがバージョン 9.3、青がバージョン 10 となっています。テストごとに、ワーカーノードの数を 0、2、4 と増やし、ワーカーノードごとにバックグラウンダープロセスを 1 つだけ構成しました。(ワーカーノード数 1 と 3 のテストを省略したことで、テスト結果の正確性が損なわれることはありません。)比較可能な一貫した測定結果が得られるように、1 つのワーカーノードに対するバックグラウンダープロセスは 1 つに制限しました。

バックグラウンダーのサブスクリプションの負荷を増加させても、Tableau Server 10 では、実行されたすべての処理を完了させることができました。同じテストで、Tableau Server 9.3 では負荷が過多になり始め、一部のサブスクリプションが失敗しています。リソースが不足していたり負荷が高すぎたりするトポロジでは、この現象がさらに悪化する可能性があります。さらに、Tableau Server 10 では、システムのワーカーノードを増やした場合、さらに多くのサブスクリプションに対応できることが確認できました。Tableau Server 10 での機能強化で、このような結果をもたらしたのは、通知のイメージキャッシュの導入に関する部分です。この機能により、バックグラウンダーが処理する、スケジュールに関するタスクで重複するものについては、処理量が大きく減少します。同じスケジュールで同じワークロードが実行されるシナリオの場合、Tableau Server 10 では、最初の実行時の結果がキャッシュされ、同じワークロードのそれ以降のリクエストではその結果が利用されます。これにより、同じ処理をより効率的に実行できるようになります。

同じスケジュールで同じワークブックを使用するサブスクリプションでは、そのサブスクリプションの実行に必要な時間に、バージョン 9.3 と比較して 60 ～ 90% の短縮が見られました。図 13 は、この改善を示しています。

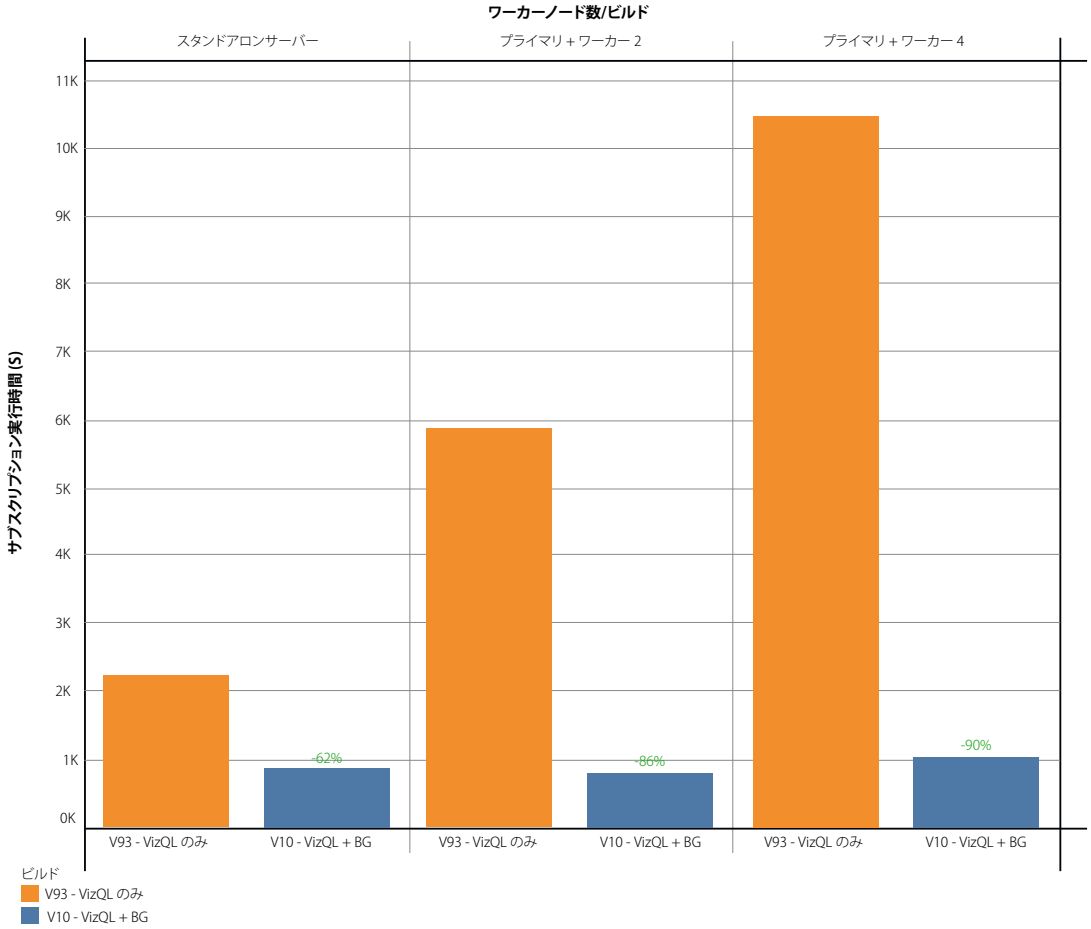

図 13: サブスクリプション実行時間 (実行完了までの所要時間): 9.3 と 10

これは、同じスケジュールで同じワークブックに対して実行される通知では、実行の所要時間がはるかに短くなることを意味しています。ただし、通知の対象のワークブックが異なっていたり、ユーザーフィルターが設定されていたりすると、その通知をユーザーに届ける前にサブスクリプションの処理に時間がかかります。異なるワークブックやユーザーフィルターを使用すると、Tableau Server でエンドユーザーのビジュアライゼーションのパイプライン全体を実行する必要があります。これには、各ワークブックやフィルターされたビューのデータのクエリやデータのビジュアル処理などが含まれます。

サブスクリプションのメリットは、ビジネスユーザーが関心を持っているデータをタイミングよく提供できることです。抽出の更新を効果的に実行することで、最新のデータに基づいて適切な判断を行うことができます。バックグラウンダープロセスでは、これらの重要な機能の両方を管理しているため、重複する処理が同じスケジュールに結び付けられるようにサブスクリプションのスケジュールを設定すると、キャッシュのメリットが得られます。

バックグラウンダープロセスの分離

バックグラウンダープロセスを専用のワーカーノードに分離した状態で、同じサブスクリプションの実験を行いました。このバックグラウンダーワーカーノードでは、4 コア搭載の単一のマシンで 400 のサブスクリプションを実行することができました。これは、4 つのワーカーマシンで各 VizQL ワーカーと単一のバックグラウンダープロセスを共存させた場合の記録と同じサブスクリプション数です。

ここで明らかになった重要なことは、バックグラウンダープロセス自体をスケールすることができますが、プロセスを分離することで同様のスケーラビリティが得られるため、エンドユーザーのサービス品質に影響しないということです。バックグラウンダープロセスはシングルスレッドのプロセスで、可能な限り迅速にジョブを実行するように設計されています。このような設計になっているため、バックグラウンダープロセスは処理すべきワークロードがあればコアを使用し尽くします。マシンを分離した場合、バックグラウンダープロセスによって多量の計算処理が発生しても、ユーザー向けの他の Tableau サービスが妨げられることはなくなります。

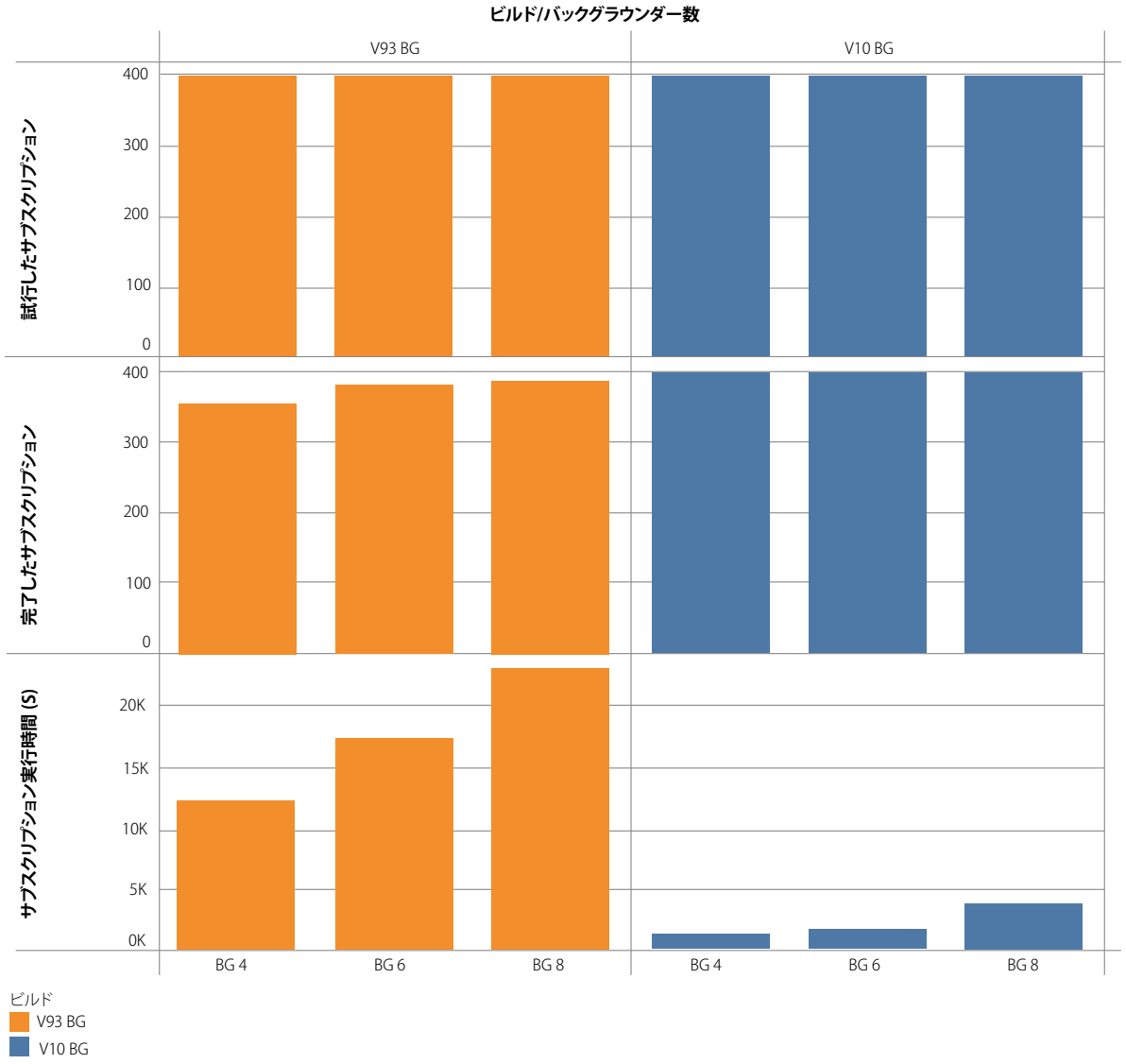

図 14: 4 コア搭載のコンピューターに対するバックグラウンダープロセスの追加: 9.3 と 10

図 14 からわかるように、4 コア搭載の単一のコンピューターにバックグラウンダープロセスを追加すると、オレンジの Tableau Server 9.3 と比較して、青の Tableau Server 10 では、はるかに短時間でサブスクリプションの処理を完了できました。ただし、物理的にコア数の制限のあるコンピューターにバックグラウンダープロセスを追加し続けると、マイナスの影響が及びます。4 コアという制限のあるコンピューターに 8 つのバックグラウンダープロセスを追加すると、完了までの所要時間が長くなっています。これは、バックグラウンダーがシングルスレッドのプロセスであるためです。

最後に、バックグラウンダープロセスを分離して実行しているワーカーノードで抽出の更新を実行するテストを行ったところ、Tableau Server 9.3 と 10 は同等という結果が出ました。どちらのバージョンでも、同数の抽出の更新が、ほぼ同じ時間で処理されていました。抽出の更新を行うためのスケーリングで考慮すべき重要なことは、抽出の更新で適切なパフォーマンスを実現するには、外部のデータベースが大きく影響するということです。(このテストでは、Microsoft の SQL Server からパブリッシュされた抽出を使用するワークブックを使ってデータを更新しました。) 抽出の更新のパフォーマンスおよびスケールは、データベースのハードウェアの仕様に大きく依存します。さらに、

結合の種類や実行されるクエリの複雑度など、データの特性がスケールに影響します。このため、そのシステムで予想されるエンドユーザーによる負荷のピークが来る前に抽出の更新や通知の処理を完了するのに十分な容量を確保する必要があります。

バックグラウンダーの考慮事項

バックグラウンダープロセスは、抽出の更新、サブスクリプション、スケジュールが設定されているその他のバックグラウンドジョブに関連する作業の多くを実行します。これらのジョブをピーク時以外に実行するようにスケジュールを設定すれば、容量で競合することはありません。これが難しい場合は、バックグラウンダーおよびユーザーに直接関係のない他のワークロードを、ユーザーに直接関係のあるプロセスと同時に実行するために必要な容量を追加することを計画してください。

バックグラウンダーはできるだけ早く作業を終了するように設計されているため、プロセスごとにコアのすべての容量を消費します。バックグラウンダープロセスを複数実行する場合、バックグラウンダープロセスは、同じマシンで実行されている他のサービスと計算リソースやネットワークのリソースを奪い合う可能性があることを考慮する必要があります。

ベストプラクティス - 自分で行うスケールテスト

自分の環境とワークロードで Tableau Server がどのようにスケールするかを確認する負荷テストを行う場合、次のようなベストプラクティスが役立ちます。

1. **Tableau Server をブラックボックスとして扱わない。**従来の負荷テストでは、対象のアプリケーションをブラックボックスとして扱うことがよくありました。その場合、負荷の条件に応じて導入環境の調整や構成変更を行わないことが前提となります。Tableau は、スケールアップやスケールアウトを行えるように設計されているため、Tableau のアーキテクチャを理解して、具体的な状況に応じて成果が出るように、スケーラビリティテストにその情報を反映させることが役立ちます。

2. **テストに適したツールを選ぶ。**Tableau Server は多機能であり、複雑かつリソース集約的な作業を行います。Tableau Server に負荷をかけるためのツールも数多くあります。Tableau は直接これらのツールをサポートしていませんが、最も使いやすく、本番環境に最も近いものを選択してください。もう 1 つ考慮すべきことは、負荷テストを実施する際に利用するツールと Tableau Server に関する適切な専門知識を、テストの担当者が持っている必要があるということです。ここでは、テストに Tabjolt を使用しました。TabJolt は、JMeter をベースとしており、指定して実行するだけで利用できる負荷テスト用ツールです。スクリプトのメンテナンスが不要で、Tableau のようにアドホックな分析ソリューションで利用できます。

3. **適切なワークブックを選ぶ。**多くの場合、パフォーマンスやスケーリングに関して問題が生じるのは、ベストプラクティスを念頭に置いて作られていないワークブックを使用しているためです。ワークブックに対するシングルユーザーテストで応答時間が非常に遅くなっている場合は、負荷テストプロジェクトを始める前にワークブックを最適化する必要があります。実際の稼働環境では、パフォーマンスの良くないダッシュボードをそのまま使用し続けることはありません。テストでも同じようにしてください。

4. **既定の構成から始める。**ライブ接続を使ってワークブックをテストする場合、Tableau Server 9.0 で並列処理が導入されたことにより、必要な VizQL サーバーの数は、それ以前のバージョンの Tableau Server の場合と比較して、少なくなることがあります。新しい 2 つのプロセスの既定の構成から始めて、必要に応じて、順次スケールアップしていきます。

実稼働環境における最適化のベストプラクティス

最適に設計されたシステムに加えて、パフォーマンスを大幅に向上させ、平均応答時間を削減することができるベストプラクティスがあります。

- ワークブックの設計では、外観とパフォーマンスを考えます。ワークブックの処理に時間がかかるのは、パフォーマンスを念頭に置いて設計されていないからだというお客様の声をよく耳にします。1 人のユーザーで読み込みに時間がかかっていれば、負荷が高い場合にも、ワークブックの応答時間は遅くなります。分析の文化を浸透させるのと同時に、見た目も美しく、インサイトに溢れ、パフォーマンスにも優れたワークブックを設計できるようユーザーをサポートできるようにしておくことで、スケーラビリティに優れたビジュアライゼーションを構築し、提供できるようにもなります。「効率的に作業できるワークブックの設計」というホワイトペーパーでは、効率的でパフォーマンスに優れたダッシュボードの構築について詳しく説明しています。

- エンドユーザーから見た合計の応答時間には、さまざまな要素が含まれますが、その中でも大きなものは、Tableau の処理時間とデータ取得にかかる時間です。バックエンドのデータベースが遅い場合や、クエリに時間がかかる場合、ビジュアライゼーションにも時間がかかります。このことをデータ戦略に織り込んでおくことが重要です。組織内のデータソースがまとめられ、共有されていることは少なくありません。データがビジネスユーザーの生産的な活動に役立つような方法で、重要なデータを提供できるようにする必要があります。つまり、データの最適化です。たとえば、インデックス化されたテーブルに対して高速にクエリを実行できるように、最適な結合や適切なレベルの集計を確実に行う必要があります。ビジュアライゼーションのパフォーマンスを維持するうえで、データを最適な状態に保つためのプロセスを確立しておくことが重要です。

- Tableau データ抽出を使用します。データベースクエリが遅い場合は、抽出を使用して、クエリのパフォーマンスを向上させることを検討してください。抽出はサーバー上にローカルに保存され、インメモリで実行されるので、データベースにリクエストを送信しなくても迅速にデータにアクセスできます。抽出では、フィルター処理も集計も簡単に行うことができ、行レベルの詳細情報が必要でない場合には最適です。抽出を使用すると、応答時間が大幅に改善されるので、ユーザーは分析をスムーズに進めることができます。

- ピーク時を外して更新を行うようにスケジュールを設定します。データソースは多くの場合、リアルタイムで更新され続けていますが、ユーザーに必要なのは日次または週次のデータのみということがあります。オフピーク時間に抽出のスケジュールを設定すれば、データベースと Tableau Server の両方に対するピーク時の負荷を軽減できます。さらに、十分なコア容量がある場合は、既存のマシンにバックグラウンダーを追加したり、専用のハードウェアを使ったりすることができます。抽出をより早く終える場合は、この方法を検討してください。

- ピーク時には負荷の高い処理は避けるようにします。パブリッシュでは、特にサイズの大きなファイルの場合、非常に多くのリソースが消費されます。たとえば月曜日の朝などのように多忙な時間帯を避けて、ピーク時以外にパブ

リッシュを行うようにユーザーに働きかけることで、パブリッシュの影響を低減することは比較的簡単にできます。サーバーが最も使用されているタイミングを知るには、管理ビューを使用して、実際の使用状況に基づいてポリシーを作成します。Tableau Server 10.0 をどのように構成したかによっては、可用性を高めるために、パブリッシュ時に抽出のコピーが各クラスタノードに作られる場合があります。ピーク時以外にこれを行うことにより、ネットワークの帯域幅を最大限に活用できます。

- ビューのキャッシュを行います。複数のユーザーが Tableau Server へのアクセスを開始すると、最初は共有リソースを求める競合により応答時間が長くなります。キャッシュを有効にすると、システムへの各リクエストからのビューがキャッシュされ、同じダッシュボードの次のユーザーが同じビューをレンダリングするときに表示時間が短縮されます。

- Tableau Server 9.0 で導入されたキャッシュサーバープロセスは、抽出の更新が終わってから一般的なビューのメールのスケジュールを設定することによってウォームアップできます。このようにして、その後、ビューを見る人は、以前のリクエストからキャッシュされたデータを使用します。他の方法を使ってキャッシュをウォームアップすることもできます。たとえば、自動化ツールを使って、定期的に大きなトラフィックを生む主要なビジュアライゼーションを読み込む方法です。ユーザーはいつでも手動で外部クエリのキャッシュを無効にして、データソースからデータを更新することができます。この操作により、強制的にキャッシュの再作成も行われます。このようにして、最新バージョンのデータがすでにキャッシュにあるかどうかにかかわらず、ユーザーはいつでも最新バージョンのデータを取得することができます。

まとめ

Tableau Server 10 は、エンタープライズレベルで、あらゆる規模の組織に対応できる、スケーラビリティに優れたプラットフォームです。オンプレミスでも、プライベートクラウドでもパブリッククラウドでも実行することができ、ワーカーを増やすことで直線的なスケーリングが可能です。環境によってそれぞれ独自の特徴や構成がありますが、Tableau Server のアーキテクチャは、ユーザーの求めるニーズに応じて導入をスケールできるようになっています。

求められるスケーラビリティやパフォーマンスはそれぞれのお客様によって異なり、あらゆる状況に最適な推奨事項というわけではありませんが、Tableau Server 10 では、8 ～ 16 コアで 25 ～ 100 人のユーザーを擁するチームや部署に対応できます。100 ～ 1,000 人規模のチームをサポートする場合は、使用状況やデータ更新のニーズにもよりますが、16 ～ 24 コアから始めてみるのが良いでしょう。ユーザー数がそれより多い場合やバックグラウンドの負荷が高い場合は、32 ～ 64 コアに増やして、ワーカーノードを追加することで増加するワークロードに対応でき、クラウド規模にまで拡張することも可能です。

Tableau について

Tableau は、インパクトを生み出すアクションにつながるインサイトを、お客様がデータから引き出せるように支援しています。どこにあるどのような形式のデータにでも、簡単にアクセスできます。隠れたビジネスチャンスを見つけ出すアドホック分析もすぐに行えます。ドラッグ & ドロップ操作で、高度なビジュアル分析を行えるインタラクティブなダッシュボードを作成できます。そして組織全体で共有すれば、チームメンバーが自分の視点からデータ を分析できるようになります。グローバルな大企業から、中小企業やスタートアップまで、あらゆる場所で多くのお客様が Tableau の分析プラットフォームを使い、データを見て理解しています。

リソース

企業向け Tableau: IT の概要

サーバー管理者ガイド

Tableau Server 10.0 の高可用性: 規模に応じたミッションクリティカルな分析を実行する

Amazon Web Services における Tableau