Microsoft

Web アプリケーションの
パフォーマンス テスト ガイダンス

patterns & practices

フィードバック/コメント送付先 : PerfTest@microsoft.com

Web アプリケーションの
パフォーマンス テスト ガイダンス

patterns & practices

J.D. Meier
Carlos Farre
Prashant Bansode
Scott Barber
Dennis Rea

本書は『patterns & practices Performance Testing Guidance for Web Applications』の日本語訳です。
原書については以下の URL を参考にしてください。
http://www.codeplex.com/PerfTestingGuide

著作権

序文 (Alberto Savoia)

Web アプリケーションのパフォーマンスをテストすることは簡単です。しかし、シナリオのデザインは非現実的になりがちです。また、無関係なパフォーマンス データを収集および計測してしまうこともよくあります。正しいシナリオのデザインや、適切なデータの収集を管理していても、結果の集計や表示に誤った統計手法を用いてしまう場合もあります。

私は、90 年代後半から、インターネット バブルがピークを迎え、落ち着きを見せる今日まで、Web アプリケーションのパフォーマンス テストに多くの時間を費やしてきました。この間、著名なインターネット企業向けに、いくつかミッション クリティカルな Web パフォーマンス テストや負荷テストをデザインし、導入しました。こうした各企業社内のパフォーマンスの "専門家" との共同作業は非常に意義深いものであったと同時に驚くべき体験でもありました。Web アプリケーションのパフォーマンスに関する作業に従事する大半のエンジニアは、優秀で、労を惜しまず、献身的でした。彼らは、高価なソフトウェアやハードウェアに投資し、適切な書籍を読み、その時点の "ベスト プラクティス" に従っていました。それでもどういうわけか、パフォーマンスの測定や予測の結果は現実に即していませんでした。パフォーマンス テストでは、Web アプリケーションのパフォーマンスやスケーラビリティを "過大" 評価することがあります。こうした過大評価は、Web アプリケーションを展開したときに、問題を紛糾させ、犠牲の大きなクラッシュにつながります。また、処理能力やスケーラビリティを "過小" 評価することもあります。こうした過小評価は、ハードウェアやインフラストラクチャに対する不必要な出費につながります。このようなテストのエラーは小さなものではありません。テストによっては、実際のパフォーマンスや処理能力を 1 桁以上過大にまたは過小に評価するものもあります。その結果、どのようなことが起きるでしょうか。

私の経験によれば、Web アプリケーションのパフォーマンス テストでのすべてのエラーの大多数は、テストを単純にし過ぎた結果によるものです。より正確に言うと、ユーザーの操作を単純化し過ぎた結果であり、テスト結果の集計やレポートを単純化し過ぎた結果です。輸送関係のエンジニアが指定されたハイウェイの交通パターンを予測するときに、大部分のドライバが同じ平均速度で運転し、同じ反応時間かつ同じ割合でブレーキやアクセルを踏み、決して車線は変更しないと想定している場合を想像してみてください。これは、単純ですが、まったく無意味なシナリオです。同様に、同じエンジニアが、平均速度は時速 57 マイルで、ラッシュアワーの平均速度は時速 25 マイルを上回らないため、交通渋滞は起きないとレポートしている場面を想像してください。これは単純ですが、大きな誤解を招く可能性のある結果です。残念ながら、Web アプリケーションのパフォーマンスをテストする大半のテスタは、過度の単純化により、この想

像上の輸送エンジニアよりも大きな過ちを犯します。

私は単純化には大賛成です。しかし、アルバート アインシュタインの次の言葉を思い出してください。「何事もでき得る限り単純化しなければならないが、必要以上に単純化してはならない。」Web アプリケーションのパフォーマンスをテストすることに関して言えば、注目すべき (待望久しい) 本書で読者に伝えたい核心が、この言葉です。執筆者たちは、彼らの情熱、経験、および苦労して得た知識を駆使し、読者が Web パフォーマンスのテストに正しい方法で取り組むために必要な、広範かつ詳細で拡張性のある基盤を提供しています。『Web アプリケーションのパフォーマンス テスト ガイダンス』では、不必要に細かい部分までは踏み込まず、テストの設計、実施、分析を行う際に必要な主要パラメータと変数を把握できる (見落とさない) ようにしています。

Web パフォーマンス テストの初心者であれば、本書が多くの時間や労力を節約できる、正しいアプローチへの出発点となるでしょう。Web パフォーマンス テストの経験が豊富なベテランの方でも、新たな見識が得られ、いくつか共通の見慣れたミスに関する項目をお読みいただけば、得心していただける点が多々あると確信しています。いずれにせよ、『Web アプリケーションのパフォーマンス テスト ガイダンス』はすべての Web パフォーマンス エンジニアにとって必携の書です。

Alberto Savoia
Agitar Software Inc. 社の創設者兼 CTO
2007 年 7 月

著書 : 『The Science and Art of Web Site Load Testing』、『Web Load Test Planning』、『Trade Secrets from a Web Testing Expert』

序文 (Rico Mariani)

パフォーマンスのチューニングよりも難解と考えられるものを想像することは困難です (おそらく、パフォーマンスのテストを除けば)。

マイクロソフト社内の各部門を次々と訪れ、パフォーマンスのテストについて調べただけでも、品質や成功の度合いが多種多様な、さまざまなアプローチが数多く見つかります。どの部門の担当者も自身のアプローチが間違いなく最善のアプローチだと断言するでしょう (いくぶん謙虚に語る数少ない正直者を除けば)。中には、実際その分野を非常によく研究しているため、自信を持つもっともな理由がある場合もあります。少なくとも私個人の経験では、マイクロソフト社内と社外でまったく状況が異なるということはありません。良い状況もあれば、悪い状況もあります。

この分野で見受けられる最も一般的な問題を一言で説明しなければならないとすると、それは "不均衡" です。テストする側面が多数存在するため、チームは 1 つ、2 つの側面に重点を置きがちになり、見落とした側面によって問題が生じることがあります。利用者数に注目すると、スループットだけを考えがちです。スムーズな配信に注目すると、待機時間だけを考えがちです。スケーラビリティに注目すると、コストだけを考えがちです。

設計時に主な要因を検討し、その後これらの要因を注意深く追跡し、各要因の均衡を保つことにより、優れたパフォーマンスが得られます。『Web アプリケーションのパフォーマンス テスト ガイダンス』のような書籍が読者に提供する最も優れたサービスは、おそらく、こうした要因をすべて広範に把握できるようにすることです。その結果、読者がテスト計画を立案するときに、考慮すべき選択肢のメニューが用意されることになります。さいわい、本書でこのような要因を把握できます。

このガイダンスには、最も重要な考慮事項が網羅されています。このような考慮事項には、目的のエンド ユーザー エクスペリエンスを把握および定量化する方法、調査対象の主要リソースを選択する方法、統計的に有意義な方法で結果を集計するヒント、このような実践方法をさまざまなソフトウェア ライフサイクルに適合させる方法などがあります。Web アプリケーションに正面から取り組むことに重点が置かれていますが、実際に指示されていることははるかに一般的で、多種多様なアプリケーションに容易に当てはめることができます。

優れたエンジニアリングは、予測可能なコストで予測可能な結果を生み出すことから得られます。実際、私が好んで口にすることに「測定を行っていなければ、エンジニアリングを行っていることにはならない。」というのがあります。本書は、優れたエンジニアリングを行うために必要な継続性のある評価指標を読者に提示し、パフォーマンス テストの基礎を提供します。

Rico Mariani

Visual Studio のチーフ アーキテクト
Microsoft Corporation
2007 年 7 月

Rico Mariani は 1988 年にマイクロソフトに入社しました。Microsoft® C バージョン 6.0 から言語製品に携わり、Microsoft Visual C++® バージョン 5.0 開発システムをリリースするまでこの部門に貢献しました。1995 年に、いずれ "Sidewalk" プロジェクトとなるプロジェクトの開発マネージャとなり、7 年にわたりさまざまな MSN テクノロジでのプラットフォーム作業を経験しました。2002 年夏、CLR チームのパフォーマンス アーキテクトとして開発部門に復帰しました。彼のパフォーマンスに関する取り組みが認められ、現在は Visual Studio のチーフアーキテクトを務めています。彼の関心事項には、コンパイラと言語理論、データベース、3-D アート、優れた小説などがあります。

はじめに

『Web アプリケーションのパフォーマンス テスト ガイド』では、パフォーマンス テストを実装するためのエンド ツー エンドのアプローチを提供します。パフォーマンス テストの初心者でも、パフォーマンス テストの現状のアプローチの改善方法を模索している方でも、抱えているシナリオを調整できる洞察力を得られるでしょう。

本ガイドの情報は、ユーザーのシナリオに適用される使用法を基にし、パフォーマンス テストの複数のプロフェッショナルから学んだ教訓を反映しています。ガイダンスはタスクベースで、以下の各部から構成されます。

- **第 I 部「パフォーマンス テストの概要」:** パフォーマンス テストの一般的な種類の概要、主な概念、およびパフォーマンス テストで使用される一般的な用語について説明します。
- **第 II 部「パフォーマンス テストの模範アプローチ」:** パフォーマンス テストの 7 つの主要アクティビティを示します。ここには、アジャイル ソフトウェア開発や CMMI® ソフトウェア開発など、さまざまな環境にパフォーマンス テストを適合させる方法を示すためにデザインされた情報もあります。
- **第 III 部「テスト環境の特定」:** パフォーマンス テストが必要なプロジェクトに関する情報を収集する方法について説明します。これには、システム アーキテクチャ、物理展開、ユーザー アクティビティ、関連バッチ プロセスに関する情報の収集もあります。
- **第 IV 部「パフォーマンス受け入れ基準の特定」:** パフォーマンス テストの対象を決定する方法について説明します。パフォーマンスのさまざまな目標や要件を、パフォーマンス テストの観点から明確にする方法についても説明します。
- **第 V 部「テストの計画と設計」:** 効果の高いパフォーマンス テストを設計するために、ワークロードとユーザー エクスペリエンスをモデル化する方法を示します。
- **第 VI 部「テストの実行」:** 実際のパフォーマンス テストの主要アクティビティを調べます。
- **第 VII 部「結果の分析とレポート」:** レポートの対象読者と目的を基に、有効な方法で結果を整理して提示する方法について説明します。
- **第 VIII 部「パフォーマンス テストの技法」:** 負荷テストとストレス テストの実行に関する主要技法を示します。

本ガイドの目的

本ガイドは、Web アプリケーションのパフォーマンス テストに重点を置き、以下に関する推奨事項を提供します。

- 動的 (アジャイル) 環境と構造化 (CMMI) 環境の両方でのパフォーマンス テストの管理と実施。
- 負荷テスト、ストレス テスト、その他の種類のパフォーマンス関連のテストなど、パフォーマンスのテスト。
- 目標の特定、テストの設計、テストの実行、結果の分析、レポートなど、パフォーマンス テストの主要アクティビティ。

本ガイドで扱っているトピックの多くは、他の種類のアプリケーションにも同様に当てはまりますが、一貫性を保ち、想定している読者の大多数にわかりやすい方法で概念を示すために、すべて Web アプリケーションの観点から説明しています。

本ガイドは、ツールに依存しないようにしています。つまり、本ガイドで提示する概念を実現するために特定のツールが必要になることはありません。ただし、一部の技法や概念にはある程度のツールの使用が必要になるものもあります。

本ガイドでは、パフォーマンスのチューニングを直接扱っていません。パフォーマンスのチューニングは、アプリケーションやテクノロジに大きく依存するため、本ガイドのスタイルや形式にはうまく当てはまりません。ただし、パフォーマンス テストのアクティビティとパフォーマンス チューニングのアクティビティが相互に重なり合ったり、影響を与える場合は、その対処方法を大まかに扱います。

本ガイドの執筆理由

以下のことを実現するために本ガイドを執筆しました。

- 実社会から学んだパフォーマンス テストに関する教訓を集約する
- エンド ツー エンドのパフォーマンス テストについてのロードマップを提示する
- 技術の状態と実践の状態の間にあるギャップを狭める

本ガイドの内容

- **パフォーマンス テストのアプローチ：** パフォーマンス テストを複数の論理単位に編成するアプローチを示します。こうしたアプローチは、アプリケーションのライフ サイクル全体にパフォーマンス テストを段階的に導入する場合に役立ちます。
- **原則と実践：** 本ガイドの基盤となり、推奨事項の安定した基礎を提供します。これらは、現場で使用して成功したアプローチも反映しています。
- **プロセスと手法：** パフォーマンス テストを管理および実施する場合の各手順を示します。

各手順は、単純化と実現可能な結果を目的とし、入力、出力、および手順を備えた複数のアクティビティに分割されます。このような手順をベースラインとして使用したり、独自のプロセスの進化に役立てたりすることができます。

- **ライフ サイクル アプローチ：** リスクを軽減し、総保有コスト (TCO) を低減するために、アプリケーションのライフ サイクル全般に及ぶパフォーマンス テストの管理に関するエンド ツー エンドのガイダンスを提供します。
- **モジュール化：** ガイドの各章は、それぞれ個別に読めるようにデザインされています。最初から最後まで読まなければメリットが得られないというわけではありません。必要な部分を利用してください。
- **総合的：** 本ガイドは、結末を念頭においてデザインされています。本ガイドを最初から最後までお読みいただくと、各部がまとまった形で組み合わされて編成されていることがわかります。こうした総合的な編成は、各部を寄せ集めた編成よりも優れています。
- **専門技術知識：** 本ガイドは、マイクロソフト全社のさまざまな分野の専門家と現場のユーザーからの見識を公開しています。

対象読者

本ガイドは、効果的なパフォーマンス テストを実施する必要のある各担当者に、必要なリソース、パターン、および実践方法を提示することを目的にしています。

本ガイドの使い方

最初から最後までお読みいただくことも、関連する各部や各章のみをお読みいただくこともできます。また、組織全体で採用することも、最優先のニーズへの対処に重要なコンポーネントを使用することもできます。

本ガイドの使い道

この包括的なガイダンスの使い道はたくさんあります。次に、こうした使い道をいくつか示します。

- **教科書として使用する：** パフォーマンス テストの実施方法を学習するための教科書として本ガイドを使用します。本ガイドには、各分野の専門家から得た教訓や体験が盛り込まれています。
- **参考書として使用する：** パフォーマンス テストですべきこととやってはいけないことを学習するための参考書として本ガイドを使用します。
- **アプリケーション開発ライフ サイクルにパフォーマンス テストを組み込む：** 目的に合

ったアプローチと実践方法を導入し、アプリケーションのライフ サイクルに組み込みます。

- **パフォーマンス テストの設計時に使用する：** 本ガイドで提示する原則やベスト プラクティスを使用して、アプリケーションを設計します。学習した教訓からメリットを得ます。
- **トレーニングを作成する：** 本ガイド全体で使用している概念や技法を基に、トレーニングを作成します。

本ガイドの編成

最初から最後までお読みいただくことも、仕事に必要な章のみをお読みいただくこともできます。

部

本ガイドは 8 部構成です。

- 第 I 部 パフォーマンス テストの概要
- 第 II 部 パフォーマンス テストの模範アプローチ
- 第 III 部 テスト環境の特定
- 第 IV 部 パフォーマンス受け入れ基準の特定
- 第 V 部 テストの計画と設計
- 第 VI 部 テストの実行
- 第 VII 部 結果の分析とレポート
- 第 VIII 部 パフォーマンス テストの技法

第 I 部 パフォーマンス テストの概要

- 第 1 章 Web アプリケーション パフォーマンス テストの基礎
- 第 2 章 パフォーマンス テストの種類
- 第 3 章 パフォーマンス テストで対処するリスク

第 II 部 パフォーマンス テストの模範アプローチ

- 第 4 章 Web アプリケーション パフォーマンス テストの主要アクティビティ
- 第 5 章 パフォーマンス テストと反復処理の調整
- 第 6 章 アジャイル パフォーマンス テスト サイクルの管理
- 第 7 章 規制された (CMMI) 環境でのパフォーマンス テスト サイクルの管理

第 III 部 テスト環境の特定

- 第 8 章 パフォーマンス テストの効果を高めるためのシステムの評価

第 IV 部 パフォーマンス受け入れ基準の特定

- 第 9 章 パフォーマンス テストの対象の決定
- 第 10 章 エンド ユーザー応答時間の目標の定量化
- 第 11 章 さまざまな種類のパフォーマンス受け入れ基準の確立

第 V 部 テストの計画と設計

- 第 12 章 アプリケーションの使用法のモデル化
- 第 13 章 各のユーザーのデータとばらつきの判断

第 VI 部 テストの実行

- 第 14 章 テストの実行

第 VII 部 結果の分析とレポート

- 第 15 章 パフォーマンス テストの主要な数学原理
- 第 16 章 パフォーマンス テスト レポートの基礎

第 VIII 部 パフォーマンス テストの技法

- 第 17 章 Web アプリケーションの負荷テスト
- 第 18 章 Web アプリケーションのストレス テスト

本ガイドで使用するアプローチ

テストの各アクティビティの主なタスクは、情報を収集して、関係者がテスト対象のアプリケーションの全体的な品質に関連する決定を行う際に、多くの情報を利用できるようにすることです。さらに、パフォーマンス テストでは、システム内のボトルネックの特定、システムのチューニング、将来のテスト用のベースラインの確立、パフォーマンスの目標や要件に対する準拠度合いの判断に役立つことに重点を置くようにします。また、アプリケーションを運用に移行する際、そのアプリケーションのサポートに必要なハードウェア構成の見積もりに、パフォーマンス テストの結果や分析結果が役立つようにします。

本ガイドで使用するパフォーマンス テストのアプローチは、次の複数のアクティビティから構成されます。

- **アクティビティ 1. テスト環境の特定 :** テスト チームが利用できるツールやリソースだけでなく、物理テスト環境と運用環境を特定します。物理環境には、ハードウェア構成、ソフトウェア構成、ネットワーク構成などがあります。最初にテスト環境全体を徹底的に把握することで、テストの設計や計画の効果を高め、プロジェクトの早期段階でテストの課題を見極めることができます。状況によっては、この処理をプロジェクトのライフ サイクル全体で定期的に見直す必要があります。
- **アクティビティ 2. パフォーマンス受け入れ基準の特定 :** 応答時間、スループット、およびリソース使用率の目標と制約を特定します。一般に、応答時間にはユーザーが、スループットにはビジネスが、リソースの使用率にはシステムが関与します。また、プロジェクトの目標や制約にとらわれずに、プロジェクトを成功と見なす基準を特定します。たとえば、パフォーマンス テストを使用して、構成設定のどの組み合わせから結果として最も望ましいパフォーマンス特性が得られるかを評価します。

- **アクティビティ 3. テストの計画と設計 :** 主要なシナリオを特定し、代表的なユーザー間のばらつきと、そのばらつきのシミュレーションを行う方法を決定します。さらに、テスト データを定義し、収集する評価指標を確立します。実装、実行、および分析の対象となるシステム使用法のモデルに、こうした情報を編成します。このようなモデルは複数になることもあります。
- **アクティビティ 4. テスト環境の構成 :** 機能やコンポーネントがテストで利用できるようになるまでに、各テスト方針の実行に必要なテスト環境、ツール、およびリソースを準備します。必要に応じてリソースを監視するための装備がテスト環境に備わっていることを確認します。
- **アクティビティ 5. テスト設計の実装 :** テストの設計に従って、パフォーマンス テストを開発します。
- **アクティビティ 6. テストの実行 :** テストを実行し、監視します。テスト、テスト データ、および収集した結果を検証します。テストやテスト環境を監視しながら、分析のために検証済みのテストを実行します。
- **アクティビティ 7. 結果の分析、レポート、および再テスト :** 結果データを集約し、共有します。各個人と複数の機能チーム間での協力により、データを分析します。残りのテストの優先順位を付け直し、必要に応じて再実行します。すべての測定値が受け入れ基準内に収まり、設定したしきい値を超えるものがなく、目的の情報がすべて収集された時点で、特定の構成での特定のシナリオのテストが完了します。

本ガイドへのフィードバック

本ガイドと付属コンテンツの正確性を保つため、あらゆる努力を行ってきました。本ガイドへのご意見、ご感想は、英語で PerfTest@microsoft.com まで電子メールでお送りください。

特に以下の内容に関するフィードバックをお待ちしています。

- 推奨事項固有の技術的な問題点
- 実用性や有用性に関する問題点

本ガイドの執筆者

本ガイドは、以下のチーム メンバが作成しました。

- J.D. Meier
- Carlos Farre
- Prashant Bansode
- Scott Barber
- Dennis Rea

寄稿者と校閲者

Alan Ridlehoover、Clint Huffman、Edmund Wong、Ken Perilman、Larry Brader、Mark Tomlinson、Paul Williams、Pete Coupland、Rico Mariani

社外の寄稿者と校閲者

Alberto Savoia、Ben Simo、Cem Kaner、Chris Loosley、Corey Goldberg、Dawn Haynes、Derek Mead、Karen N. Johnson、Mike Bonar、Pradeep Soundararajan、Richard Leeke、Roland Stens、Ross Collard、Steven Woody

成功事例のご報告のお願い

本ガイドが皆様のお役に立ったかどうかをお知らせください。直面した問題点を簡単にまとめ、本ガイドがどのように役に立ったかを教えてください。ご報告は英語で MyStory@Microsoft.com まで電子メールでお送りください。

第 I 部

パフォーマンス テストの概要

内容 :

- **Web アプリケーション パフォーマンス テストの基礎**
- **パフォーマンス テストの種類**
- **パフォーマンス テストで対処するリスク**

第 1 章 Web アプリケーション パフォーマンス テストの基礎

目的

- パフォーマンス テストについて学習する。
- パフォーマンス テストの主要アクティビティについて学習する。
- パフォーマンス テストが重要である理由について学習する。
- プロジェクトのコンテキストとパフォーマンス テストの関連性について学習する。
- パフォーマンスのチューニングをパフォーマンス テスト サイクルに適合させる方法について学習する。

概要

パフォーマンス テストとは、特定のワークロードの下で、システムの応答性、スループット、信頼性、スケーラビリティを判断するテストです。
通常、パフォーマンス テストは、以下の項目を達成するために実施されます。

- 運用への準備が整っているかどうかを査定する
- パフォーマンス基準に照らして評価する
- 複数のシステムのパフォーマンス特性やシステム構成を比較する
- パフォーマンスの問題点の発生源を見つける
- システムのチューニングをサポートする
- スループットのレベルを見つける

ここでは、パフォーマンス テストの原則を理解し、パフォーマンス テスト プロジェクトを成功に導く基礎となる、一連の基本的な構成要素を示します。また、本ガイド全体で使用しているさまざまな用語や概念についても説明します。

本章の使い方

ここでは、パフォーマンス テストの目的と主なアクティビティについて理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「プロジェクトのコンテキスト」では、パフォーマンス テスト中に関連する項目に注目する方法を理解します。
- 「パフォーマンス テストとチューニングの関係」では、パフォーマンス テストとパフォー

マンス チューニングの関係、およびパフォーマンス チューニングの全体的なプロセスについて理解します。

- 「パフォーマンス テスト、負荷テスト、およびストレス テスト」では、さまざまな種類のパフォーマンス テストについて理解します。
- 「ベースライン」と「ベンチマーク」では、アプリケーションの評価に使用できる、パフォーマンスのさまざまな比較方法について理解します。
- 「用語」では、プロジェクトのコンテキスト内で用語を正しく、明確に使用できるように、パフォーマンス テストに関する一般的な用語について理解します。

パフォーマンス テストの主要アクティビティ

パフォーマンス テストでは、通常、システム内のボトルネックの特定、将来のテスト用のベースラインの確立、パフォーマンス チューニング作業のサポート、パフォーマンスの目標や要件に対する準拠度合いの判断に役立つテストを行い、関係者がテスト対象のアプリケーションの全体的な品質に関連する決定を行う際、多くの情報を利用できるように、その他のパフォーマンス関連データを収集します。また、アプリケーションを運用に移行する際、そのアプリケーションのサポートに必要なハードウェア構成の見積もりに、パフォーマンス テストの結果や分析結果が役立つようにします。

図 1.1 パフォーマンス テストの主要アクティビティ

本ガイドで使用するパフォーマンス テストのアプローチは、次の複数のアクティビティから構成されます。

1. **アクティビティ 1. テスト環境の特定 :** テスト チームが利用できるツールやリソースだけでなく、物理テスト環境と運用環境を特定します。物理環境には、ハードウェア構成、ソフトウェア構成、ネットワーク構成などがあります。最初にテスト環境全体を徹底的に把握することで、テストの設計や計画の効果を高め、プロジェクトの早期段階でテストの課題を見極めることができます。状況によっては、この処理をプロジェクトのライフ サイクル全体で定期的に見直す必要があります。
2. **アクティビティ 2. パフォーマンス受け入れ基準の特定 :** 応答時間、スループット、およびリソース使用率の目標と制約を特定します。一般に、応答時間にはユーザーが、スループットにはビジネスが、リソースの使用率にはシステムが関与します。また、プロジェクトの目標や制約にとらわれずに、プロジェクトを成功と見なす基準を特定します。たとえば、パフォーマンス テストを使用して、構成設定のどの組み合わせから結果として最も望ましいパフォーマンス特性が得られるかを評価します。
3. **アクティビティ 3. テストの計画と設計 :** 主要なシナリオを特定し、代表的なユーザー間のばらつきと、そのばらつきのシミュレーションを行う方法を決定します。さらに、テスト データを定義し、収集する評価指標を確立します。実装、実行、および分析の対象となるシステム使用法のモデルに、こうした情報を編成します。このようなモデルは複数になることもあります。
4. **アクティビティ 4. テスト環境の構成 :** 機能やコンポーネントがテストで利用できるようになるまでに、各テスト方針の実行に必要なテスト環境、ツール、およびリソースを準備します。必要に応じてリソースを監視するための装備がテスト環境に備わっていることを確認します。
5. **アクティビティ 5. テスト設計の実装 :** テストの設計に従って、パフォーマンス テストを開発します。
6. **アクティビティ 6. テストの実行 :** テストを実行し、監視します。テスト、テスト データ、および収集した結果を検証します。テストやテスト環境を監視しながら、分析のために検証済みのテストを実行します。
7. **アクティビティ 7. 結果の分析、レポート、および再テスト :** 結果データを集約し、共有します。各個人と複数の機能チーム間での協力により、データを分析します。残りのテストの優先順位を付け直し、必要に応じて再実行します。すべての測定値が受け入れ基準内に収まり、設定したしきい値を超えるものがなく、目的の情報がすべて収集された時点で、特定の構成での特定のシナリオのテストが完了します。

パフォーマンス テストを行う理由

おおまかに言えば、ほとんどの場合、経費、機会費用、継続性、企業の評判に関連する 1 つまたは複数のリスクに対処するためにパフォーマンス テストを実施します。次に、パフォーマンス テストを実施するより具体的な理由をいくつか示します。

- 以下により、リリースの準備が整っているかどうかを査定する。
    - アプリケーションの運用時のパフォーマンス特性を予測または推測できるようにし、こうした予測を基に、パフォーマンスの懸案事項に対処されているかどうかを評価できるようにする。このような予測は、アプリケーションをリリースする準備が整っているかどうか、将来の拡張に対処できるかどうか、リリース前にパフォーマンスの改善やハードウェアのアップグレードが必要かどうかを関係者が判断する際にも貴重な資料になります。
    - ユーザーがシステムのパフォーマンス特性に対して不満を抱く可能性を示唆するデータを提供する。
    - スケーラビリティや安定性の問題、アプリケーションの応答時間に対するユーザーの不満などにより、収入の減少やブランド信用力の低下を予測する際に役立つデータを提供する。
- 以下により、インフラストラクチャの妥当性を査定する。
    - 現在の処理能力の妥当性を評価する。
    - 安定度の妥当性を判断する。
    - アプリケーションのインフラストラクチャの処理能力を判断し、受け入れ可能なアプリケーション パフォーマンスを提供するために必要な将来のリソースを決定する。
    - アプリケーションとビジネスの両方にとって最適に機能する構成を判断するために、さまざまなシステム構成を比較する。
    - 利用できるリソースに予算上の制約がある中で、アプリケーションが目的のパフォーマンス特性を示すことを確認する。
- 以下により、開発したソフトウェアのパフォーマンスの妥当性を査定する。
    - ソフトウェアに変更を加える前後に、目的のアプリケーション パフォーマンス特性を判断する。
    - アプリケーションのパフォーマンス特性について、現在値と目標値の比較を行う。
- 以下により、パフォーマンス チューニングの効果を高める。
    - さまざまな負荷レベルでアプリケーションの動作を分析する。
    - アプリケーションのボトルネックを特定する。
    - 製品を運用環境にリリースする前に、製品の速度、スケーラビリティ、および安

定性に関連する情報を提供し、多くの情報を利用して、システムをチューニングするかどうか、およびチューニングする場合はそのタイミングを決定できるようにする。

プロジェクトのコンテキスト

パフォーマンス テスト プロジェクトを成功に導くために、パフォーマンスをテストするアプローチとテスト自体の両方を、プロジェクトのコンテキストに関連付ける必要があります。プロジェクトのコンテキストを把握しておかないと、パフォーマンス テストは往々にして、時間の浪費、欲求不満、矛盾につながる本当に重要な項目ではなく、パフォーマンス テスタやテスト チームが重要だと考える項目のみに限定されます。

プロジェクトのコンテキストとは、プロジェクトを成功に導くことに関連する、または関連すると考えられるものにほかなりません。このようなコンテキストには以下のものがありますが、これに限定されるわけではありません。

- プロジェクトの全体的なビジョンまたは目的
- パフォーマンス テストの対象
- パフォーマンスの合格基準
- 開発のライフ サイクル
- プロジェクト スケジュール
- プロジェクトの予算
- 使用可能なツールと環境
- パフォーマンス テスタとテスト チームのスキル
- 検出したパフォーマンスの懸案事項の優先順位
- パフォーマンスが低いアプリケーションを展開した場合にビジネスに及ぼす影響

次に、プロジェクトのコンテキストの中でパフォーマンスのテスト作業に関連すると考えられる例をいくつか示します。

- **プロジェクトのビジョン：** パフォーマンス テストを開始する前に、プロジェクトの現在のビジョンを確実に理解します。プロジェクトのビジョンは、どのようなパフォーマンス テストが必要かつ重要かを判断する際の基盤になります。ビジョンは変化する可能性があるため、定期的に見直します。
- **システムの目的：** テスト対象のアプリケーションまたはシステムの目的を把握します。その結果、テストで重点を置くべき、最も優先順位の高いパフォーマンス特性を見極めるこ

とができます。システムの目的のほか、実際に展開するハードウェアとソフトウェアのアーキテクチャ、および代表的なエンド ユーザーの特性を把握する必要があります。

- **顧客やユーザーの期待値 :** パフォーマンス テストを計画するときは、顧客またはユーザーの期待値に留意します。顧客やユーザーの満足度は、単に明示した要件への準拠度合いではなく、期待値に左右されることを覚えておいてください。
- **ビジネスの原動力 :** ビジネス ニーズやビジネス チャンスなど、予算、スケジュール、リソースにある程度制約されるビジネスの原動力を把握します。利用可能な予算の範囲内でタイムリーにビジネス要件を満たすことが重要です。
- **パフォーマンスをテストする理由 :** プロジェクトのごく初期に、パフォーマンス テストを実施する理由を把握します。これを行わないと、非効率なパフォーマンス テストにつながることがあります。このような理由は、プロジェクトの進捗と共にパフォーマンスの受け入れ基準の一覧に収まらなくなり、優先順位が変化したり、付け替えられたりすることがよくあるため、テスタやテスト チームがアプリケーション、そのパフォーマンス、および顧客やユーザーについてさらに詳しく理解できるように、定期的に見直します。
- **パフォーマンス テストがプロジェクトにもたらす価値 :** プロジェクト レベルおよびビジネス レベルの対象を、具体的で、特定可能かつ管理可能なパフォーマンス テストのアクティビティに変換することで、パフォーマンス テストからプロジェクトにもたらされることを期待する価値を把握します。こうしたアクティビティを調整し、優先順位を付けることで、パフォーマンス テストのどのアクティビティが付加価値を提供する可能性があるかを判断します。
- **プロジェクトの管理と人員配置 :** パフォーマンス テストを効果的に実施するためのチームの編成、運用手法、およびコミュニケーション手法を把握します。
- **プロセス :** チームのプロセスを把握し、そのプロセスをどのようにパフォーマンス テストに当てはめるかを解明します。チームのプロセス ドキュメントでパフォーマンス テストを直接扱っていない場合は、ドキュメントを推敲して、できる限りパフォーマンス テストを含めるようにします。その後、プロジェクト マネージャやプロセス エンジニアに改訂したドキュメントの承認を得ます。
- **コンプライアンス基準 :** プロジェクトに関連する規制要件を把握します。テストに関連する特定の言語やコンテキストのステートメントがあることを確認するため、コンプライアンス ドキュメントを入手します。こうした情報は、コンプライアンス テストの特定や、準拠製品の確認に重要です。また、パフォーマンス テストの性質により、機能テストのために開発されたプロセスには事実上従うことができないことも理解してください。
- **プロジェクト スケジュール :** プロジェクトの開始日と終了日、ハードウェアと環境を利用

できる日付、ビルドとリリースの流れ、プロジェクト スケジュール内のチェックポイントとマイルストーンに注意します。

パフォーマンス テストとチューニングの関係

エンド ツー エンドのパフォーマンス テストで、受け入れ可能ではないと考えられるシステム特性やアプリケーション特性が明らかになると、多くのチームはアプリケーションのパフォーマンスを受け入れ可能にするために必要な点を見つけ出すために、パフォーマンス テストからパフォーマンス チューニングに注目を移します。また、パフォーマンス基準を満たしていても、プラットフォームに余裕を持たせ、必要なハードウェアの量を減らし、システムのパフォーマンスをさらに改善するために、使用するリソースの量を減らしたいと考え、重点をチューニングに移すこともあります。

共同作業

チューニングは大部分のパフォーマンス テスタにとって直接の担当ではありませんが、テスト対象のアプリケーションやシステムに関与する、以下のようなすべての関係者が共同作業を行うと、チューニング プロセスの効果が最も高くなります。

- 製品ベンダ
- アーキテクト
- 開発者
- テスタ
- データベース管理者
- システム管理者
- ネットワーク管理者

複数の機能チームの協力がなければ、パフォーマンスの問題を効果的または効率的に解決するために必要なシステム規模の観点を得ることはほぼ不可能です。

チューニングでは、通常、さまざまな負荷条件や構成の下でコンポーネント、リソース、および応答時間を新たに監視する必要が生じるため、パフォーマンス テストのテスタ (テスト チーム) はこのような共同作業チームの重要な一員です。一般的に言えば、こうした情報を効果的な方法で提供するツールや専門知識を備えているのがパフォーマンス テスタです。そのため、チューニングにはパフォーマンス テスタが欠かせません。

チューニング プロセスの概要

チューニングは反復処理です。この反復処理は、通常、プロジェクトが従うパフォーマンス テスト アプローチから分離されるもので、まったく独立したものではありません。以下で、代表的なチューニング プロセスの概要を簡単に説明します。

- テスト プロセスの開始時点の構成やテスト結果を把握、再現できるように、十分に定義され、管理されたテスト環境に展開されたシステムやアプリケーションを使ってテストを実施します。
- テストで受け入れられないと思えるパフォーマンス特性が明らかになった時点で、パフォーマンス テストとチューニングのチームは、テスト環境やアプリケーションへの変更が必要になる、診断と修正の段階 (チューニング) に入ります。診断を目的として問題点を際立たせるよう慎重に設計された一時的な変更を加えたり、変更がパフォーマンスの向上につながるかどうかを確認するためにテスト環境に変更を加えたりすることは、珍しいことではありません。
- 一般に、テストとチューニングの共同作業チームには、チューニング フェーズの効果を最大限に引き出すために、テスト環境に対する完全かつ独占的な制御権が与えられます。
- パフォーマンス テストを実行します。つまり、改善に向けた変更の影響を測定するため、テスト環境に変更を加えるたびにテストを再実行します。
- チューニング プロセスでは、通常、変更とテストを間断なく実行する必要があります。チューニング フェーズ中のこうした作業に、テストとチューニングの共同作業チームを完全に利用できなかったり、独占的に利用できない場合は、プロセスに多くの時間がかかることがあります。
- チューニング フェーズが完了すると、通常、テスト環境が初期状態にリセットされ、改善に向け成果のあった変更が再度適用され、成果のなかった変更は (一時的に実装した変更や診断のための変更と共に) 破棄されます。その後、正しく変更が認識されたことを実証するために、パフォーマンス テストを繰り返します。最低限必要な運用環境に応じた新しい期待値を反映するために、テスト環境自体を変更する場合もあります。このような変更はあまり一般的ではありませんが、チューニング作業の結果として生じる可能性があります。

パフォーマンス テスト、負荷テスト、およびストレス テスト

パフォーマンス テストは、通常、以下の 3 つのカテゴリのいずれかに属するものとして説明されます。

- **パフォーマンス テスト：** この種のテストでは、テスト対象のシステムやアプリケーションの速度、スケーラビリティ、および安定性の各特性が判断または検証されます。パフォーマンスでは、応答時間、スループット、リソースの使用率がプロジェクトや製品のパフォーマンス目標を満たすレベルを達成することが懸案事項になります。本ガイドでは、パフォーマンス テストは、パフォーマンス関連のテストの他のサブカテゴリすべてのスーパーセットを表します。
- **負荷テスト：** これはパフォーマンス テストのサブカテゴリで、運用中に予測されるワークロードや負荷の量を前提としたときの、テスト対象のシステムやアプリケーションのパフォーマンス特性を判断または検証することに重点を置きます。
- **ストレス テスト：** これはパフォーマンス テストのサブカテゴリで、運用中に予測される状態を超えたときの、テスト対象のシステムやアプリケーションのパフォーマンス特性を判断または検証することに重点を置きます。また、メモリ制限、ディスク容量不足、サーバー障害など、ストレスの高い状態を前提としたときの、テスト対象のシステムやアプリケーションのパフォーマンス特性を判断または検証することに重点を置くこともあります。このようなテストは、アプリケーションがどのような状態で、どのように失敗するか、および障害の発生を警告するためにどのようなインジケータを監視できるかを判断できるように設計します。

ベースライン

ベースラインの作成とは、その後のパフォーマンスを改善するためにシステムやアプリケーションに加える変更の効果を評価する目的で、パフォーマンスの測定データを把握する一連のテストを実行するプロセスです。ベースラインの重要な点は、比較のために具体的に変化させるものを除き、すべての特性や構成オプションを変化させてはいけないことです。ベースラインとの比較のために意図的に変更させなかった部分に変更を加えると、そのベースラインの測定値は比較の基礎としては有効ではなくなります。

Web アプリケーションの場合は、ベースラインを使用して、パフォーマンスが向上したか低下したかを判断し、さまざまなビルドやバージョン間の差異を見つけます。たとえば、読み込み時間、一定時間あたりに処理したトランザクション数、一定時間あたりにサービスを提供した Web ページ数、メモリ使用率やプロセッサ使用率などのリソース使用率を計測できます。ベースラインを使用する際には以下のような考慮事項があります。

- **システム、コンポーネント、またはアプリケーションに対するベースラインを作成できます。** データベース、Web サービスなど、アプリケーションのさまざまな層に対するベース

ラインを作成することもできます。

- **ベースラインでは、将来の最適化や処理能力の低下を追跡する際に比較対象となる標準を設定できます。**環境やワークロードの特性により、複数のテスト結果に大きなばらつきが生じる可能性があるため、ベースラインの結果に再現性があることを確認することが重要です。
- **ベースラインは、パフォーマンスの変化の特定に役立ちます。**開発ライフ サイクルの過程で行う規模縮小や最適化を反映してパフォーマンスが変化するのを製品チームが特定する際にベースラインが有効です。既知の状態や構成との比較でこうした変化を特定すると、多くの場合、パフォーマンスの問題の解決が容易になります。
- **ベースライン資産は再利用可能にする必要があります。**再利用可能な一連のテスト資産を使用して作成したベースラインが最も価値が高くなります。このようなテストでは、再現可能かつ操作可能なワークロード特性のシミュレーションを正確に行うことが重要です。
- **ベースラインは評価指標です。**ベースラインの結果は、応答時間、プロセッサの処理能力、メモリ使用率、ディスク容量、ネットワーク帯域幅などのパフォーマンスの主要インジケータを使用して明確に示すことができます。
- **ベースラインは、参照用に共有する枠組みとして機能します。**ベースラインの結果をチームで共有すると、アプリケーションやコンポーネントのパフォーマンス特性について獲得した知識を格納するための共通ストアを構築できます。
- **ベースラインは汎用化しすぎないようにします。**プロジェクトでアプリケーションのエンジニアリングを大きくやり直す必要が生じた場合、アプリケーションをテストするためのベースラインを再確立する必要があります。ベースラインをアプリケーション固有にしておくことが、さまざまなバージョン間でパフォーマンスを比較する際に最も有効です。場合によっては、現在以降のバージョンと以前のバージョンのアプリケーションが大きく異なると、ベースラインは比較には有効ではなくなります。
- **アプリケーションの動作を把握します。**ベースラインの作成時点にアプリケーションの動作を完全に把握しておくことをお勧めします。最適化を重点目標にしてシステムに変更を加える前に動作を把握しておかないと、変更が逆効果になることがよくあります。
- **ベースラインを進化させます。**ときには、ベースラインの初期作成時以降にシステムに変更が加えられたため、ベースラインを再定義する必要が生じることがあります。

ベンチマーク

ベンチマークとは、内部で作成したベースラインに対して、またはなんらかの第三者組織が裏付けしている業界標準に対して、システムのパフォーマンスを比較するプロセスです。

Web アプリケーションの場合、アプリケーションのベンチマーク スコアを判断するのに必要なパフォーマンス測定基準を捕捉するため、業界ベンチマークの仕様に準拠する一連のテストを実行することになります。その後、アプリケーションのスコアを、同じベンチマークで他のシステムやアプリケーションが獲得したスコアと比較できます。ベンチマークの一定のスコアを達成または上回るために、アプリケーションのパフォーマンスのチューニングを選択することもできます。ベンチマークには、以下のような考慮事項があります。

- **規則に従って作業する必要があります。**ベンチマークは、業界仕様を使って作業するか、このような標準を満たすように既存の実装を移植することによって行います。ベンチマークでは、同時に実行する必要のあるコンポーネント、製品を投入する市場、測定する具体的な評価指標などをすべて特定する必要があります。
- **規則に従って作業するため、結果を公開できます。**ベンチマークの結果は外部に公開できます。競合他社が比較を行う可能性があるので、結果の信頼性を確保するため、テストやデータに一連の厳密な標準アプローチを採用します。
- **さまざまな評価指標から結果を判断します。**パフォーマンスの評価指標には、読み込み時間、一定時間あたりに処理したトランザクション数、一定時間あたりにアクセスされた Web ページ数、プロセッサ使用率、メモリ使用率、検索時間などがあります。

用語

本ガイド全体で使用している定義を以下に示します。ここに示す用語と定義は、公式に使用されているものや業界標準と矛盾しないように努めました。ただし、ここに示した用語の中には、特定の業界や組織で別の定義や意味を持っているものが含まれていることも認識しています。これらの定義はコミュニケーションを助ける目的で示しており、汎用的な標準を作成する試みではないことに留意してください。

用語 / 概念	説明
処理能力 (キャパシティ)	システムの "処理能力" とは、事前に決定したパフォーマンスの主要受け入れ基準に違反しないで処理できるワークロードの合計です。
処理能力テスト	"処理能力テスト" は、負荷テストを補完するテストで、サーバーの最終的な障害ポイントを特定します。これに対して、負荷テストは、さまざまなレベルの負荷パターンやトラフィック パターンで結果を監視します。処理能力テストは、処理能力計画に関連して実施します。処理能力計画は、将来のユーザー ベースの拡大やデータ量の増加に備えた計画を

	立てるために実施します。たとえば、将来の負荷増加に対応するには、予想される使用レベルのサポートに必要な追加リソース (プロセッサの処理能力、メモリ使用量、ディスク容量、ネットワーク帯域幅など) を把握する必要があります。処理能力テストを使用して、規模の拡大方針を明確にし、スケール アップとスケール アウトのどちらを選択するかを決定できます。
コンポーネント テスト	"コンポーネント テスト" は、アプリケーション アーキテクチャのコンポーネントを対象とするパフォーマンス テストです。一般的にテストされるコンポーネントには、サーバー、データベース、ネットワーク、ファイアウォール、記憶装置などがあります。
耐久テスト	"耐久テスト" は、テスト対象の製品を長期間運用し、予想されるワークロード モデルと負荷の量を当てはめたときに、どのようなパフォーマンス特性を示すかを確認または検証することに重点を置いたパフォーマンス テストです。耐久テストは、負荷テストのサブセットです。
調査	"調査" は、テスト対象の製品の速度、スケーラビリティ、安定性の各特性に関連する情報を収集することを基本とするアクティビティで、製品の品質を判断または改善するのに役立ちます。調査は、多くの場合、観測された 1 つ以上のパフォーマンスの問題の根本原因に関する仮説が正しいかどうかを検証するために行われます。
待機時間	待機時間は、応答性の測定値で、要求の実行が完了するまでにかかった時間を表します。また、サブタスクの合計待機時間を表すこともあります。
評価指標 (メトリクス)	"評価指標" は、実行しているパフォーマンス テストで得られる測定値で、一般に理解されている尺度で表現されます。パフォーマンス テストで通常得られる評価指標には、時間ごとのプロセッサ使用率や負荷別のメモリ使用量などがあります。
パフォーマンス	"パフォーマンス" とは、アプリケーションの応答時間、スループット、リソース使用率のレベルに関する情報のことです。
パフォーマンス テ	"パフォーマンス テスト" は、テスト対象の製品の応答性、速度、スケー

スト	ラビリティ、安定性に関する特性を判断または検証するための技術調査です。パフォーマンス テストは、ここで説明したパフォーマンス テストの他のサブカテゴリをすべて含むスーパーセットです。
パフォーマンス予算 (割り当て)	"パフォーマンス予算" ("割り当て") は、開発者がコンポーネントに使用できるリソース量に関して課せられる制約です。
パフォーマンス目標	"パフォーマンス目標" とは、製品をリリースする前に満たす必要のある基準です。ただし、このような基準は、特定の環境下では調整可能な場合があります。たとえば、あるトランザクションの応答時間の目標が 3 秒に設定されていて、実際の応答時間が 3.3 秒だった場合、関係者はこのアプリケーションのリリースを選択し、このトランザクションのパフォーマンス チューニングを次回以降のリリースまで延期する可能性があります。
パフォーマンス対象	"パフォーマンス対象" は、通常、応答時間、スループット (1 秒あたりのトランザクション数)、およびリソース使用率のレベルに関して指定されます。一般には、ユーザーの満足度に直接つながる可能性のある評価指標に重点が置かれます。
パフォーマンス要件	"パフォーマンス要件" は、契約上の義務、サービス レベル アグリーメント (SLA)、ビジネスの固定ニーズなどにより、絶対に調整できない基準です。基準に合格するまでリリースを遅らせる決定に明白にはつながらないパフォーマンス基準は、必ずしも必要ではありません。したがって、要件とはなりません。
パフォーマンス ターゲット	"パフォーマンス ターゲット" は、特定の条件セットの下でプロジェクトに特定された評価指標の目標値で、通常、応答時間、スループット、リソース使用率のレベルに関連して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク I/O、ネットワーク I/O などがあります。通常、パフォーマンス ターゲットはプロジェクトの目標と一致します。
パフォーマンス テストの対象	"パフォーマンス テストの対象" とは、パフォーマンス テスト プロセスから収集されるデータのことです。これらのデータは、製品品質を判断

	または改善する際に必要な値を持つと予測されます。ただし、こうした対象は定量的である必要はなく、パフォーマンスの要件、目標、または規定したサービスの品質 (QoS) 仕様に直接関連する必要もありません。
パフォーマンスのしきい値	"パフォーマンスのしきい値" は、プロジェクトで特定された評価指標の許容最大値で、通常、応答時間、スループット (1 秒あたりのトランザクション数)、およびリソース使用率のレベルに関して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク I/O、ネットワーク I/O などがあります。通常、パフォーマンスのしきい値は要件と一致します。
リソース使用率	"リソースの使用率" とは、システム リソースに関するプロジェクトのコストです。主なリソースには、プロセッサ、メモリ、ディスク I/O、およびネットワーク I/O があります。
応答時間	"応答時間" は、クライアント要求に対するアプリケーションやサブシステムの応答の速さを示す測定値です。
飽和状態	"飽和状態" とは、リソースがすべて使用状態になった時点のことです。
スケーラビリティ	"スケーラビリティ" とは、プロセッサ、メモリ、記憶容量などを追加することで生じる新たなワークロードを、パフォーマンスに悪影響を与えることなく処理できる、アプリケーションの能力のことです。
シナリオ	パフォーマンス テストのコンテキストでは、シナリオはアプリケーション内の一連の手順です。シナリオでは、製品カタログの検索、ショッピング カートへの商品の追加、発注など、ユース ケースやビジネス機能を表現することができます。
スモーク テスト	"スモーク テスト" とは、パフォーマンス テストの中で最初に実施するテストで、通常の負荷でアプリケーションが正常に動作するかどうかを確認します。
スパイク テスト	"スパイク テスト" は、テスト対象の製品を短期間運用し、予想される程度を上回るまで、ワークロード モデルのレベルと負荷の量を徐々に引き上げ、どのようなパフォーマンス特性を示すかを判断または検証するこ

	とに重点を置いたパフォーマンス テストです。スパイク テストは、ストレス テストのサブセットです。
安定性	パフォーマンス テストのコンテキストでの安定性とは、さまざまな条件下でのアプリケーションの全体的な信頼性、堅牢性、機能とデータの整合性、可用性、応答の一貫性のことです。
ストレス テスト	"ストレス テスト" とは、通常の負荷状態やピーク時の負荷状態を超えるように負荷を押し上げたときのアプリケーションの動作を評価するようにデザインされた、一種のパフォーマンス テストです。ストレス テストの目標は、高い負荷がかかったときにのみ表面化するアプリケーションのバグを明らかにすることです。考えられるバグとしては、同期に関する問題、競合状態、メモリ リークなどがあります。ストレス テストでは、アプリケーションの弱点や、非常に高い負荷がかかった状態でのアプリケーションの動作を特定できます。
スループット	スループットとは、単位時間あたりに処理できる作業の単位数です。たとえば、1 秒あたりの要求数、1 日あたりの呼び出し回数、1 秒あたりのアクセス回数、1 年間のレポート数などがあります。
単体テスト	パフォーマンス テストのコンテキストでの "単体テスト" とは、コードのモジュール、つまりアプリケーションの既存のコード ベース全体の中の論理サブセットを対象とした、パフォーマンス特性に重点を置いたテストのことです。一般にテストされるモジュールには、関数、プロシージャ、ルーチン、オブジェクト、メソッド、クラスなどがあります。多くの場合、パフォーマンスに関する単体テストは、テストするモジュールのコードを記述した開発者によって作成および実施されます。
使用率	パフォーマンス テストのコンテキストでの "使用率" とは、ユーザーにサービスを提供するためにリソースがビジー状態になっている時間の割合のことです。これ以外の時間はアイドル時間と見なされます。
検証テスト	"検証テスト" は、テスト対象の製品の速度、スケーラビリティ、安定性に関する特性を、製品に設定または推定された期待値と比較するテストです。

ワークロード	"ワークロード" とは、使用パターンのシミュレーションを行うために、システム、アプリケーション、またはコンポーネントに与えられる、同時実行やデータ入力に関連する負荷のことです。ワークロードには、ユーザーの総数、同時にアクティブになるユーザー数、データの量、トランザクション ミックスを伴うトランザクションの量などがあります。パフォーマンスをモデル化する場合に、ワークロードを個別のシナリオに関連付けます。

まとめ

パフォーマンス テストは、システム内のボトルネックの特定、将来のテスト用のベースラインの確立、パフォーマンス チューニング作業のサポート、パフォーマンスの目標や要件に対する準拠度合いの判断に役立ちます。開発ライフ サイクルのごく初期の段階にパフォーマンス テストを含めると、プロジェクトに大きな付加価値が提供されます。

パフォーマンス テスト プロジェクトを成功に導くには、テストをプロジェクトのコンテキストに関連付ける必要があります。その結果、本当に重要な項目のみに注目することができます。

パフォーマンスの特性を受け入れられない場合は、アプリケーションのパフォーマンスを受け入れ可能な状態にするために、パフォーマンスのテストからパフォーマンスのチューニングに重点を移します。システムのパフォーマンスをさらに改善するために、使用するリソースの量を減らしたいと考え、重点をチューニングに移すこともあります。

パフォーマンス テストのサブカテゴリには、パフォーマンス テスト、負荷テスト、およびストレス テストがあり、それぞれ異なる目的があります。

将来のパフォーマンスを改善するために、システムやアプリケーションに加える変更の効果を評価する際の基準となるベースラインを作成しておくと、通常、プロジェクトの効率は向上します。

第 2 章 パフォーマンス テストの種類

目的

- さまざまな種類のパフォーマンス テストについて学習する。
- 各種パフォーマンス テストによってもたらされる価値とメリットを把握する。
- 各種パフォーマンス テストによってもたらされる可能性があるデメリットを把握する。

概要

パフォーマンス テストとは、さまざまな種類のパフォーマンス関連のテストを総称する用語です。各種テストは特定の問題領域に対応しており、それぞれ独自のメリット、リスク、および課題があります。

ここでは、いくつか一般的な種類やカテゴリに属するパフォーマンス関連テストのメリットと、それらがプロジェクトに与えるリスクを定義し、各テストの内容を概説します。この章を一読すれば、既に確立されたチーム内でも使い方や解釈を誤りやすいパフォーマンス テストに関連する多くの用語について、正しい捉え方ができるようになります。

本章の使い方

ここでは、さまざまな種類のパフォーマンス関連テストについて理解します。各種テストを理解すれば、チームは現状のリスク、懸案事項、またはテスト結果を基に、どのパフォーマンス関連のテストが特定のプロジェクトに最も大きな価値をもたらす可能性が高いかを判断できるようになります。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「パフォーマンス テストの主な種類」では、特定の懸案事項との関連性が最も高いテストを判断し、各種テストのトレードオフを行って、十分な情報に基いて決定を下します。
- 「主な種類のパフォーマンス テストによってもたらされるメリットの概要」では、特定の種類のテストから得られるメリットだけでなく、そのテストでは適切に対処できない可能性がある課題や懸案事項を理解します。
- 「その他の概念と用語」では、プロジェクトにさらなる価値をもたらす可能性がある他のパフォーマンス テストについて理解します。その結果、共通の背景を持たないメンバともパフォーマンス テストに関する会話をスムーズに行うことができるようになります。

パフォーマンス テスト

"パフォーマンス テスト" とは、テスト対象の製品の速度、スケーラビリティ、安定性に関する特性を判断または検証するための技術調査と定義できます。テストやチューニングなどのパフォーマンス関連のアクティビティは、テスト対象のアプリケーションの応答時間、スループット、リソース使用率などのパフォーマンスを目標レベルまで向上させることを目的としています。パフォーマンス テストは、さまざまなサブセットをすべて含む総称的な用語であるため、ここで紹介する他の種類のパフォーマンス テストによってもたらされる価値やメリットは、パフォーマンス テスト全般のメリットと考えることもできます。

パフォーマンス テストの主な種類

次の表では、Web アプリケーションに最も一般的に使用されるパフォーマンス テストの種類について説明します。

用語	目的	メモ
パフォーマンス テスト	速度、スケーラビリティ、安定性を判断または検証する。	• パフォーマンス テストは、テスト対象の製品の応答性、速度、スケーラビリティ、安定性に関する特性を判断または検証するための技術調査です。
負荷テスト	通常時とピーク時の負荷状態でアプリケーションの動作を検証する。	• 負荷テストは、アプリケーションのパフォーマンスが目標を満たしているかどうかを検証するために実施されます。多くの場合、パフォーマンスの目標は、サービス レベル アグリーメント (SLA) で指定されます。負荷テストでは、応答時間、スループット、リソース使用率を計測し、アプリケーション限界点がピーク時の負荷状態以下で生じると想定して、限界点を特定できます。 • 耐久テストは、負荷テストのサブセ

		ットです。"耐久テスト" は、テスト対象の製品を長期間運用し、予想されるワークロード モデルと負荷の量を当てはめたときに、どのようなパフォーマンス特性を示すかを確認または検証することに重点を置いたパフォーマンス テストです。 • 耐久テストは、平均故障間隔 (MTBF) や平均故障時間 (MTTF) などの指標を算出するために使用される場合があります。
ストレス テスト	通常時またはピーク時の負荷状態を超える負荷がかかったときのアプリケーションの動作を判断または検証する。	• ストレス テストの目標は、高い負荷がかかったときにのみ表面化するアプリケーションのバグを明らかにすることです。考えられるバグとしては、同期に関する問題、競合状態、メモリ リークなどがあります。ストレス テストでは、アプリケーションの弱点や、非常に高い負荷がかかった状態でのアプリケーションの動作を特定できます。 • スパイク テストは、ストレス テストのサブセットです。"スパイク テスト" は、テスト対象の製品を短期間運用し、予想される程度を上回るまで、ワークロード モデルのレベルと負荷の量を徐々に引き上げ、どのようなパフォーマンス特性を示すかを判断または検証することに重点を置いたパフォーマンス テストです。
処理能力テスト	特定のシステムがサポートを提供しながらパフォ	• 処理能力テストは、処理能力計画に関連して実施します。処理能力計画

	ーマンスの目標を満たし続けることができるユーザー数やトランザクション数を判断する。	は、将来のユーザー ベースの拡大やデータ量の増加に備えた計画を立てるために実施します。たとえば、将来の負荷増加に対応するには、予想される使用レベルのサポートに必要な追加リソース (プロセッサの処理能力、メモリ使用量、ディスク容量、ネットワーク帯域幅など) を把握する必要があります。 • 処理能力テストを使用して、規模の拡大方針を明確にし、スケール アップとスケール アウトのどちらを選択するかを決定できます。

Web アプリケーションのパフォーマンスに関する最も一般的な懸案事項としては、十分な処理速度が得られるか、すべてのクライアントがサポートされるか、問題が発生した場合にどのような現象が見られるか、顧客の数が増加した場合に備えてどのような計画を立てればよいか、などがあります。普段の会話では、ほとんどの人がパフォーマンス テストから "十分な処理速度" を、負荷テストからは "現在のユーザー ベースや今後予想されるユーザー ベース" を、ストレステストからは "問題の発生" を、処理能力テストからは "今後の規模拡大に備えた計画" を連想します。まとめると、Web アプリケーションに使用される 4 種類の主要パフォーマンス テストの基盤は、このようなリスクから形成されると言えます。

主な種類のパフォーマンス テストによってもたらされるメリットの概要

用語	メリット	対処されない課題と領域
パフォーマンス テスト	• アプリケーションの速度、スケーラビリティ、安定性の各特性を判断し、ビジネスに関する適切な意思決定につなげることができる。 • システムのユーザーがア	• 負荷がかかったときにのみ表面化する機能上の欠陥が検出されないことがある。 • 設計と検証を注意深く行わないと、パフォーマンスの各特性を確認できる

	プリケーションのパフォーマンス特性に満足するかどうかを判断することに重点が置かれている。 • パフォーマンス関連の期待値と実際の結果との不一致を特定できる。 • チューニング、処理能力計画、最適化作業がサポートされる。	運用シナリオがごく少数に限られる可能性がある。 • 運用環境のハードウェア、つまりユーザーが使用するものと同じコンピュータでテストを行わないと、常に、結果の信頼性がある程度失われる。
負荷テスト	• 運用環境で予想されるピーク時の負荷をサポートするために必要なスループットを判断できる。 • ハードウェア環境の妥当性を判断できる。 • 負荷分散機能の妥当性を評価できる。 • 同時実行に関する問題点を検出できる。 • 負荷がかかったときに発生する機能の異常を検出できる。 • スケーラビリティと処理能力計画を目的としてデータを収集できる。 • アプリケーションがパフォーマンスに悪影響を与えずに処理できるユーザーの数を判断できる。 • ハードウェアがリソース使用率の制限値を上回る	• 応答速度を重視して設計されていない。 • テスト結果は、他の関連負荷テストの結果との比較にしか使用できない。

	ことなく処理できる負荷の量を判断できる。	
ストレス テスト	• システムに過度のストレスをかけたときにデータが破損する可能性があるかどうかを判断できる。 • アプリケーションの許容負荷をどの程度上回ったときに、障害、エラー、および処理速度の低下が発生するかを推定できる。 • 障害の危険性を警告する、アプリケーション監視トリガを確立できる。 • ストレスがかかった状態でセキュリティの脆弱性が生じるかどうかを確認できる。 • 一般的なハードウェア エラーやサポート アプリケーションのエラーによる二次的な影響を判断できる。 • 対策を立てる必要性が最も高い障害の種類を判断できる。	• ストレス テストでは意図的に非現実的な設計が行われるため、関係者がテスト結果を頭から否定する可能性がある。 • 多くの場合、どの程度のストレスをかければよいかを判断するのが困難。 • アプリケーションをテスト環境に分離しておかないと、アプリケーションやネットワークで障害が発生した場合に、業務が長時間にわたって中断する可能性がある。
処理能力テスト	• ビジネス要件を満たすために必要なワークロードの処理方法に関する情報を提供できる。 • 処理能力計画の担当者が	• 処理能力モデルの検証テストの作成は複雑な作業。 • 処理能力計画モデルのすべての点が大きな価値を

	モデルや予測値を検証または強化するために使用できる実データを提供できる。 • さまざまなテストを実施して、容量計画に関する複数のモデルや予測値を比較できる。 • 既存のシステムの現在の使用状況と処理能力を判断して、処理能力計画に役立てることができる。 • 既存のシステムの使用状況と処理能力に関する傾向を処理能力計画に役立てることができる。	もたらすが、テストを通じて検証できない点もある。

パフォーマンス テストによって得られると考えられるメリットは、上記に挙げた課題がもたらすデメリットをはるかに上回りますが、可変要素、シナリオ、状況の妥当な組み合わせをすべてテストすることはできないため、テスト結果の妥当性が不確かなものになり、パフォーマンス テストを行うこと自体を疑問視する組織もあります。ただし、実際には、システムで (徹底的ではなくても) 理にかなったパフォーマンス テストを実施すれば、パフォーマンスに関する重大な問題が発生する可能性が劇的に低下します。特に、チームがパフォーマンス テストを使用して運用環境の監視対象を特定し、アプリケーションでパフォーマンスに関する問題が発生しそうになったときに早期に警告を受け取るようにすれば、問題の発生を低く抑えることができます。

その他の概念と用語

パフォーマンス テストを実施するにあたり、次の用語を頻繁に見たり聞いたりすることがあるでしょう。これらの用語の中には、組織、業界、または仲間同士でよく使用されているものもあるかもしれません。ここでは、よく使用される用語や概念の中で、混同しやすく、正しく理解しておいた方がよいものについて説明します。

用語	メモ

コンポーネント テスト	"コンポーネント テスト" は、アプリケーション アーキテクチャのコンポーネントを対象とするパフォーマンス テストです。一般的にテストされるコンポーネントには、サーバー、データベース、ネットワーク、ファイアウォール、クライアント、記憶装置などがあります。
調査	"調査" は、テスト対象の製品の速度、スケーラビリティ、安定性の各特性に関連する情報を収集することを基本とするアクティビティで、製品の品質を判断または改善するのに役立ちます。調査は、多くの場合、観測された 1 つ以上のパフォーマンスの問題の根本原因に関する仮説が正しいかどうかを検証するために行われます。
スモーク テスト	"スモーク テスト" とは、パフォーマンス テストの中で最初に実施するテストで、通常の負荷でアプリケーションが正常に動作するかどうかを確認します。
単体テスト	パフォーマンス テストのコンテキストでの "単体テスト" とは、コードのモジュール、つまりアプリケーションの既存のコード ベース全体の中の論理サブセットを対象とした、パフォーマンス特性に重点を置いたテストのことです。一般にテストされるモジュールには、関数、プロシージャ、ルーチン、オブジェクト、メソッド、クラスなどがあります。多くの場合、パフォーマンスに関する単体テストは、テストするモジュールのコードを記述した開発者によって作成および実施されます。
検証テスト	"検証テスト" は、テスト対象の製品の速度、スケーラビリティ、安定性に関する特性を、製品に設定または推定された期待値と比較するテストです。

まとめ

パフォーマンス テストは、さまざまな形式で実施され、多くのリスクに対処し、多様な価値を

組織に提供する、広範かつ複雑なアクティビティです。

リスクを軽減し、コストを最小限に抑え、特定のパフォーマンス テスト プロジェクトの中で該当するテストをいつ実施すればよいかを知るためにも、さまざまな種類のパフォーマンス テストについて理解しておくことが重要です。パフォーマンス テスト プロジェクトの中でさまざまな種類のテストを採用するには、次の重要なポイントを評価する必要があります。

- パフォーマンス テストの対象。
- パフォーマンス テストのコンテキスト。テスト作業に必要なリソース、コスト、予想される結果などがあります。

第 3 章 パフォーマンス テストで対処するリスク

目的

- パフォーマンス テストの観点から、速度、スケーラビリティ、および安定性について理解する。
- パフォーマンス テストを使用して、速度、スケーラビリティ、および安定性に関連するリスクを軽減する方法を学習する。
- このようなリスクのうち、パフォーマンス テストでは十分に対処できない側面について学習する。

概要

ビジネス上の重大なリスクを管理するためにはパフォーマンス テストが必要不可欠です。たとえば、Web サイトで受信するトラフィックの量を処理できなければ、顧客は別の場所で買い物をするでしょう。パフォーマンス テストは、明確なリスクを特定するだけでなく、考えられる他の多くの問題を検出する有効な手段になります。パフォーマンス テストは他の種類のテストの代わりにはなりませんが、他の方法では入手することが難しい、有用性、機能、セキュリティ、および企業イメージに関連する情報を明らかにすることができます。

企業やパフォーマンス テスタの多くは、速度、スケーラビリティ、および安定性という 3 つのカテゴリに関してパフォーマンス テストで対処できるリスクを考えることは有用であると考えています。

本章の使い方

ここでは、パフォーマンス上の代表的なリスク、このようなリスクに関連するパフォーマンス テストの種類、およびこのようなリスクの軽減に実績のある方針について学習します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「各種パフォーマンス テストで対処されるリスクの概要」では、テストのさまざまな種類と、そのテストで軽減できるリスクを理解します。
- リスクのさまざまな種類に関するセクションでは、状況に応じて最適なテスト アプローチを判断する際に役立つ、関連する方針を理解します。

各種パフォーマンス テストで対処されるリスクの概要

パフォーマンス テストの種類	対処されるリスク
処理能力	• システムの処理能力が通常時とピーク時の両方の負荷状態で業務量をこなすかどうか。
コンポーネント	• コンポーネントが期待値を満たすかどうか。 • コンポーネントが適切に最適化されているかどうか。 • 観測されたパフォーマンスの問題の原因がコンポーネントにあるかどうか。
耐久	• 時間が経過してもパフォーマンスに一貫性があるかどうか。 • まだ検出されていない、徐々に拡大していく問題があるかどうか。 • 考慮していなかった外部干渉があるかどうか。
調査	• パフォーマンスの傾向は時間の経過と共にどの方向に向かうか。 • どのような目的で今後のテストと比較するか。
負荷	• アプリケーションに特定のワークロードが課せられている場合、望ましくない動作が発生するまでにどの程度のユーザー数を処理できるか。 • データベース サーバーやファイル サーバーで、どの程度のデータ量を処理できるか。 • ネットワーク コンポーネントが適切かどうか。
スモーク	• 新たなパフォーマンス テストに対してビルドや構成の準備が整っているかどうか。 • 次にどの種類のパフォーマンス テストを実施すべきか。 • 今回のビルドは前回のビルドに比べてパフォーマンスが向上しているかどうか。

スパイク	• 運用環境の負荷がピーク時の予想負荷を超えた場合にどのような現象が発生するか。 • どの種の障害に対して対策を立てる必要があるか。 • どのようなインジケータが必要か。
ストレス	• 運用環境の負荷が予想負荷を超えた場合にどのような現象が発生するか。 • どの種の障害に対して対策を立てる必要があるか。 • 障害の発生前に介入するために、どのようなインジケータが必要か。
単体	• コード セグメントの効率は妥当かどうか。 • パフォーマンスの予算内に収まったかどうか。 • 負荷がかかったときにコードが予想どおりに動作しているかどうか。
検証	• アプリケーションが目標や要件を満たしているかどうか。 • 今回のバージョンの速度が前回のバージョンよりも高速かどうか。 • リリースする場合、契約やサービス レベル アグリーメント (SLA) に違反するかどうか。

リスク	パフォーマンス テストの種類									
	処理能力	コンポーネント	耐久	調査	負荷	スモーク	スパイク	ストレス	単体	検証
速度関連のリスク										
ユーザー満足度			O	O	O			O		O
同期性		O	O	O	O		O	O	O	
サービスレベルアグリーメント (SLA) 違反			O	O	O					O
応答時間の傾向		O	O	O	O	O			O	
構成			O	O	O	O		O		O
一貫性		O	O	O	O				O	O
スケーラビリティ関連のリスク										
処理能力	O	O	O	O	O					O
処理量	O	O	O	O	O					O
SLA 違反			O	O	O					O
最適化	O	O		O					O	
処理効率	O	O		O					O	
将来の拡張性	O	O		O	O					O
リソース使用量	O	O	O	O	O	O	O	O	O	O
ハードウェア/環境	O	O	O	O	O		O	O		O
サービスレベルアグリーメント (SLA) 違反	O	O	O	O	O					O
安定性関連のリスク										
信頼性		O	O	O	O		O	O	O	
堅牢性		O	O	O	O		O	O	O	
ハードウェア/環境			O	O	O		O	O		O
障害モード		O	O	O	O		O	O	O	O
低速リーク		O	O	O	O				O	
サービスレベルアグリーメント (SLA) 違反		O	O	O	O		O	O	O	O

復旧		O		O			O	O	O	O
データの正確性とセキュリティ		O	O	O	O		O	O	O	O
インターフェイス		O	O	O	O			O	O	O

速度関連のリスク

速度関連のリスクがすべてエンド ユーザーの満足度につながるとは限りませんが、大半のユーザーは速度を第一義に考えます。また、速度は、特定のビジネス関連のリスクやデータ関連のリスクの要因の 1 つでもあります。
以下に、パフォーマンス テストで対処できる速度関連の最も一般的なリスクをいくつか示します。

- アプリケーションの速度がエンド ユーザーを十分満足させられるかどうか。
- アプリケーションで収集したデータが古くなる前に、ビジネスでそのデータを処理および利用できるかどうか (たとえば、月末の報告書の期限はその月の最終日の業務が終了してから 24 時間以内なのに、アプリケーションではデータの処理に 48 時間かかってしまう)。
- アプリケーションからユーザーに最新情報 (株価など) を提示できるかどうか。
- エラーがスローされるまでの予想最大応答時間以内に、Web サービスが応答するかどうか。

速度関連のリスクを軽減するための方針

速度関連のリスクの軽減には、次の方針が有効です。

- パフォーマンスの要件や目標が、アプリケーションのユーザーのニーズや希望を表していることを確認します。
- 速度の測定値を以前のバージョンや競合アプリケーションと比較します。
- 通常時と予想ピーク時の両方の実ワークロードを再現する負荷テストを設計します。
- 運用中に業務で使用されるのと同じ、データの種類、分布、および量でパフォーマンス テストを実施します (たとえば、製品数、保留状態の注文数、ユーザー ベースの規模など)。データはデータベース サーバーやファイル サーバーに蓄積できます。または、負荷テストの実行前に、必要量のデータを作成することもできます。
- 関係者がアーキテクチャやビジネスに関連する決定を行う際に、パフォーマンス テストの結果を使用して、多くの情報を利用できるようにします。
- ピーク時の予想負荷状態にあるシステムに対するユーザー満足度について示したフィードバックを要請します。
- 処理時間を重視するトランザクションをパフォーマンス テストに含めます。
- 定期的に実行されるシステム プロセス (ウイルス定義の更新プログラムのダウンロードや毎週実施されるバックアップなど) の実行中に、少なくともパフォーマンス テストの一部が実施されるようにします。
- さまざまな状態、負荷レベル、およびシナリオの組み合わせの下で速度を測定します (ユー

ザーは速度に一貫性のあることを評価します)。

- パフォーマンス テスト中に適切なデータすべてが表示および保存されることを検証します (たとえば、ユーザーが情報を更新しても、トランザクションがデータベースへの書き込みを完了していなければ、確認画面にまだ古い情報が表示されたままになります)。

スケーラビリティ関連のリスク

スケーラビリティ関連のリスクでは、アプリケーションがサポートできるユーザー数だけではなく、アプリケーションが保持および処理できるデータ量や、アプリケーションが処理能力の限界に近づいていることを特定できるかどうかも懸案事項になります。以下に、パフォーマンス テストで対処できるスケーラビリティ関連の一般的なリスクを示します。

- アプリケーションがユーザー ベース全体に一貫性のある許容応答時間を提供できるかどうか。
- アプリケーションの実行中に収集されるデータをすべて格納できるかどうか。
- アプリケーションがピーク時の処理能力の限界に近づいていることを示す警告を発生しているかどうか。
- アプリケーションの使用量が増えたときに、アプリケーションのセキュリティが確保されるかどうか。
- アプリケーションの使用量が増えたときに、アプリケーションの機能が損なわれるかどうか。
- アプリケーションがピーク時の予想負荷に耐えることができるかどうか。

スケーラビリティ関連のリスクを軽減するための方針

スケーラビリティ関連のリスクの軽減には、次の方針が有効です。

- さまざまな負荷の下で測定された速度を比較します (エンド ユーザーは、同時にアプリケーションを使用しているユーザーが他に何人いるかを意識しないことに留意してください)。
- 通常時と予想ピーク時の両方の実ワークロードを再現する負荷テストを設計します。
- 運用中に業務で使用されるのと同じ、データの種類、分布、および量でパフォーマンス テストを実施します (たとえば、製品数、保留状態の注文数、ユーザー ベースの規模など)。データはデータベース サーバーやファイル サーバーに蓄積できます。または、負荷テストの実行前に、必要量のデータを作成することもできます。
- 関係者がアーキテクチャやビジネスに関連する決定を行う際に、パフォーマンス テストの結果を使用して、多くの情報を利用できるようにします。

- 実際の要件に対応する有意義なパフォーマンス テストを実行します。
- スケーラビリティの限界がわかったら、段階的に負荷を軽減して再テストし、信頼できるインジケータとなる指標を特定できるようにします。信頼できるインジケータとは、対応策を実施するのに十分な時間的余裕を持って、アプリケーションが限界に近づいていることを警告するものです。
- 作成されたデータベース エントリをチェックしたり、特定のユーザー要求に応じて返されたコンテンツを検証したりして、さまざまな負荷状況におけるアプリケーション機能の正確性を検証します。
- ピーク時の予想負荷を上回る状態でパフォーマンス テストを実施し、代表的なユーザーや関係者がパフォーマンス テストの実施中および実施後に手動でアプリケーションにアクセスできるようにして、その動作を観察します。

安定性関連のリスク

安定性とは、信頼性、稼働時間、復元可能性などの分野を網羅する総称的な用語です。安定性関連のリスクは、通常、高負荷テスト、耐久テスト、およびストレス テストで対処されますが、場合によっては、最も基本的なパフォーマンス テストの実行中に検出されることもあります。以下に、パフォーマンス テストで対処できる安定性関連の一般的なリスクをいくつか示します。

- データが破損したり、処理速度が低下したり、サーバーの再起動が必要になったりすることなく、アプリケーションを長時間実行できるかどうか。
- アプリケーションが予期せず動作しなくなった場合に、部分的に完了したトランザクションはどうなるか。
- 予定されたダウンタイムまたは予定外のダウンタイムの後にアプリケーションが再度オンラインになった場合、ユーザーが期待するすべてのことを確認または実行できるかどうか。
- 予定外のダウンタイムの後にアプリケーションが再度オンラインになったとき、アプリケーションが適切な位置から再開されるかどうか。特に、キャンセルされたトランザクションの再開を試みないかどうか。
- エラーの組み合わせや繰り返し発生する機能的なエラーがシステム クラッシュの原因になるかどうか。
- システム規模の二次的な影響の原因となるトランザクションがあるかどうか。
- 負荷分散された環境の一部を停止しても、ユーザーに中断なくサービスを提供できるかどうか。
- システムを停止せずに修正プログラムを適用したり更新したりできるかどうか。

安定性関連のリスクを軽減するための方針

安定性関連のリスクの軽減には、次の方針が有効です。

- 現実的な耐久テストの時間をとります。
- 主要なシナリオでストレス テストを実施します。パフォーマンスの主要インジケータ (ネットワーク、ディスク、プロセッサ、メモリ) と、ビジネス インジケータ (失注数、ユーザー ログインの失敗数など) を使用します。
- 実際の運用環境と同等の業務運用を再現するデータ フィード (製品数、保留状態の注文数、ユーザー ベースの規模など) を使用してストレス テストを実施します。データはデータベース サーバーやファイル サーバーに蓄積できます。または、ストレス テストの実行前に、必要量のデータを作成することもできます。その結果、データベースやアプリケーションのデッドロックや、その他のストレス障害パターンなど、重大なエラーを再現できます。
- テスト中にサーバーをオフラインにし、残りのシステムの機能、パフォーマンス、およびデータ整合性に関する動作を観察します。
- システムの再起動の直前と直後に同じテストを実行し、結果を比較します。サービスやプロセスをリサイクルする場合に同じアプローチを使用できます。
- パフォーマンス テストのシナリオにエラーや例外の事例を含めます (間違った資格情報を使用してログオンを試みるユーザーなど)。
- パフォーマンス テスト中にシステムに修正プログラムを適用します。
- パフォーマンス テスト中にバックアップやウイルス定義の更新を強制的に実行します。

まとめ

ユーザーの満足度や、ビジネス上の目標を達成するためのアプリケーションの機能など、アプリケーション関連およびビジネス関連のリスクの大部分は、パフォーマンス テストで対処できます。

一般的に、パフォーマンス テストで対処されるリスクは、速度、スケーラビリティ、安定性の観点で分類されます。通常、速度ではエンド ユーザーが、スケーラビリティではビジネスが、安定性では技術や保守が懸案事項となります。

プロジェクト関連のリスクと、パフォーマンス テストを適用できる関連した軽減方針を特定することは、ほぼ例外なく、時間の節約につながる有用な手法と見なされます。

第 II 部

パフォーマンス テストの模範アプローチ

内容 :

- **Web アプリケーション パフォーマンス テストの主要アクティビティ**
- **パフォーマンス テストと反復処理の調整**
- **アジャイル パフォーマンス テスト サイクルの管理**
- **規制された (CMMI) 環境でのパフォーマンス テスト サイクルの管理**

第 4 章 Web アプリケーション パフォーマンス テストの主要アクティビティ

目的

- 大多数のパフォーマンス テスト プロジェクトに不可欠な 7 つの主要アクティビティについて学習する。
- タスクとプロセスを 7 つのアクティビティにどのように対応付けるかを判断できるよう、各アクティビティの詳細を理解する。
- 主要アクティビティに関連して構築できるさまざまなパフォーマンス テスト アプローチについて理解する。

概要

ここでは、アプリケーションとアプリケーションをサポートするシステムのパフォーマンス テストに関連する最も一般的な主要アクティビティについて大まかに説明します。パフォーマンス テストは、"すべてを 1 つのアプローチに" または "大部分を 1 つのアプローチに" 効果的にまとめることができない複雑なアクティビティです。プロジェクト、環境、ビジネスの原動力、受け入れ基準、テクノロジ、スケジュール、法的意味合い、利用可能なスキルやツールだけをとっても、共通で汎用的なアプローチという概念を非現実的なものにしています。

とは言え、プロジェクト レベルのパフォーマンス テストのほぼすべての作業に含まれるアクティビティもいくつか存在します。このようなアクティビティは、実行されるタイミングが異なったり、別の呼び方をされたり、重要度に違いがあったり、暗黙に行われたり、明示的に実施されたりすることもありますが、最終的には、本ガイド全体で特定および参照している 7 つの主要アクティビティに関連した意思決定が行われることのないパフォーマンス テスト プロジェクトはめったにありません。これらの 7 つの主要アクティビティ自体でパフォーマンス テストの 1 つのアプローチが構成されることはありません。どちらかと言えば、プロジェクトにとって適切なアプローチを構築できる基礎を表します。

本章の使い方

ここでは、パフォーマンス テストの主要アクティビティと、各アクティビティが実現する内容について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してくださ

い。

- 「パフォーマンス テストの主要アクティビティ一覧表」では、パフォーマンス テストの主なアクティビティの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、パフォーマンス テストの最も重要なタスクの詳細と各アクティビティの考慮事項を理解します。

アクティビティの概要

これ以降では、成功したパフォーマンス テスト プロジェクトで行われていた最も一般的な 7 つのアクティビティについて説明します。これらのアクティビティを効果的に実装するために重要な点は、各アクティビティを実行するタイミング、アクティビティの呼び方、各アクティビティに重なり合う部分があるかどうかなどではなく、概念を理解し、注意深く検討して、独自のプロジェクト コンテキストにとって最も価値ある方法で各アクティビティを当てはめることです。

大部分のチームでは、まず、プロジェクトのコンテキストを大まかに把握し、テスト環境とパフォーマンス受け入れ基準をほぼ同時に特定することから始めます。これは、アクティビティ 1 と 2 以外のすべてのアクティビティは、アクティビティ 1 と 2 で収集された情報の影響を受けるためです。一般に、チームのメンバがアプリケーション、アプリケーションのユーザーや機能、アプリケーションが抱えるパフォーマンス関連のリスクを詳しく理解していくにつれ、これらのアクティビティが定期的に見直されます。

プロジェクトのコンテキスト、テスト環境、およびパフォーマンス受け入れ基準を十分理解したら、アクティビティ 3 と 4 で説明するように、パフォーマンス テストの計画と設計、およびテスト環境の構成に着手します。テスト環境の構成では、パフォーマンス テストの実施や、現時点で必要になると予測されるデータの収集に必要なツールを使用します。繰り返しになりますが、多くの場合、利用可能な情報が増えるにつれ、これらのアクティビティを定期的に見直します。

大部分のチームは、少なくともアクティビティ 1 ～ 4 に関連する側面を達成した時点で、反復テスト サイクル (アクティビティ 5 ～ 7) に移行します。反復テスト サイクルでは、設計したテストをなんらかの負荷生成ツールを使用して実装し、実装したテストを実行して、こうしたテストの結果をその時点で使用できるコンポーネントと機能の関連性の観点から分析し、報告します。

機能やコンポーネントが使用可能になるにつれて行われる、機能やコンポーネントのテストによって、また、アプリケーション、アプリケーションのユーザー、機能、パフォーマンス関連のリスク (テストを通じて明らかになります) に関する継続的な情報収集によって必然的に生じる反復サイクルがあります。こうした反復サイクルは、システムやアプリケーションが完成する前にパフォーマンス テストを開始するレベルまで達します。

パフォーマンス テストの主要アクティビティー覧表

以下の表は、パフォーマンス テストの 7 つの主要アクティビティを、各アクティビティの最も一般的な入出力と共にまとめたものです。プロジェクトのコンテキストは、各アクティビティの重要な入力項目ですが、この表には掲載していません。

アクティビティ	入力	出力
アクティビティ 1. テスト環境の特定	• 運用の論理および物理アーキテクチャ • テストの論理および物理アーキテクチャ • 使用可能なツール	• テスト環境と運用環境の比較結果 • 環境関連の懸案事項 • 追加ツールが必要かどうかの決定
アクティビティ 2. パフォーマンス受け入れ基準の特定	• クライアントの期待値 • 軽減するリスク • ビジネス要件 • 契約上の義務	• パフォーマンス テストの合格基準 • パフォーマンスの目標と要件 • 重要な調査分野 • パフォーマンスの主要インジケータ • ビジネスの主要インジケータ
アクティビティ 3. テストの計画と設計	• 使用可能なアプリケーション機能とコンポーネント • アプリケーション使用法のシナリオ	• 概念的な方針 • テストを実行する前提条件 • 必要なツールとリソース • シミュレーション対象の

	• 単体テスト • パフォーマンス受け入れ基準	アプリケーション使用法のモデル • テストの実装に必要なテスト データ • 実装の準備が整ったテスト
アクティビティ 4. テスト環境の構成	• 概念的な方針 • 使用可能なツール • 設計済みのテスト	• 構成済みの負荷生成ツールとリソース監視ツール • パフォーマンス テストの準備が整った環境
アクティビティ 5. テスト設計の実装	• 概念的な方針 • 使用可能なツール/環境 • 使用可能なアプリケーション機能とコンポーネント • 設計済みのテスト	• 検証済みで実行可能なテスト • 検証済みのリソース監視 • 検証済みのデータ収集
アクティビティ 6. テストの実行	• タスク実行計画 • 使用可能なツール/環境 • 使用可能なアプリケーション機能とコンポーネント • 検証済みで実行可能なテスト	• テストの実行結果
アクティビティ 7. 結果の分析、レポート、および再テスト	• タスクの実行結果 • パフォーマンス受け入れ基準 • リスク、懸案事項、問題点	• 結果の分析 • 推奨事項 • レポート

パフォーマンス テストの主なアクティビティのウォークスルー

パフォーマンス テストの 7 つの主要アクティビティは、次のようにまとめることができます。

図 4.1 パフォーマンス テストの主要アクティビティ

1. **アクティビティ 1. テスト環境の特定 :** テスト チームが利用できるツールやリソースだけでなく、物理テスト環境と運用環境を特定します。物理環境には、ハードウェア構成、ソフトウェア構成、ネットワーク構成などがあります。最初にテスト環境全体を徹底的に把握することで、テストの設計や計画の効果を高め、プロジェクトの早期段階でテストの課題を見極めることができます。状況によっては、この処理をプロジェクトのライフ サイクル全体で定期的に見直す必要があります。
2. **アクティビティ 2. パフォーマンス受け入れ基準の特定 :** 応答時間、スループット、およびリソース使用率の目標と制約を特定します。一般に、応答時間にはユーザーが、スループットにはビジネスが、リソースの使用率にはシステムが関与します。また、プロジェク

トの目標や制約にとらわれずに、プロジェクトを成功と見なす基準を特定します。たとえば、パフォーマンス テストを使用して、構成設定のどの組み合わせから結果として最も望ましいパフォーマンス特性が得られるかを評価します。

3. **アクティビティ 3. テストの計画と設計 :** 主要なシナリオを特定し、代表的なユーザー間のばらつきと、そのばらつきのシミュレーションを行う方法を決定します。さらに、テスト データを定義し、収集する評価指標を確立します。実装、実行、および分析の対象となるシステム使用法のモデルに、こうした情報を編成します。このようなモデルは複数になることもあります。
4. **アクティビティ 4. テスト環境の構成 :** 機能やコンポーネントがテストで利用できるようになるまでに、各テスト方針の実行に必要なテスト環境、ツール、およびリソースを準備します。必要に応じてリソースを監視するための装備がテスト環境に備わっていることを確認します。
5. **アクティビティ 5. テスト設計の実装 :** テストの設計に従って、パフォーマンス テストを開発します。
6. **アクティビティ 6. テストの実行 :** テストを実行し、監視します。テスト、テスト データ、および収集した結果を検証します。テストやテスト環境を監視しながら、分析のために検証済みのテストを実行します。
7. **アクティビティ 7. 結果の分析、レポート、および再テスト :** 結果データを集約し、共有します。各個人と複数の機能チーム間での協力により、データを分析します。残りのテストの優先順位を付け直し、必要に応じて再実行します。 すべての測定値が受け入れ基準内に収まり、設定したしきい値を超えるものがなく、目的の情報がすべて収集された時点で、特定の構成での特定のシナリオのテストが完了します。

アクティビティ 1. テスト環境の特定

パフォーマンス テストを実行する環境と、パフォーマンス テストの実行に必要なツールや関連ハードウェアを組み合わせたものが、テスト環境になります。運用環境でのアプリケーションのパフォーマンス特性を判断することが目標であるとすると、運用環境を正確に再現し、それに負荷生成ツールやリソース監視ツールを加えたテスト環境が理想的です。運用環境を正確に再現したテスト環境はめったにありません。

テストの状態と実際の運用状態との間で、アプリケーションのハードウェア構成、ソフトウェア構成、およびネットワーク構成にどの程度類似性があるかが、パフォーマンス テストの実施内容やテストする負荷の規模を決定する際に大きな考慮事項になることがよくあります。パフォー

マンス テストに影響する物理環境やソフトウェア環境だけでなく、テスト自体の目的にも留意することが重要です。しばしば、新しいハードウェアが既存のパフォーマンスの懸案事項に対処しているという想定を検証するために、提案された新しいハードウェア インフラストラクチャに対してパフォーマンス テストを行うことがあります。

テスト環境を特定する際に重要な点は、テスト環境と運用環境の類似点と相違点を完全に把握することです。考慮する重要な点には、以下のようなものがあります。

- ハードウェア
    - 構成
    - コンピュータのハードウェア (プロセッサ、RAM など)
- ネットワーク
    - ネットワーク アーキテクチャとエンド ユーザーの所在
    - 負荷分散の意味合い
    - クラスタとドメイン ネーム システム (DNS) の構成
- ツール
    - 負荷生成ツールの制限事項
    - 監視ツールが環境に与える影響
- ソフトウェア
    - 共有環境や仮想環境にインストールまたは実行されている他のソフトウェア
    - ソフトウェア ライセンスの制約や違い
    - 記憶容量やシード データの量
    - ログ記録のレベル
- 外部要因
    - ネットワーク上の追加トラフィックの量と種類
    - 定期プロセスやバッチ プロセス、更新、またはバックアップ
    - 他のシステムとの相互作用

考慮事項

テスト環境の特性を明らかにする際は、次の点を考慮します。

- テスタが、テスト対象のアプリケーションをインストール、構成、および管理することはほとんどありませんが、テスタにサーバーやソフトウェアへのアクセス権を与え、テスタから担当管理者にアクセスできるようにしておくと便利です。

- 環境の実際状態のエミュレーションを行うために、アプリケーションがシードする必要のあるデータの種類と量を特定します。
- 重要なシステム コンポーネントを特定し、どのシステム コンポーネントがパフォーマンスに関係するかを把握します。テストで制御が及ばない統合ポイントを把握します。
- IT スタッフと懇意になります。おそらく、全ネットワーク トラフィックの監視や、現実的な数のインターネット プロトコル (IP) アドレスのシミュレーションを行うための負荷生成ツールの構成などのタスクの実行に、IT スタッフの支援が必要になるでしょう。
- 負荷分散ツールの構成をチェックします。
- DNS による名前解決を検証します。データベース接続を開く際の長い待機時間を考慮する場合もあります。
- ファイアウォール、DNS、ルーティングなどが、運用環境で生じる典型的な負荷と同じになるように生成した負荷を処理することを検証します。
- テスト環境にリソース監視ソフトウェア、診断ツールなどのユーティリティをセットアップする際は、多くの場合、システム管理者に依頼する方が適切です。

アクティビティ 2. パフォーマンス受け入れ基準の特定

一般に、開発ライフ サイクルの初期段階に、アプリケーションの目標パフォーマンス特性の特定を開始するか、少なくとも見積もることに意味があります。これを最も簡単に実現するには、ユーザーや関係者が等しく適切なパフォーマンスであると見なすパフォーマンス特性を記録します。この記録は、後の時点に定量化できます。

通常、ユーザーや関係者の満足度に関連することが多い特性の部類は、以下のとおりです。

- 応答時間 : たとえば、製品カタログは 3 秒以内に表示される必要があるなどです。
- スループット : たとえば、システムでは 1 秒間に 25 冊の書籍の注文に対応する必要があるなどです。
- リソース使用率 : たとえば、プロセッサの使用率は 75% を超えてはならないなどです。目標を設定する場合に考慮する必要がある重要なリソースには、メモリ、ディスク入出力 (I/O)、ネットワーク I/O などがあります。

考慮事項

パフォーマンスの基準を特定する際は、次の点を考慮します。

- ビジネス要件
- ユーザーの期待値
- 契約上の義務
- 規制のコンプライアンス基準と業界標準
- サービス レベル アグリーメント (SLA)
- リソース使用率の目標
- 多種多様で現実的なワークロードのモデル
- 予想される全負荷状態
- システム ストレスの状態
- すべてのシナリオとコンポーネントのアクティビティ
- パフォーマンスの主要インジケータ
- これまでにリリースしたアプリケーション
- 競合他社のアプリケーション
- 最適化の目標
- 安全性の要因、拡張の余地、およびスケーラビリティ
- スケジュール、スタッフ、予算、リソースなどの優先事項

アクティビティ 3. テストの計画と設計

パフォーマンス テストの計画と設計には、主な使用シナリオの特定、ユーザー間の妥当なばらつきの判断、テスト データの特定と生成、および収集する指標の指定が必要です。最終的には、これらの項目がワークロードやワークロードのプロファイルの基礎を提供します。

運用パフォーマンスの特性を明らかにする目的でテストを設計および計画するときは、組織が多くの情報を利用してビジネス上の意思決定を行える信頼性の高いデータを提供するために、現実の環境のシミュレーションを行うことを目標にします。現実環境に即したテスト設計により、結果データの妥当性と有用性が大幅に向上します。

アプリケーションの主な使用シナリオは、通常、アプリケーションの目標パフォーマンス特性を特定する過程で浮かび上がってきます。計画中のテスト プロジェクトにこのような状況が当てはまらない場合は、最も作成する価値の高い使用シナリオを明確に判断する必要があります。主な使用シナリオを特定する際は、次の点を考慮します。

- 契約上の義務がある使用シナリオ
- パフォーマンス テストの目標や対象から暗黙に示される、または必要とされる使用シナリオ
- 最も一般的な使用シナリオ
- ビジネス クリティカルな使用シナリオ
- パフォーマンスに大きな影響を与える使用シナリオ
- 技術上懸念のある使用シナリオ
- 関係者からの懸念のある使用シナリオ
- 注目度の高い使用シナリオ

指標を適切に特定、捕捉、および報告することで、アプリケーションのパフォーマンスを目標のパフォーマンス特性と比較する方法に関する情報が提供されます。また、指標により、アプリケーション内で問題のある領域やボトルネックを特定できます。

テストの設計中にパフォーマンス受け入れ基準に関連する指標を特定しておくと、テストの設計を実装する際に、こうした指標を収集する手法をテストに統合する場合に有効です。指標を特定する際は、目標とする明確な特性、またはこうした特性に直接的、間接的に関連するインジケータのいずれかを使用します。

考慮事項

テストを計画および設計する際は、次の点を考慮します。

- 現実に即したテストの設計は、人材、ネットワーク アクティビティ、アプリケーションと相互作用する他のシステムなど、システムの制御外にある依存関係からの影響を受けます。
- 現実に即したテストの設計は、理論や予測ではなく、実環境での使用が想定される内容に基づきます。
- 現実に即したテストの設計は、結果の信頼性を高めるため、パフォーマンス テストの価値が向上します。
- 現実に即したテストには、コンポーネント レベルのパフォーマンス テストが不可欠です。
- 現実に即したテストの設計は、実装のコストや時間が増加する可能性がありますが、ビジネスや関係者にとっての正確性は大幅に向上します。
- 非現実的なテストで得られたパフォーマンス結果から推測を行うと、システムの範囲が広がるにつれて不正確さの悪影響が増し、多くの場合、間違った意思決定につながる可能性があります。

- 価値を高める可能性のある指標を判断し、このような指標を捕捉する方法を最も適切にテストに統合する手法を判断するプロセスには、開発者と管理者を関与させます。
- ツールがテストの設計に影響する可能性があることに注意します。ツールやテストを実行できることを前提にテストを設計し、前提に誤りがあることが実証されたらテストやツールを状況に合わせて改定することが、ほぼ常に、適切なテストにつながります。テストを実行する際にツールにアクセスできないことを前提に特定のテストを設計しないでください。

現実に即したテストの設計には、次のことを含めます。

- ユーザーによる遅延や思考時間の現実的なシミュレーション。これは、テストの正確さに大きく影響します。
- ユーザーによる放棄。ユーザーは、なんらかの理由でタスクを放棄する可能性があります。
- 一般的なユーザー エラー。

アクティビティ 4. テスト環境の構成

テストの設計の実装とテストの実行のために、機能やコンポーネントがテストに利用できるようになる前にテスト環境、ツール、およびリソースを準備しておくと、こうした機能やコンポーネントが利用できるようになった時点で実施できるテストの量が大幅に増加します。

負荷生成ツールやアプリケーション監視ツールを想定通りに簡単に起動および実行できることはほとんどありません。分離したネットワーク環境のセットアップ、ハードウェアの入手、IP スプーフィング用 IP アドレスの専用バンクの調整、監視ソフトウェアとサーバー オペレーティング システムとの間のバージョンの互換性など、問題の原因がどこにあるかは別として、常に、どこからか問題が生じるものです。準備を早く始めれば、テストを開始する前に問題を解決できます。

また、プロジェクトの全工程に渡って、負荷生成環境や関連ツールの再構成、更新、追加、またはその他の強化を定期的に行うことを計画します。テスト対象のアプリケーションが同じ状態のままで、負荷生成ツールが正しく機能していたとしても、収集する指標が変化する可能性があります。こうした再計画の頻度は、監視ツールの変更や追加の程度によって異なります。

考慮事項

テスト環境を構成する際は、次の点を考慮します。

- 負荷生成ツールがボトルネックに到達するまでに、どの程度の量の負荷を生成できるかを判断します。通常、負荷生成ツールのボトルネックは、まず、メモリで生じ、次にプロセッサで生じます。
- 当然のことのようですが、リソース データの収集対象となるすべてのコンピュータ間でシステム クロックの同期がとられていることを確認することが重要です。同期がとられていれば大幅に時間を節約でき、データをすべて破棄してから、システム クロックを同期した後にテストを繰り返す必要がなくなります。
- スイッチやネットワーク カードなどのハードウェア コンポーネントに対する負荷テスト実行の正確性を検証します。たとえば、全二重モードの操作が正しく行われているか、ユーザーの待機時間や帯域幅のエミュレーションが正しく行われているかなどを確認します。
- 負荷分散構成のサーバー クラスタに関連した負荷テストの実行の正確性を検証します。同じ IP アドレスを使用することによるクライアントとサーバーの密接な関係を避ける負荷テスト技法の使用を検討します。大部分の負荷生成ツールには、負荷テスト生成ツール間で異なる IP アドレスを使用してシミュレーションを行う機能があります。
- 負荷の分散を検証する負荷テスト中に、負荷分散を構成する各サーバー間のリソースの使用率 (CPU、ネットワーク、メモリ、ディスク、単位時間あたりのトランザクション数) を監視します。

アクティビティ 5. テスト設計の実装

実行可能なパフォーマンス テストを作成する細かい手順は、ツールに大きく左右されます。ただし、使用するツールとは無関係に、パフォーマンス テストを作成する場合、通常、1 つの使用シナリオを作成してからそのシナリオを強化し、他のシナリオと組み合わせて、最終的に完全なワークロード モデルを表現します。

負荷生成ツールが、テクノロジや実践方法の進化に遅れをとることは避けられません。ツールの作成者は、その時点で最も優れたテクノロジに対するサポートを組み込むことしかできません。その場合でも、サポートを組み込む前にそのテクノロジが明確になっていなければなりません。つまり、多くの場合、テスト対象のアプリケーションがシミュレーションを行ったユーザーと現実のユーザーとの間の差異を正当に指示できない方法で、ユーザーを大まかにシミュレーションするという、ある程度現実的と思われるテストを最初に実装することがパフォーマンス テスト プロジェクトにかかわる最大の課題になります。このような状況に対して計画を立てるため、すべてがスムーズに機能するために必要な時間が予定よりも大幅に長くなることに驚かないでください。

考慮事項

テストの設計を実装する際は、次の点を考慮します。

- テスト データのフィードが正しく実装されていることを確認します。テスト データのフィードは、負荷テスト中にパラメータ置換のシミュレーションを行うために使用するデータのリポジトリで、データベース、テキスト ファイル、メモリ変数、またはスプレッドシートの形式になります。たとえば、アプリケーション データベース テスト リポジトリに全製品セットが含まれているとしても、負荷テストでは、新製品やマーケティング キャンペーンなどに関係するシナリオにより、製品セットの中でユーザーが購入したサブセットのみのシミュレーションを行う必要がある場合もあります。
- テスト データ フィードは、製品データ リポジトリのサブセットになることもあります。アプリケーション データのフィードが、データベースやその他のアプリケーション コンポーネントに正しく実装されていることを確認します。アプリケーション データのフィードは、テスト対象のアプリケーションが使用する、製品や注文のデータベースなどのデータ リポジトリです。負荷テスト スクリプトによって実行される主なユーザー シナリオで、こうしたデータのサブセットを使用することがあります。
- トランザクションの検証が正しく実装されていることを確認します。多くのトランザクションが正常であると Web サーバーから報告されますが、正しく完了していないものもあります。正しい行数でデータベースにエントリが挿入されていること、製品情報が返されること、適切なコンテンツが HTML 形式でクライアントに返されることなどを検証します。
- 非表示フィールドやその他の特殊データが正しく処理されることを確認します。これは、Web サーバーから返され、その後の要求で再送信する必要のあるデータのことです。セッション ID や 製品 ID などは次の要求に渡す前に増加する必要があります。
- パフォーマンスの主要インジケータ (KPI) の監視を検証します。
- ビジネス パフォーマンスを明確にするのに役立つ、適切なインジケータを追加します。
- 要求がパラメータを受け取る場合は、サーバー側でのキャッシュを避けるための一意データや変数がパラメータ データに正しく設定されることを確認します。
- ツールがあまり自動化されていない場合は、要求の応答時間を測定するために、テスト スクリプトに要求のラッパーを追加することを検討します。
- 一般に、スクリプトを作成する時間を節約するために設計したテストを変更するよりも、設計したテストに合うようにスクリプトを作成することに時間をかける方が価値があります。
- スクリプト開発のテストまたは検証のため、実行したテストで収集した出力データを期待

値に照らして評価することで、大きな価値が得られます。

アクティビティ 6. テストの実行

パフォーマンス テストについて考えるとき、大半の人が思いを巡らせるのはテストを実行することです。テストを実行するプロセス、フロー、および技術的な詳細は、使用するツール、環境、およびプロジェクトのコンテキストに大きく依存することは理解できます。それでも、テストを実行する際に留意する必要のある、かなり普遍的なタスクや考慮事項がいくつかあります。

現在利用できるパフォーマンス テスト関連のトレーニングの多くは、テストの実行とは、テストを開始し、想定どおりに実行されていることを確認するために監視することにすぎないとしています。このアクティビティは、実際には、単にボタンをクリックして、コンピュータを監視することよりもはるかに複雑です。

テストの実行は、次のサブタスクの組み合わせと見なすことができます。

1. テストの実行と監視をチームで調整します。
2. テスト、構成、および環境の状態とデータを検証します。
3. テストの実行を開始します。
4. テストの実行中に、スクリプト、システム、およびデータを監視し、検証します。
5. テスト完了時にすばやく結果をレビューし、テストに欠陥があったことを明確に示す兆候を探します。
6. 必要に応じて後からテストを繰り返すために必要なテスト、テスト データ、結果などの情報を保管します。
7. 開始時刻、終了時刻、結果データの名前などをログに記録します。その結果、テスト終了後にデータを順番に識別できます。

テストの実行を開始する準備として、次の項目を再確認する時間をとります。

- テスト環境が、期待する構成またはテスト用に設計した構成に一致していることを確認します。
- 指標の収集用に、テストとテスト環境の両方が正しく構成されていることを確認します。
- 実際のテストを実行する前に、簡単なスモーク テストを実行し、テスト スクリプトやリモート パフォーマンス カウンタが正しく機能していることを確認します。パフォーマンス テストのコンテキストでは、アプリケーションがすべての操作を通常の負荷で短期間正常に実行できるかを判断するようにスモーク テストを設計します。
- シナリオで他に作業が必要な場合を除いて、システムをリセットし、正式なテストの実行

を開始します。

- テスト スクリプトの実行が、シミュレーション対象のワークロード モデルを表していることを確認します。
- 現時点で対象となる主要パフォーマンス インジケータやビジネス インジケータを収集するようにテストが構成されていることを確認します。

考慮事項

テストを実行する際は、次の点を考慮します。

- データベース内の完了した注文など、データの更新に関してテストの実行を検証します。
- 負荷テスト スクリプトが、ビジネス シナリオのシミュレーションを現実に即して行うように、製品 ID や注文 ID など、適切なデータ値を使用しているかどうかを検証します。
- できる限り、各テストの実行を 1 日から 2 日に制限します。各サイクルの終了後にレビューを行い、優先順位を付け直します。
- 可能であれば、各テストを 3 回実行します。初回のテスト結果は、ダイナミック リンク ライブラリ (DLL) のロード、サーバー側キャッシュの設定、テスト対象のコードが必要とするスクリプトなどのリソースの初期化による影響を受ける可能性があります。2 回目と 3 回目の反復テストの結果がほぼ一致する場合を除いて、テストを再実行します。差異が生じた要因の判断を試みます。
- 実行中のテストを監視し、通常とは異なると感じる動作に十分注意します。直感は正しいことが多く、少なくとも貴重なインジケータにはなります。
- 共有テスト環境を使用している場合は、前々からテストのスケジュールが決まっていたとしても、(その日の) テストを開始する 5 分から 30 分前にチームに警告します。また、1 時間以上継続してテストを実行する予定がない場合はチームに連絡し、チームの作業の完了を妨げないようにします。
- 負荷を生成しているコンピュータで負荷の生成中にデータの処理、レポートの作成、ダイアグラムの描画などを行いません。このような操作は、テストの結果をゆがめる可能性があります。
- テスト結果を無意識のうちに歪曲する可能性を最小限に抑えるために、テスト中は負荷を生成するコンピュータのウイルス スキャンを停止します。
- 負荷の生成中は、監視データと結果データを後で比較できるように、テストを実行中の負荷生成環境外部にあるコンピュータからシステムに手動でアクセスします。
- 立ち上げ期間とクールダウン期間のシミュレーションを適切に行うことを忘れないでください。

- アプリケーション スクリプトのコンパイル、Web サーバー キャッシュの構築などの理由から最初の反復テストの結果を放棄しないでください。この反復テストの結果を個別に測定し、システム規模の再起動が行われた後に最初のユーザーにどのようなことが予想できるかを把握します。
- テストの実行が実際に完了することはありませんが、特定のテストの結果は最終的には収束に向かいます。価値ある情報が収集されなくなったら、他のテストに移行します。
- 観測している問題点の把握が進まないと感じている場合は、1 つ以上の変数や考えられる原因を取り除いてからテストを再実行すると効果が上がることがあります。

アクティビティ 7. 結果の分析、レポート、および再テスト

上司や関係者は、さまざまなテストからの結果だけでは満足しません。彼らは、結論と、このような結論の背景にある整理されたデータを必要とします。技術チームのメンバも、単なる結果以上のものを必要とします。彼らは、結果を取得した方法の背後にある分析、比較、および詳細を必要とします。あらゆる種類のチーム メンバがパフォーマンス結果を共有する頻度を高めることに価値があります。

結果を報告する前に、データを分析する必要があります。パフォーマンス テストから返されたデータを分析する際は、次の点を考慮します。

- 各個人と複数の機能チーム間での共同作業により、データを分析します。
- 取得したデータを分析し、結果を指標の受け入れ可能レベルまたは予想レベルと比較して、テスト対象のアプリケーションのパフォーマンスの傾向が目標に近づいているのか目標から離れていくのかを判断します。
- テストに失敗した場合、通常、診断とチューニング作業を確保します。
- ボトルネックを解消した場合は、テストを繰り返して、解消されたことを検証します。
- チームはパフォーマンス テストの結果を使用して、細かいレベルでコンポーネントを分析し、その情報を実環境に戻して、適切なテストの設計や使用法の分析に関連付けることができます。
- パフォーマンス テストの結果から多くの情報を利用して、アーキテクチャ上やビジネス上の意思決定を行うことができます。
- 多くの場合、分析により、特定のテストの結果を完全に理解するために、その後のテスト実行サイクルでは新たな指標を取得する必要があることが明らかになります。
- テスト結果をチーム全体でただちに共有し、未加工のデータを利用できるようにします。
- データの利用者と対話し、テストが目的の結果を達成し、データの意味が利用者の意図に

即していることを検証します。

- 結果がテストで定義した内容を表さないと判断した場合は、新たな情報、より適切な情報、または異なる情報を取得するようにテストを変更します。
- 現在の結果を使用して、次のテストの優先順位を設定します。
- 指標を頻繁に収集すると、データ量が膨大になります。データ量の削減を考えがちですが、貴重なデータが失われる可能性があるため、データの削減手法を使用する際は常に注意が必要です。

大半のレポートは、次の 2 つのカテゴリのいずれかに分類されます。

- **技術レポート**
  - ワークロード モデルやテスト環境などのテストの説明
  - 最低限の事前処理で容易に理解できるデータ
  - 完全なデータ セットとテスト条件へのアクセス
  - 共同作業向けの監視内容、懸案事項、疑問点、要請などの簡単な説明
- **関係者へのレポート**
  - 結果が関係する基準
  - 多くの関連データの直感的で見やすい表現
  - 基準に関連した図やグラフの口頭での簡単なまとめ
  - ワークロード モデルやテスト環境の直感的で見やすい表現
  - 関連技術レポート、完全なデータ セット、およびテスト条件へのアクセス
  - 監視内容、懸案事項、および推奨事項のまとめ

効果的なレポートを作成するには、対象のユーザーが関心のある情報を、迅速、簡単、かつ直感的な方法で提示することが重要です。効果的なレポートを実現する場合に基盤となる原則の一部を以下に示します。

- レポートを迅速かつ頻繁に提供する
- 見やすいレポートを提供する
- 直感的なレポートを提供する
- 適切な統計を使用する
- データを正しく集約する
- データを効果的にまとめる
- 対象ユーザー向けにカスタマイズする
- 強力でも事実に基づく言葉遣いを使用した簡潔な口頭でのまとめを使用する
- 関係者がデータを利用できるようにする

- 不要なデータはフィルタで除外する
- 中間結果を報告する場合は、次の複数のテスト実行サイクル向けの優先順位、懸案事項、およびブロックを含める

まとめ

パフォーマンス テストには、プロジェクトのさまざまな段階で行われる一連の共通する主要アクティビティが関与します。各アクティビティには、実現する具体的な特性やタスクがあります。これらのアクティビティは、執筆者や校閲者が体験した、熟慮され成功したパフォーマンス テスト プロジェクトすべてに存在している (または、少なくとも、こうしたアクティビティの 1 つを除外するというアクティブでリスク ベースの意思決定に含まれている) ことがわかっています。各アクティビティを詳しく理解し、プロジェクトのコンテキストに最適な方法でこれらのアクティビティを適用することが重要です。

第 5 章 パフォーマンス テストと反復処理の調整

目的

- 反復処理内でパフォーマンス テストの調整を行うアプローチについて学習する。
- プロジェクトの早期段階で重大な問題を検出して解決する方法について学習する。
- 管理を犠牲にすることなく柔軟性を最大限に高める方法について学習する。
- 上司や関係者に進捗状況と価値のインジケータを提示する方法について学習する。
- 情報を捕捉する際に、リリース スケジュールに大きな影響を与えない構造を実現する方法について学習する。
- 仕方なく変更を受け入れるのではなく、変更を積極的に取り入れるように設計されたアプローチの適用方法について学習する。

概要

ここでは、アジャイル ソフトウェア開発、eXtreme Programming (XP)、Rational Unified Process (RUP) などのソースで行われるパフォーマンス テストと反復処理の調整に関するガイダンスを提供します。また、反復処理内でパフォーマンス テストを成功に導くために必要なアクティビティの基になる概念、およびプロジェクトにすぐに適用し、大きなメリットを得ることができる具体的な実施可能項目について説明します。

パフォーマンス テストでは、ユーザー エクスペリエンスのアーキテクチャ上の側面をテストし、ソフトウェアの全体的品質を示すため、多くのソフトウェア プロジェクトにとって重大な側面となります。多くの場合、パフォーマンス テストの設定と統合には多額の費用がかかるため、プロジェクトの開発ライフ サイクルやテスト ライフ サイクルの最後までパフォーマンス テストを実施しない状況が数多く見受けられます。このようなアプローチでは、開発ライフ サイクルの終了間近に重大な問題が見つかることになり、問題を解決するのにさらに多額のコストがかかるという二次的な影響が生じる可能性があります。

反復作業サイクル内で作業する際に重要なのは、チームの連携です。このため、パフォーマンス テスタは、状況の変化に合わせて反復サイクルごとに測定および分析する対象を変更できる必要があります。

本章の使い方

ここでは、反復開発環境でのパフォーマンス テストに関連するアクティビティ、およびそれらのアクティビティとパフォーマンス テストの主要アクティビティとの関係について理解します。また、これらのアクティビティの実行中に実現できるタスクについても理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「反復的なパフォーマンス テストのアクティビティ」では、反復開発環境で実行するパフォーマンス テストの主要アクティビティの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、パフォーマンス テストの最も重要なタスクの詳細を理解します。
- さらに、「第 4 章 Web アプリケーション パフォーマンス テストの主要アクティビティ」で、パフォーマンス テストが成功するプロジェクトに関連する一般的な主要アクティビティについて理解します。これにより、主要アクティビティを支える概念をパフォーマンス テストの特定のアプローチに当てはめることができるようになります。

アプローチの概要

このアプローチでは、線遠近法の観点から、まず、ソフトウェア開発プロジェクト全体を調べてから、関係者がプロジェクトにパフォーマンス テストを導入することにした理由、およびパフォーマンス テストによってプロジェクトにもたらされると期待される価値を調べます。この調査の結果には、パフォーマンス テスト作業の合格基準に対するチームの考えが含まれます。

合格基準を大まかに把握したら、パフォーマンス テストのどのアクティビティが開発ライフ サイクルのさまざまな時点で最も大きな価値をもたらすと予想されるかをまとめることで、これらの合格基準を満たす汎用的なアプローチをガイドする全体的な方針が浮かび上がります。開発ライフ サイクルのさまざまな時点とは、主要プロジェクトの提供、チェックポイント、スプリント、反復、毎週のビルドなどです。ここでは、これらのイベントをまとめて "パフォーマンス ビルド" と呼びます。パフォーマンスの専門家やチームは、多くの場合、この方針を進化させながら、パフォーマンス テスト環境と負荷生成環境のセットアップを開始します。

テスト チームは、方針を念頭に置き、必要な環境を整えて、間もなく実施されるパフォーマンス ビルド向けに特定される主要なテストやタスクの計画を立てます。パフォーマンス ビルドを提供するときに、(現在入手可能なすべての情報に基づき) 優先順位に従って計画のタスクを実行します。その際、タスクのレポート、記録、改訂、優先順位の付け直し、追加、削除などを適宜行い、作業が進むにつれてアプリケーションと計画全体を強化します。

反復的なパフォーマンス テストのアクティビティ

このアプローチは、次の 9 つのアクティビティを使って表すことができます。

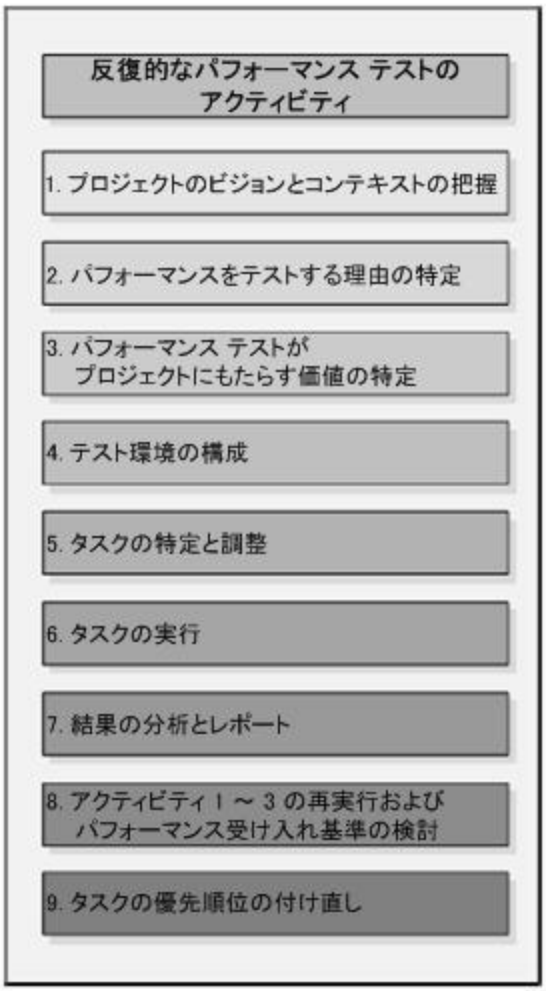


図 5.1 反復的なパフォーマンス テストのアクティビティ

- **アクティビティ 1. プロジェクトのビジョンとコンテキストの把握 :** このアクティビティの結果として、プロジェクトのビジョンとコンテキストについて共通理解を得ます。
- **アクティビティ 2. パフォーマンスをテストする理由の特定 :** パフォーマンスをテストする理由を明確に特定します。
- **アクティビティ 3. パフォーマンス テストがプロジェクトにもたらす価値の特定 :** プロジェクト レベルおよびビジネス レベルのテスト対象を、具体的で、特定可能かつ管理可能なパフォーマンス テストのアクティビティに変換します。
- **アクティビティ 4. テスト環境の構成 :** 負荷生成ツール、およびテスト対象のシステム (総

称、パフォーマンス テスト環境) を構成します。

- **アクティビティ 5. タスクの特定と調整 :** タスクの効率を高め、成功に導くために、サポート、リソース、およびスケジュールに優先順位を付け、それらを調整します。
- **アクティビティ 6. タスクの実行 :** 現在の反復処理のアクティビティを実行します。
- **アクティビティ 7. 結果の分析とレポート :** チームで結果を分析し、共有します。
- **アクティビティ 8. アクティビティ 1 ～ 3 の再実行とパフォーマンス受け入れ基準の検討 :** 各反復処理の間に基本的な情報が変更されていないことを確認します。ユーザーからのフィードバックなどの新たな情報を統合し、必要に応じて方針を更新します。
- **アクティビティ 9. タスクの優先順位の付け直し :** テスト結果、新しい情報、および機能やコンポーネントの可用性に基づいて、方針に含まれるタスクの優先順位の付け直し、追加、または削除を行い、アクティビティ 5 に戻ります。

パフォーマンス テストの主要アクティビティとの関係

次の図は、第 4 章で説明した 7 つの主要アクティビティと上記の 9 つのアクティビティとの関連を示しています。

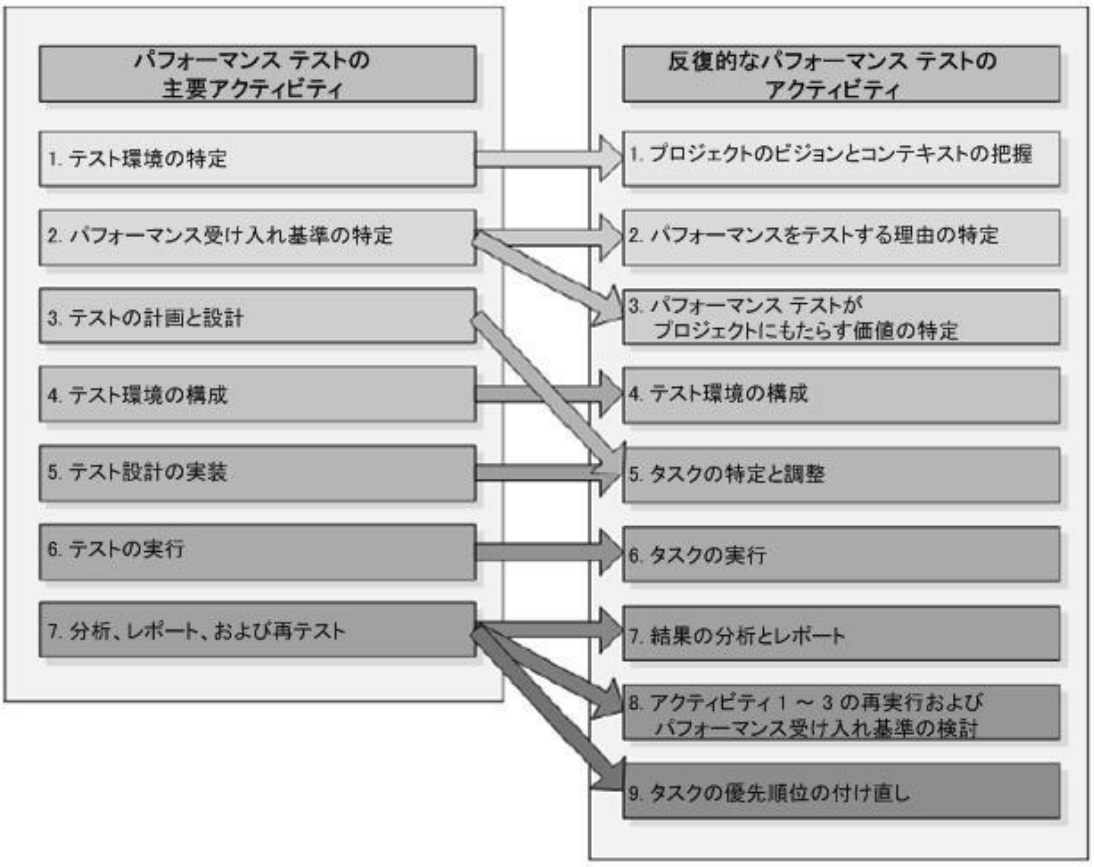


図 5.2 パフォーマンス テストの主要アクティビティとの関係

アクティビティ 1. プロジェクトのビジョンとコンテキストの把握

プロジェクトのビジョンとコンテキストは、パフォーマンス テストのどのアクティビティが必要かつ重要かを判断する際の基盤になります。パフォーマンス テスタはこのような項目を担当していないため、プロジェクトのビジョンとコンテキストがパフォーマンスに及ぼす影響についてチームを教育したり、成功に導くために今後調整が必要と考えられる領域を特定したりする場合に、調整という局面に委ねられることが多くなります。

反復処理を伴う作業では、適切な質問を問いかけ、適切な価値をもたらし、各アクティビティに関連する正しいタスクを実行することが重要です。状況が変わったり、質問、価値、またはタスクが増えたりすることはありますが、作業の開始時に使用できる各アクティビティのサンプル チェックリストを紹介します。

チェックリスト

問いかける質問 :

- プロジェクトのビジョンがパフォーマンスにどのような影響を及ぼすか。
- アプリケーションからの提供を意図しているサービスが、パフォーマンスにどのような影響を及ぼすか。または、ユーザーのためにどのような問題を解決しようとしているか。
- プロジェクトのスケジュール、構造、および利用可能なリソースに関連して、チームはパフォーマンス テストをどのように想定しているか。

もたらされる価値 :

- 製品の概念にかかわる。
- すべての懸案事項をすぐに指摘する。
- 利用可能なリソース、ツール、およびリソース監視ツールに関係する条件が生じたら、プロジェクトのビジョンとコンテキストに基づいてすぐに指摘する。

実行するタスク :

- チーム全体に質問を問いかけ、回答を得る。
- パフォーマンス テストに対するチームの認識を判断する。
- プロジェクトからパフォーマンスに及ぼす重大な影響の概念を理解する。

- パフォーマンス テストの実施に必要なツールやリソースの定義を始める。
- 予算、人員、ツールなど、リソースの制約を把握する。
- チームの調整方法を把握する。
- チームのコミュニケーション方法を把握する。

調整の対象 :

- チーム全体

アクティビティ 2. パフォーマンスをテストする理由の特定

特定のプロジェクトでパフォーマンスをテストする根本的な理由は、ビジョンとコンテキストに基づくだけでは必ずしも明確になるわけではありません。通常、プロジェクト チームは、パフォーマンス関連のリスクや懸案事項を軽減する必要性を感じている場合を除いて、プロジェクトにパフォーマンス テストを含めません。このようなリスクや懸案事項は、次の基本アクティビティで明確に特定します。このアクティビティでは、パフォーマンス テストのどのアクティビティがプロジェクトに大きな価値をもたらすかを特定します。

プロジェクトの開始当初からチームに常勤のパフォーマンス テスタを加えることには大きなメリットがありますが、あまり行われてはいません。一般に、プロジェクトの開始時にパフォーマンス テスタが参加するのは、テスタが対処すべき重大なリスクが具体的に存在する場合に限られています。

パフォーマンス テスタがチームに参加する時期には関係なく、プロジェクトのビジョンとコンテキストを把握しれたら、チームが抱えるリスクや懸案事項に基づいて、パフォーマンス テスト作業の全体的な目的を言葉や文書で表現することをお勧めします。このアクティビティの実行には、次のチェックリストが役立ちます。

チェックリスト

問いかける質問 :

- パフォーマンス テストで軽減することを目的としているこのプロジェクトのリスクは何か。
- 契約、コンプライアンス、またはユーザーのパフォーマンスに関して、必要になることが既にわかっている、具体的な期待値はあるか。
- このプロジェクトに関連して、既に存在するパフォーマンスの懸案事項は何か。

もたらされる価値 :

- 製品の概念にかかわる。
- すべての懸案事項をすぐに指摘する。
- リソースとツールに条件が生じたら、プロジェクトのビジョンとコンテキストに基づいて指摘する。
- パフォーマンス テストの対象を収集および特定するプロセスを示す。
- パフォーマンスに関する特定の懸案事項の暗黙の使用シナリオを把握する。
- 暗黙のパフォーマンスの目標、要件、対象、およびしきい値が話題に上ったときに把握する。

実行するタスク :

- チーム全体に質問を問いかけ、回答を得る。
- プロジェクト レベルのパフォーマンス テスト実施対象を特定する。
- パフォーマンス テストの実施に必要なツールやリソースの見積もりの精度を高める。
- パフォーマンス テスト作業の対象と、利用できるようになるツールやリソースとのずれを特定する。
- 後から具体化する、暗黙のパフォーマンスの目標、要件、対象、およびしきい値を把握する。
- 後から具体化する、特定の懸案事項の暗黙の使用シナリオを把握する。

調整の対象 :

- チーム全体

アクティビティ 3. パフォーマンス テストがプロジェクトにもたらす価値の特定

アクティビティ 1 と 2 で入手した情報から、パフォーマンス テストによってもたらされる価値を明確にし、その価値をパフォーマンス テストの概念的な方針にできるようになります。重要なのは、プロジェクト レベルとビジネス レベルのテスト対象を、具体的で、特定可能かつ管理可能なパフォーマンス テストのアクティビティに変換することです。このアクティビティの調整局面では、チーム全体で話し合い、パフォーマンス テストのどのアクティビティが付加価値や有益な情報を提供する可能性があるかについて合意し、現時点でそれらのアクティビティを計画する価値があるかどうかを判断します。

チェックリスト

問いかける質問 :

- パフォーマンス テストの目的を達成するのに役立つ、パフォーマンス テストのアクティビティは何か。
- 契約上、コンプライアンス、プロジェクト、またはユーザーについて、現時点でわかっているパフォーマンスの合格基準や期待値を検証するために必要なパフォーマンス テストのアクティビティは何か。
- 現在わかっているパフォーマンスの懸案事項に対処するのに役立つ、パフォーマンス テストのアクティビティは何か。

もたらされる価値 :

- パフォーマンス テストのアクティビティがチーム全体でサポートされる。
- パフォーマンス テストのアクティビティについて、新たなチーム メンバのサポートが必要になることがチームに適切に警告される。
- リソースとツールの想定が適切かどうかを判断する。
- パフォーマンス テスト対象の測定方法を決定するプロセスを示す。
- パフォーマンスに関する特定の懸案事項の新たな暗黙の使用シナリオを把握する。
- 新たな暗黙のパフォーマンスの目標、要件、対象、およびしきい値が話題に上ったときに把握する。

実行するタスク :

- チーム全体に質問を問いかけ、回答を得る。
- パフォーマンス テストを実施する目的を満たしているかどうかを判断するために、プロジェクト レベルの概念的な方針を決定する。
- パフォーマンス テストの実施に必要なツールやリソースの見積もりの精度を高める。
- パフォーマンス テスト作業の対象と、利用できるようになるツールやリソースとのずれを特定する。
- 後から具体化する、新たな暗黙のパフォーマンスの目標、要件、対象、およびしきい値を把握する。
- 後から具体化する、特定の懸案事項の新たな暗黙の使用シナリオを把握する。

調整の対象 :

- チーム全体

アクティビティ 4. テスト環境の構成

概念的な方針が適切であれば、機能とコンポーネントをテストに利用できるように、この方針の実行に必要なツールとリソースを準備します。できる限り早い段階でこのアクティビティを実行して、チームが最初からこのリソースを保持できるようにします。

このアクティビティは、非常に簡単です。負荷生成ツール、およびテスト対象のシステム (総称、パフォーマンス テスト環境) を設定し、この環境がエンジニアリングのニーズを満たすことを確認します。通常、このアクティビティの調整局面では、チームやパフォーマンス テスタが直接管理しないツールなどのリソースの取得と構成を上司や管理者に依頼します。

チェックリスト

問いかける質問 :

- テスト対象のアプリケーションのパフォーマンス テスト環境を管理するのはだれか。
- 負荷生成ツールや環境を管理するのはだれか。
- テスト対象アプリケーションのリソース監視ツールを構成および操作するのはだれか。
- 特定量の負荷を生成するには、特別な権限が必要か。
- テスト対象のアプリケーションをリセットできるのはだれか。
- 特別な調整が必要なその他のコンポーネントは何か。
- 複数ユーザーのシミュレーションを行う場合のセキュリティや認証に関する考慮事項は何か。
- ソフトウェアの記録や監視を可能にするには、どのような調整を行う必要があるか。

もたらされる価値 :

- チームが必要とするときに、負荷生成環境とパフォーマンス テスト環境が準備されている。
- パフォーマンス テスト環境のサポートを得るための連絡先をチーム全体が把握する。
- パフォーマンス テストのサポート スタッフがサポート内容を認識する。

実行するタスク :

- パフォーマンス テスト環境を構成し、テストを開始する準備を整える。
- 負荷生成環境を構成し、テストを開始する準備を整える。
- サポート担当を割り当てる。
- 特別な権限、高負荷テストの実行日時などを決定する。

調整の対象 :

- システム管理者
- ネットワーク サポート スタッフ
- データベース管理者
- インフラストラクチャ サポート スタッフ
- 上記メンバの上司
- 開発チーム

アクティビティ 5. タスクの特定と調整

パフォーマンス テストの各タスクは、それぞれが独立した状況では実行されません。パフォー

マンスの専門家は、タスクの効果を高め、成功に導くために、チームと連携してサポート、リソース、およびスケジュールに優先順位を付け、調整する必要があります。

反復計画の事前会議で、プロジェクトの現状、および次に行う必要がある作業や行うことができる作業を確認します。反復サイクルを計画する際、パフォーマンス テスタはこのサイクルで決められている目標に向かって作業を進めます。また、このサイクル中に実行されるアクティビティへのサインアップも含みます。

チェックリスト

問いかける質問 :

- このサイクルのパフォーマンス目標は何か。
- プロジェクトのパフォーマンスの全体目標から見たプロジェクトの現状は。
- システムのパフォーマンス目標はすべて達成されたか。
- 前回の反復処理以降にチューニングが行われたか。
- この反復処理中に価値をもたらす分析、レポート、または再テストは何か。
- パフォーマンス テストを実施するために連携する必要があるのはだれか。
- 利用できる時間はどの程度か。
- 各タスクにかかる時間はどの程度か。
- 最も重要なアクティビティは何か。

もたらされる価値 :

- プロジェクト全体がどの程度目標を達成しているかがわかる。
- このサイクルで測定およびレポートできるのは何かがわかる。
- 前回の反復サイクルで発生した重大な問題がすべてわかる。
- 他のチーム メンバに提案する。
- テストから明らかになったときに学習した教訓を伝える。
- 開発者と連携して、パフォーマンスの単体テストを強化する。
- 単体テストの再利用に役立つ。
- 機能テストの再利用に役立つ。

実行するタスク :

- 実行できる作業量を見積もる。
- 連携させる必要があるメンバがいるかどうかを判断する。
- 実行可能な作業に優先順位を付ける。

- このサイクルの主要タスクと代替タスクを特定する。

調整の対象 :

- 上司と関係者
- 開発者と管理者
- インフラストラクチャとテスト環境のサポート スタッフ
- ユーザーまたはユーザーの代表者

アクティビティ 6. タスクの実行

1 ～ 2 日単位でタスクを実行します。タスクを完了まで確認しますが、新たな価値がもたらされる可能性があれば、回り道をすることも重要です。アクティビティ 5 では、この反復処理でチーム メンバがサインアップする作業を定義します。この時点で、この反復処理のアクティビティを実行します。

チェックリスト

問いかける質問 :

- 最近のテスト結果やプロジェクトの更新によって、現在実施できる他のテストより、このタスクの価値が高くなったか、低くなったか。
- このタスクに関与させる新たなチーム メンバとは。
- このタスクと並行して実施できる重要なタスクは他にあるか。
- 仮の結果に意味があるか。
- 想定したデータがテストに提供されているか。

もたらされる価値 :

- アルゴリズムの効率を評価する。
- リソースの使用傾向を監視する。
- 応答時間を測定する。
- スケーラビリティと処理能力の計画に関するデータを収集する。
- テストから明らかになったときに学習した教訓を伝える。
- パフォーマンス テスタと開発者を連携させ、パフォーマンスの単体テストを向上する。
- 単体テストの再利用に役立つ。
- 機能テストの再利用に役立つ。

実行するタスク ：

- テストを実施する。
- データを収集する。
- テストの想定と技法を検証する。
- テスト中にチューニングを行う可能性がある。
- 他のチーム メンバと連携する。開発者やテスタとの連携のみでなく、ライタと連携してシステム パフォーマンスのしくみに関するライタの知識を取り込んだり、ユーザーと直接連携したりすることもあります。

調整の対象 ：

- 開発者と管理者
- インフラストラクチャとテスト環境のサポート スタッフ
- ユーザーまたはユーザーの代表者
- 上司と関係者
- 同じプロジェクトに携わっていない他のパフォーマンス テスタ

アクティビティ 7. 結果の分析とレポート

反復処理に対応していくには、結果を分析し、すばやく共有する必要があります。分析の結論が出ない場合は、できる限り早い時点で再テストします。その結果、チームがパフォーマンスの問題点に対処する時間を最大限に引き出すことができます。通常、最終リリースに向けてプロジェクトが凍結されている場合は、後から会議を開いて収集した教訓を報告します。多くの場合、毎日または 1 日おきに情報を更新して共有し、次のタスクを調整します。

チェックリスト

問いかける質問 ：

- 仮の結果に意味があるか。
- 想定したデータがテストに提供されているか。
- 役立つデータか。
- データから意味を引き出すために、さらにテストが必要か。
- チューニングは必要か。必要な場合、チューニングの対象を把握しているか。
- 結果は、まだ計画していない新たなテストを実行する必要があることを示唆しているか。
- 結果は、計画しているテストを実施する必要がなくなったことを示唆しているか。
- 達成されたパフォーマンスの目標はあるか。

- 廃止されるパフォーマンスの目標はあるか。

もたらされる価値 :

- アルゴリズムの効率を評価する。
- リソースの使用傾向を監視する。
- 応答時間を測定する。
- スケーラビリティと処理能力の計画に関するデータを収集する。
- テストから明らかになったときに学習した教訓を伝える。

実行するタスク :

- 共同でデータを分析する。
- 結果の意味を判断する。
- チーム全体でデータを共有する。
- 学習した教訓を将来の反復テスト計画に取り入れる。

調整の対象 :

- 開発者と管理者
- 上司と関係者
- ユーザーまたはユーザーの代表者
- 同じプロジェクトに携わっていない他のパフォーマンス テスタ

アクティビティ 8. アクティビティ 1 ～ 3 の再実行とパフォーマンス受け入れ基準の検討

各反復処理の間に基本的な情報が変更されていないことを確認します。ユーザーからのフィードバックなどの新たな情報を統合し、必要に応じて方針を更新します。

チェックリスト

問いかける質問 :

- プロジェクトのビジョンがパフォーマンスに及ぼす影響は変化したか。
- 提供するサービスがパフォーマンスに及ぼす影響は変化したか。または、ユーザーのために解決しようとしている問題が変化したか。
- プロジェクトのスケジュール、構造、または利用可能なリソースは変化したか。
- パフォーマンス テストの目的は変化したか。
- 契約、コンプライアンス、プロジェクト、またはユーザーのパフォーマンスの合格基準や期待値の検証に必要なパフォーマンス テストのアクティビティは変化したか。

- 現在わかっているパフォーマンスの懸案事項に対処するのに役立つ、パフォーマンス テストのアクティビティは何か。

もたらされる価値 :

- リソースとツールの想定とニーズを更新する。
- すべての懸案事項を指摘する。
- リソースとツールのニーズやリスクを指摘する。
- パフォーマンス テストの目的を更新する。
- パフォーマンスに関する特定の懸案事項の使用シナリオを強化および更新する。
- パフォーマンスの目標、要件、対象、およびしきい値を強化および更新する。
- 間もなく実施されるパフォーマンス テストのアクティビティについて、新たなチーム メンバのサポートが必要になることがチームに適切に警告される。

実行するタスク :

- プロジェクトからパフォーマンスに及ぼす重大な影響に関する知識を強化および更新する。
- 予算、人員、ツールなど、リソースの制約を更新する。
- チームの連携方法を更新および強化する。
- チームのコミュニケーション方法を更新および強化する。
- パフォーマンス テストの方針を改訂する。
- パフォーマンス テストの実施に必要なツールやリソースの見積もりの精度を高める。
- パフォーマンス テスト作業の目的と、利用できるようになるツールやリソースとの非互換性や矛盾点を特定する。
- パフォーマンスの新たな目標、要件、対象、およびしきい値を把握する。
- 特定の懸案事項の新たな使用シナリオを把握する。
- パフォーマンス テストの現在状態をレポートする。

調整の対象 :

- チーム全体

アクティビティ 9. タスクの優先順位の付け直し

テスト結果、新しい情報、および機能やコンポーネントの可用性に基づいて、方針に含まれるタスクの優先順位の付け直し、追加、または削除を行い、アクティビティ 5 に戻ります。

チェックリスト

問いかける質問 :

- 現在わかっているパフォーマンスの懸案事項に対処するのに役立つ、パフォーマンス テストのアクティビティは何か。
- このサイクルのパフォーマンス目標は何か。
- プロジェクトのパフォーマンスの全体目標から見たプロジェクトの現状は。
- システムのパフォーマンス目標はすべて達成されたか。
- 前回の反復処理以降にチューニングが行われたか。
- この反復サイクル中に価値をもたらす分析、レポート、または再テストはどれか。
- パフォーマンス テストを実施するために連携する必要があるのはだれか。
- 利用できる時間はどの程度か。
- 各タスクにかかる時間はどの程度か。
- 最も重要なアクティビティは何か。

もたらされる価値 :

- プロジェクト全体がどの程度目標を達成しているかがわかる。
- このサイクルで測定およびレポートできるのは何かがわかる。
- 前回の反復サイクルで発生した可能性がある重大な問題がすべてわかる。
- 他のチーム メンバに提案する。
- テストから明らかになったときに学習した教訓を伝える。
- 開発者と連携して、パフォーマンスの単体テストを強化する。
- 単体テストの再利用に役立つ。
- 機能テストの再利用に役立つ。

実行するタスク ：

- パフォーマンス テストの現在状態をレポートする。
- 実行できる作業量を見積もる。
- 連携させる必要があるメンバがいるかどうかを判断する。
- 実行可能な作業に優先順位を付ける。
- このサイクルの主要タスクと代替タスクを特定する。

調整の対象 ：

- 上司と関係者
- 開発者と管理者
- インフラストラクチャとテスト環境のサポート スタッフ
- ユーザーまたはユーザーの代表者

まとめ

反復処理を伴うパフォーマンス テストは、アジャイル、XP、RUP、他のソースなどの開発サイクルでは一般的な方法です。パフォーマンス テストの効果を高めるには、反復処理の計画時および反復処理中にパフォーマンス テストを正しく管理する必要があります。

第 6 章 アジャイル パフォーマンス テスト サイクルの管理

目的

- アジャイル パフォーマンス テストの管理アプローチについて学習する。
- 管理を犠牲にすることなく柔軟性を最大限に高める方法について学習する。
- 上司や関係者に進捗状況と価値のインジケータを提示する方法について学習する。
- 情報を捕捉する際に、リリース スケジュールに大きな影響を与えない構造を実現する方法について学習する。
- 仕方なく変更を受け入れるのではなく、変更を積極的に取り入れるように設計されたアプローチの適用方法について学習する。

概要

ここでは、アプリケーションのパフォーマンス テストを管理する際のアジャイル アプローチについて説明します。アジャイル アプローチの鍵は、その名前が示すように、柔軟性にあります。柔軟性といっても、非能率や非効率を意味するわけではありません。アジャイル環境に効率性やきちょうめんさを残すには、これまでパフォーマンス テストの管理に使用していた方法を変更する必要があります。

アジャイルの考え方を実装するということは、eXtreme Programming (XP) から、効率を高めるように設計された多数の短い反復処理やドキュメントを備えたプロジェクトまで、さまざまなチームにさまざまな意味があります。ここで概要を説明するアプローチは、こうした多様性を持つチームに最小限の変更で正しく適用されます。

ここでは、パフォーマンスの専門家がこうしたチームに新たに参加し、それまで推し進め、指導してきたタスクに傾注することを想定しています。これは、チームの職責の概念を最小限に抑える試みでも、役割を分離する試みでもありません。パフォーマンスの専門家が、メンバの組み合わせなど、チーム実務に参加するメンバに欠かせなくなると、そのチームは最高に機能します。役割の分離が感じられる部分は意図的なものではなく、説明を簡単にする試みの結果です。

パフォーマンス テストを管理するアプローチは、次の理由から最初は複雑に思えることがあります。

- プロジェクトのライフ サイクルを通じて変更を受け入れる。
- 繰り返す (必ずしも予測可能なパターンではありません)。

- ある程度事前にチームを調整する計画を立てることを推奨する (この事前計画は、実行するために大きな改訂を必要とするほど長期的なものではありません)。

しかし、タスクや作業項目のレベルから見ると、このアプローチは実際には、「現時点でプロジェクトに最も価値を追加するのに適したタスクは何か」と継続的に問いかけ、それに答えていくことを原則とする直感的なプロセスです。

本章の使い方

ここでは、アジャイル開発環境でのパフォーマンス テストに対するアプローチ、およびこのアプローチとパフォーマンス テストの主要アクティビティとの関係について理解します。また、これらのアクティビティの実行中に実現できるタスクについても理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「アジャイル パフォーマンス テストのアクティビティ」では、アジャイル環境でのパフォーマンス テストに対するアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、パフォーマンス テストの最も重要なタスクの詳細を理解します。
- さらに、「第 4 章 Web アプリケーション パフォーマンス テストの主要アクティビティ」で、パフォーマンス テストが成功するプロジェクトに関連する一般的な主要アクティビティについて理解します。これにより、主要アクティビティを支える概念をパフォーマンス テストの特定のアプローチに当てはめることができるようになります。

アプローチの概要

このアプローチでは、線遠近法の観点から、まず、ソフトウェア開発プロジェクト全体を調べてから、関係者がプロジェクトにパフォーマンス テストを導入することにした理由、およびパフォーマンス テストによってプロジェクトにもたらされると期待される価値を調べます。この調査の結果には、パフォーマンス テスト作業の合格基準に対するチームの考えが含まれます。

合格基準を大まかに把握したら、パフォーマンス テストのどのアクティビティが開発ライフ サイクルのさまざまな時点で最も大きな価値をもたらすと予想されるかをまとめることで、これらの合格基準を満たす汎用的なアプローチをガイドする全体的な方針が浮かび上がります。開発ライフ サイクルのさまざまな時点とは、主要プロジェクトの提供、チェックポイント、スプリント、反復、毎週のビルドなどです。ここでは、これらのイベントをまとめて "パフォーマンス ビ

ルド" と呼びます。パフォーマンスの専門家やチームは、多くの場合、この方針を進化させながら、パフォーマンス テスト環境と負荷生成環境のセットアップを開始します。

テスト チームは、方針を念頭に置き、必要な環境を整えて、間もなく実施されるパフォーマンス ビルド向けに特定される主要なテストやタスクの計画を立てます。パフォーマンス ビルドを提供するときに、(現在入手可能なすべての情報に基づき) 優先順位に従って計画のタスクを実行します。その際、タスクのレポート、記録、改訂、優先順位の付け直し、追加、削除などを適宜行い、作業が進むにつれてアプリケーションと計画全体を強化します。

アジャイル パフォーマンス テストのアクティビティ

このアプローチは、次の 9 つのアクティビティを使って表すことができます。

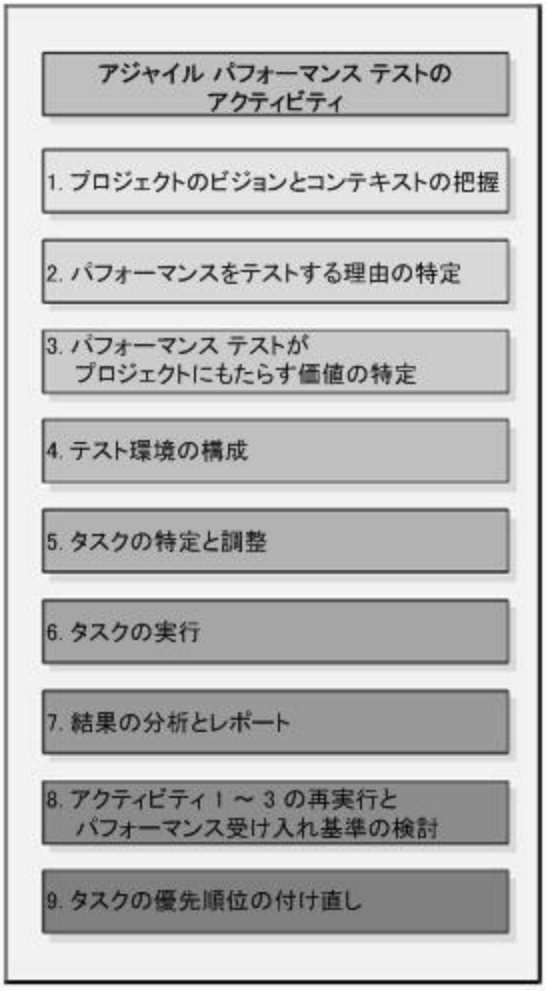


図 6.1 アジャイル パフォーマンス テストのアクティビティ

- **アクティビティ 1. プロジェクトのビジョンとコンテキストの把握 :** プロジェクトのビジョンとコンテキストは、パフォーマンス テストのどのアクティビティが必要かつ重要かを判断する際の基盤になります。
- **アクティビティ 2. パフォーマンスをテストする理由の特定 :** こうした理由は、ビジョンやコンテキストから常に明確になるわけではありません。パフォーマンス テストを行う理由を明確にするには、パフォーマンス テストのどのアクティビティがプロジェクトに最も価値を追加するかを判断することが重要です。
- **アクティビティ 3. パフォーマンス テストがプロジェクトにもたらす価値の特定 :** アクティビティ 1 と 2 で入手した情報から、パフォーマンス テストによってもたらされる価値を明確にし、その価値をパフォーマンス テストの概念的な方針に変換します。
- **アクティビティ 4. テスト環境の構成 :** 概念的な方針が適切であれば、機能とコンポーネントをテストに利用できるようになるまでに、この方針の実行に必要なツールとリソースを準備します。
- **アクティビティ 5. すぐに価値を高める限定的なタスクの特定と調整 :** パフォーマンス テストの各タスクは、それぞれが独立した状況では実行されません。このため、パフォーマンスの専門家は、タスクの効果を高め、成功に導くために、チームと連携してサポート、リソース、およびスケジュールに優先順位を付け、調整する必要があります。
- **アクティビティ 6. タスクの実行 :** 1 ～ 2 日単位でタスクを実行します。タスクを完了まで確認しますが、新たな価値がもたらされる可能性があれば、回り道をすることも重要です。
- **アクティビティ 7. 結果の分析とレポート :** 反復処理に対応していくには、結果を分析し、すばやく共有する必要があります。分析の結論が出ない場合は、できる限り早い時点に再テストを行います。その結果、チームがパフォーマンスの問題点に対処する時間を最大限に引き出すことができます。
- **アクティビティ 8. アクティビティ 1 ～ 3 の再実行とパフォーマンス受け入れ基準の検討 :** 各反復処理の間に基本的な情報が変更されていないことを確認します。ユーザーからのフィードバックなどの新たな情報を統合し、必要に応じて方針を更新します。
- **アクティビティ 9. タスクの優先順位の付け直し :** テスト結果、新しい情報、および機能やコンポーネントの可用性に基づいて、方針に含まれるタスクの優先順位の付け直し、追加、または削除を行い、アクティビティ 5 に戻ります。

パフォーマンス テストの主要アクティビティとの関係

次の図は、第 4 章で説明した 7 つの主要アクティビティと、アジャイル パフォーマンス テストの 9 つのアクティビティとの関連を示しています。

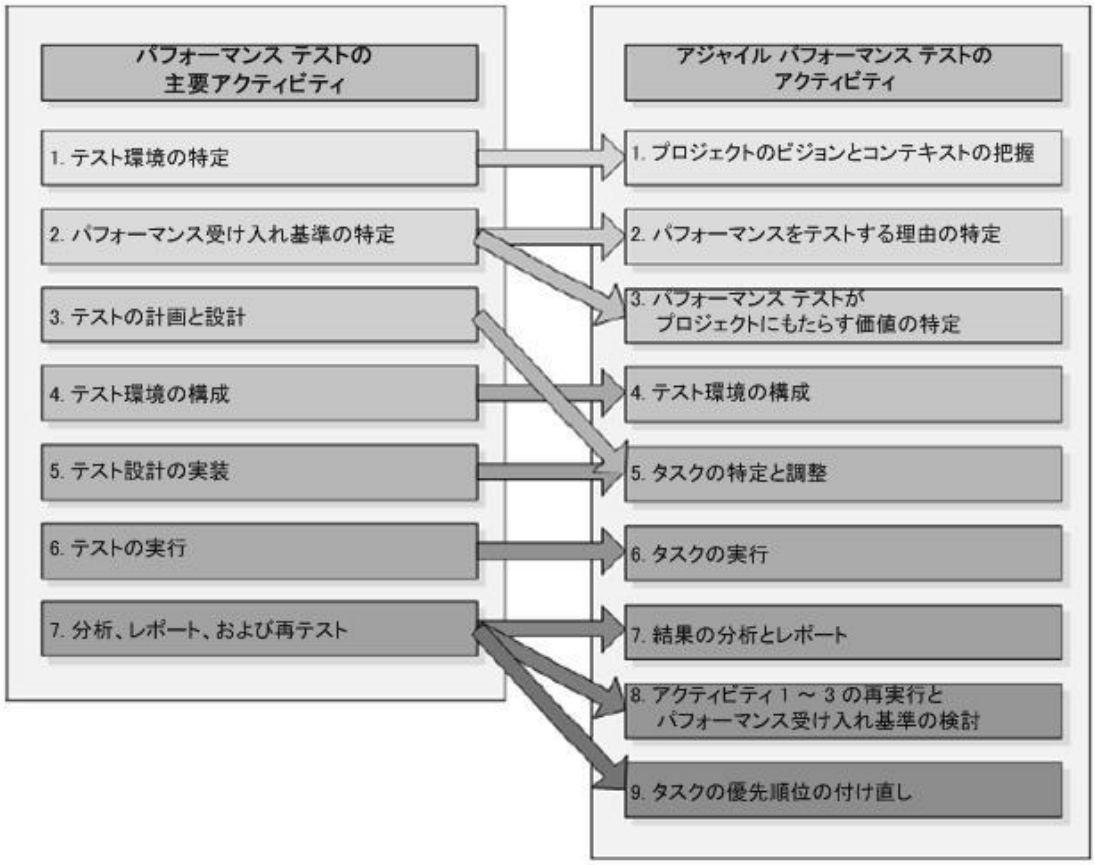


図 6.2 パフォーマンス テストの主要アクティビティとの関係

アクティビティ 1. プロジェクトのビジョンとコンテキストの把握

プロジェクトのビジョンとコンテキストは、パフォーマンス テストのどのアクティビティが必要かつ重要かを判断する際の基盤になります。テスト チームは、多くの場合、パフォーマンスの専門家、開発リーダー、プロジェクト マネージャが参加する作業セッション中に、テスト対象のシステム、プロジェクトの環境、プロジェクトを支える動機、パフォーマンス ビルド スケジュールを最初に把握します (一時的なプロジェクト スケジュールもあれば適切です)。作業セッション中に行われた決定は、システムが使いやすくなるように、反復セッションや作業セッション中にリファクタリングされることがあります。

プロジェクトのビジョン

パフォーマンス テストを開始する前に、プロジェクトの現在のビジョンを確実に理解します。機能、実装、アーキテクチャ、スケジュール、環境は、時間の経過と共に変化する可能性があり、ビジョンも同様に変化する可能性があるため、定期的に見直します。パフォーマンスについてはチームのメンバすべてが考える必要がありますが、チーム全体の関連性の詳細を率先して理解し、最新状態に保つのはパフォーマンスの専門家の役割です。次に、大まかなプロジェクト ビジョンの例を示します。

- 既存システム向けの新たなアーキテクチャを評価する。
- ビジネス上の特定の問題を解決するため、新たに独自のシステムを開発する。
- 新しいソフトウェア開発ツールを評価する。
- 新しい言語やテクノロジにチームとして上達する。
- アプリケーション エラーによってユーザーの満足度が低下しないように、ユーザーのアクティビティがピークに達する前に、不適切なアプリケーションのエンジニアリングをやり直す。

プロジェクトのコンテキスト

プロジェクトのコンテキストとは、プロジェクトのビジョンの実現に関連する、または関連すると考えられる要因にすぎません。次に、プロジェクトのコンテキストに関連すると考えられる項目の例をいくつか示します。

- クライアントの期待値
- 予算
- スケジュール
- スタッフ
- プロジェクト環境
- 管理アプローチ

こうした項目は主にプロジェクトのキックオフ ミーティングで議論されますが、テストの開発対象システムが徐々に利用できるようになり、チームがその内容を詳しく把握するにつれて、プロジェクト全体を通じて定期的に見直す必要があります。

システムを理解する

テスト対象のシステムを理解するには、システムの目的、ハードウェアとソフトウェアのアーキテクチャについて現在既知または想定されていること、および完成後のシステムの顧客やユーザーについて入手可能な情報に精通する必要があります。

多くのアジャイル プロジェクトでは、プロジェクトを進めるうちにシステムのアーキテクチャと機能が変化します。この変化を想定します。事実、少なくともこうした変化の一部は、パフォーマンス テストがきっかけとなることがよくあります。この点に留意すると、パフォーマンス テストを開始する前にそのテストの計画が過剰であったり、不足していたりすることがなくなります。

プロジェクト環境を理解する

プロジェクト環境の点からは、チームの編成、運用、およびコミュニケーションの技法を理解することが最も重要です。アジャイル チームでは、管理やコミュニケーションの手法として、長持ちするドキュメントや背景説明をあまり使用しない傾向があります。どちらかといえば、毎日の立ち話、ストーリー カード、対話形式の議論などを選択します。プロジェクトの開始時にこうした方式を理解しておかないと、パフォーマンス テストの開始が遅れる可能性があります。ここでは、次のような質問を問いかけると役に立ちます。

- チームでは定期的に会議を開催しますか、立ち話で済ませますか、それとも数人で議論しますか。
- 問題はどのように提起され、結果はどのように報告されますか。
- だれかと共同作業を行う必要がある場合、電子メール メッセージを送信しますか。会議を開催しますか。インスタント メッセージを使用しますか。相手のオフィスに出向きますか。
- チームでは、個人やチームのメンバが一定の難解なタスクを完了するまでだれにも "邪魔されたくない" と考えた場合、"入室禁止" などの措置を採用しますか。
- プロジェクト計画やプロジェクトの委員会の更新はだれが承認しますか。
- タスクはどのように割り当て、どのように追跡しますか。ソフトウェア システム、ストーリー カード、またはサインアップ。
- パフォーマンス テストの対象となるビルドをどのように判断しますか。毎日のビルド、毎週金曜日のビルド、または特殊なタグの付いたビルド。
- パフォーマンス テスト ビルドをパフォーマンス テスト環境にどのように昇格させますか。
- パフォーマンスの単体テストは開発者が作成する予定ですか。情報を共有できるように、作成した単体テストを定期的に組み合わせることはできますか。
- パフォーマンス テストの各タスクの調整はどのように行いますか。

パフォーマンス ビルドのスケジュールを理解する

この時点で、プロジェクトのスケジュール設定に着手します。卓上カレンダー、ストーリー カード、ホワイトボード、ドキュメント、だれかの記憶、ソフトウェアベースのプロジェクト管理システムなど、プロジェクト スケジュールの形式は問題ではありません。ただし、だれかが、または何かを使って、パフォーマンスの合格基準に関連する提供物、機能、ハードウェア実装の予定シーケンスを通知する必要があります。

この時点ではまだパフォーマンス テスト計画を作成していないため、日付やリソースについての懸念は重要ではないことに注意してください。代わりに、パフォーマンス ビルドの予測されるシーケンス、ビルドのコンテンツの大まかな見積もり、パフォーマンス ビルドのビルド間隔の想定時間見積もりに注意します。どのパフォーマンス ビルドに最も高い関心が寄せられるかは、どのハードウェア コンポーネント、サポート ソフトウェア、およびアプリケーション機能が調査に使用できるようになっているかに関係します。

通常、この手順の最中に、パフォーマンス ビルド固有の項目を合格基準に追加し、予測されるパフォーマンス ビルドを使って合格基準を達成することに関連するタスクの調整を開始することになります。

アクティビティ 2. パフォーマンスをテストする理由の特定

特定のプロジェクトでパフォーマンスをテストする根本的な理由は、ビジョンとコンテキストに基づくだけでは必ずしも明確になるわけではありません。パフォーマンス テストを行う理由を明確にするには、パフォーマンス テストのどのアクティビティがプロジェクトに最も価値を追加するかを判断することが重要です。

多くの場合、パフォーマンス テストを行う理由の数は、パフォーマンス受け入れ基準のリストを上回ります。各プロジェクトが、そのプロセスの一環としてパフォーマンス テストを含めるかどうかを決定する理由はさまざまです。こうした理由を特定も理解もしないことは、プロジェクトのパフォーマンス テストの側面を成功に導く確率を事実上低下させる原因の 1 つです。次に、パフォーマンス テストをプロジェクトの一部に統合する理由の例を示します。

- パフォーマンス テスタと開発者を連携させ、パフォーマンスの単体テストを向上する。
- パフォーマンス テスタと管理者を連携させ、新しいハードウェアを評価し、構成する。
- アルゴリズムの効率を評価する。
- リソースの使用傾向を監視する。
- 応答時間を測定する。

- スケーラビリティと処理能力の計画に関するデータを収集する。

一般に、プロジェクトのごく初期にパフォーマンス テストを実施する理由を特定しておくことが有効です。このような理由は、プロジェクトが進むにつれて必ず変化し、優先順位が変わるため、チームがアプリケーション、アプリケーションのパフォーマンス、アプリケーションの顧客やユーザーについての理解を深めると共に定期的に見直します。

合格基準

パフォーマンス テストを実施する理由に関連する、目標の合格基準を特定する作業もプロジェクトの早期に行っておくと有益です。テストを行う理由と同様、合格基準も必ず変化します。そのため、チームがアプリケーション、アプリケーションのパフォーマンス、アプリケーションの顧客やユーザーについての理解を深めると共に定期的に見直します。合格基準には、パフォーマンスの要件、目標、アプリケーションの対象だけでなく、実際は財政上の目的や教育上の目的など、そもそもパフォーマンス テストを実施する目的を含めます。たとえば、次のような合格基準があります。

- アプリケーションが 1 時間あたりに X 人のユーザーを処理できることを検証する。
- ユーザー操作の応答時間が Y 秒、または操作時間の 95% 未満になることを検証する。
- パフォーマンス テストの予測運用パフォーマンスの変化が±10 %以内に収まることを検証する。
- パフォーマンス上の重要な問題点を早期発見するために、利用可能になった時点でハードウェアとソフトウェアを調査する。
- パフォーマンス チーム、開発者、および管理者が連携して、最小限の管理で、アーキテクチャの処理能力のチューニングと決定を行う。
- 既存のプロジェクトの期間と予算の範囲内でパフォーマンス テストを実施する。
- 想定よりも高い負荷状態で、最も可能性の高いアプリケーションのエラー モードを判断する。
- 目標のパフォーマンス特性にとって適切なシステム構成設定を判断する。

パフォーマンスの合格基準は、プロジェクトの標準や想定に適切な方法で、チーム全体がアクセスできる場所に記録することが重要です。重要なのは、ドキュメント、ストーリー カード、チーム Wiki、タスク管理システム、ホワイトボードなど、基準を記録する方法ではなく、基準がどの程度チームで機能するかという点だけです。

多くの場合、パフォーマンス テストの合格基準の最初の決定は、パフォーマンスの専門家、開

発リーダー、プロジェクト マネージャが参加する 1 つの作業セッションで実現できます。パフォーマンス テスト作業の合格基準を明確にし、記録している段階で、まだ、パフォーマンス テスト計画を作成していないため、日付やリソースに対する懸念は重要ではありません。

一般に、パフォーマンス テストの合格基準を決定する際は、次の情報を検討します。

- アプリケーションのパフォーマンス要件と目標
- パフォーマンスに関連する対象としきい値
- 終了条件 (テストを完了するタイミングを判断する方法)
- 重要な調査分野
- 収集する重要なデータ

アクティビティ 3. パフォーマンス テストがプロジェクトにもたらす価値の特定

アクティビティ 1 と 2 で入手した情報から、パフォーマンス テストによってもたらされる価値を明確にし、その価値をパフォーマンス テストの概念的な方針に変換します。

この時点で、システム、プロジェクト、パフォーマンス テストの合格基準の最新状態を把握できたので、直近のパフォーマンス ビルドのパフォーマンス テストに関する全体方針を概念化する作業に着手できます。フィードバックや議論が活発に行われる手法を使用して、チーム全体にこの方針を伝えることが重要です。

方針では、細部をくどくどと述べたり、説明文を含めてはいけません。こうした方針は、意思決定を行いやすく、チーム全体ですぐに利用でき、だれでもメモやコメントを記入できる方法を含め、プロジェクトの進捗にあわせて簡単に変更できるようにします。

方針には広範な情報を含めることができますが、該当パフォーマンス ビルドでパフォーマンス テストを行った場合に目標とする結果と、その結果を達成すると予測されるタスクが重要な構成要素です。めったに起こりませんが、あるタスクの達成に大幅なリソース調整が必要な場合は、いくつか先のパフォーマンス ビルドまで方針を完成しておくことも意味があります。大半の方針は、パフォーマンス ビルドのリリースとほぼ同時に完成します。

議論するポイント

一般に、パフォーマンス ビルドのパフォーマンス テスト方針を用意する際に議論しておく価値

のある情報の種類は以下のとおりです。

- このビルドのパフォーマンス テストを行う理由
- 方針を実行する際の前提条件
- 必要なツールやスクリプト
- 必要な外部リソース
- 方針を実現する際のリスク
- 特別関心のあるデータ
- 懸念する領域
- 合格/不合格の基準
- 完了条件
- テストで計画している変位
- 負荷の範囲
- 方針を実現するためのタスク

アクティビティ 4. テスト環境の構成

概念的な方針が適切であれば、機能とコンポーネントをテストに利用できるようになるまでに、この方針の実行に必要なツールとリソースを準備します。

負荷生成ツールやアプリケーション監視ツールを期待通りに簡単に実装できることはほとんどありません。分離したネットワーク環境のセットアップ、ハードウェアの入手、IP スプーフィング用 IP アドレスの専用バンクの調整、監視ソフトウェアとサーバー オペレーティング システムとの間のバージョンの互換性など、問題の原因がどこにあるかは別として、常に、どこからか問題が生じるものです。

また、負荷生成ツールは、常に、テクノロジや実践方法の進化に遅れをとることは避けられません。ツールの作成者は、その時点で最も優れたテクノロジに対するサポートを組み込むことしかできません。その場合でも、サポートを組み込む前にそのテクノロジが明確になっていなければなりません。

つまり、多くの場合、テスト対象のアプリケーションがシミュレーションを行ったユーザーと現実のユーザーとの間の差異を正当に指示できない方法で、ユーザーを大まかにシミュレーションするという、ある程度現実的と思われるテストを最初に実装することがパフォーマンス テストプロジェクトにかかわる最大の課題になります。このような状況に対して計画を立てるため、すべてがスムーズに機能するために必要な時間が予定よりも大幅に長くなることに驚かないでく

ださい。

また、プロジェクトの全工程に渡って、負荷生成環境や関連ツールの再構成、更新、追加、またはその他の強化を定期的に行うことを計画します。テスト対象のアプリケーションが同じ状態のままで、負荷生成ツールが正しく機能していたとしても、収集する指標が変化する可能性があります。こうした再計画の頻度は、監視ツールの変更や追加の程度によって異なります。

アクティビティ 5. タスクの特定と調整

パフォーマンス テストの各タスクは、それぞれが独立した状況では実行されません。パフォーマンスの専門家は、タスクの効果を高め、成功に導くために、チームと連携してサポート、リソース、およびスケジュールに優先順位を付け、調整する必要があります。

ビルドが提供されるアプローチに従い、パフォーマンス テスト タスクを 1 日から 2 日単位で実行した場合に成果が上がるように、パフォーマンス テストの実施計画を作成することを考えます。つまり、おそらく、毎回のパフォーマンス ビルドごとに複数のパフォーマンス テスト実施計画を用意することになります。このような実施計画それぞれで、1 つの作業項目や作業項目のグループを完了または反復するために必要なタスクの残りの詳細を伝達する必要があります。

パフォーマンス テストの実施計画は、方針で使用しているのと同じ手法を使って、チームや関係者に連絡します。プロジェクトの進捗ペースやスケジュールによっては、方針ごとに 1 つの実施計画または複数の実施計画が存在する可能性があります。実施計画は、次のような複数の理由から、1 日か 2 日で実施できると予測されるタスクに限定することが重要です。

- 各タスクまたはタスクのグループが 1 日から 2 日で完了すると計画しても、実際に実施すると 3 日、ときには 4 日かかってしまうのは珍しいことではありません。実施に 2 日以上かかるタスクを計画していて遅延が生じた場合、前回のパフォーマンス ビルドにとって意味のあるテストを完了する前に、次のビルドが提供される可能性があります。
- アジャイル プロジェクトの場合は特に、パフォーマンスに関するタイムリーなフィードバックが重要です。タスクの実施が 2 日で完了し、パフォーマンス ビルドのサイクルが 1 週間だとしても、問題を検出してから、その問題に対処したパフォーマンス ビルドを入手するまで約 8 日間かかることになります。タスクの実施日数が増え、次のパフォーマンス ビルドまでの間隔が長くなると、この 8 日間は瞬く間に 16 日間になってしまいます。

パフォーマンス テストの実施計画は、チームで共有するために、ある程度事前に通知します。

これは、推奨事項を取り入れたり、強化したり、必要なリソース調整を行うためです。ただし、通知があまりに早すぎてもいけません。実施計画の具体性にもよりますが、あまり前に準備すると、ほぼ常に、大きな作業のやり直しにつながります。多くの場合、チーム全体でタスクの実行シーケンスに優先順位を付けます。

議論するポイント

一般に、1 つの作業項目または作業項目のグループに関するパフォーマンス テストの実施計画について議論する際に、チームが価値を見出す情報の種類は以下のとおりです。

- 作業項目の実行方法
- 収集する具体的なデータ
- データを収集する具体的な方法
- 支援する担当者、方法、およびタイミング
- 優先順位を付けた作業項目のシーケンス

アクティビティ 6. タスクの実行

1 ～ 2 日単位でタスクを実行します。タスクを完了まで確認しますが、新たな価値がもたらされる可能性があれば、回り道をすることも重要です。

各パフォーマンス ビルドが提供されたときに、そのビルドに関連し、最も優先順位の高いタスクからパフォーマンス テストを開始します。こうしたタスクには、おそらく、開発ライフ サイクルの初期であれば、「管理者と連携してアプリケーション サーバーのチューニングを行う」などの作業項目が含まれ、開発サイクルの後半になれば、「ピーク時の 50% の負荷状態で、アプリケーションが応答時間の目標を達成することを検証する」などの作業項目が含まれます。

タスクの実行で最も重要な点は、結果の分析によってタスクに新たな優先順位が付けられ、タスクやその後の方針が変更されることを忘れないことです。タスクの実行後、実行によって見つかった点をチームで共有します。その後、チームが挙げた新たな疑問点や懸案事項を基に、残りのタスクの優先順位を付け直したり、新たなタスクを追加したり、実施計画や方針から予定していたタスクを削除したりします。優先順位の付け直しが完了したら、次に最も優先順位が高いタスクに実行を移します。

パフォーマンス テストのタスクを実行する際の重要点

一般に、パフォーマンス テストのタスクを実行する際は、次の点が重要になります。

- すぐに結果を分析し、必要に応じて計画を改訂する。
- チームまたはタスクとの関連が最も高いサブチームと密接に連携する。
- チーム内で頻繁に連絡をとり、情報をオープンにする。
- 結果や見つかった重要な点を記録する。
- 後でテストを繰り返すために必要なデータを記録する。
- 実行後 2 日以内にパフォーマンス テストの優先順位を見直す。

アクティビティ 7. 結果の分析とレポート

反復処理に対応していくには、結果を分析し、すばやく共有する必要があります。分析の結論が出ない場合は、できる限り早い時点に再テストを行います。その結果、チームがパフォーマンスの問題点に対処する時間を最大限に引き出すことができます。

各タスクの完了時にデータや仮の結果を共有している場合でも、実行を定期的に一時停止して結果を集約し、傾向分析を実施して、関係者向けにレポートを作成し、開発者、アーキテクト、管理者と連携して結果を分析することが重要です。この場合の定期的とは、毎週半日、次のパフォーマンス ビルドまでに 1 日、プロジェクトのワークフローに問題なく収まるなんらかの期間などを意味します。

こうした短期の一時停止で、"大きな中断" が生じることがよくあります。継続的にレポートを作成することでチームに情報を提供することはできますが、多くの場合、こうしたレポートは電子メールの段落にスプレッドシートを添付したり、プロジェクトの Web サイトに最も関心が高いグラフへのリンクを掲載したりする形式で提供される要約にすぎません。

こうしたレポート単独で、全体的なストーリーが伝わることはめったにありません。そこで、パフォーマンスの専門家の仕事の 1 つが、時間をとって、データに含まれる傾向やパターンを見つけ出すことです。また、こうした作業により、パターンが実際に存在するかどうか、特定のテストになんらかの欠陥があるかどうかを判断するために、1 つ以上のテストを再実行する要望につながることもあります。チームではこうした手順を省きがちですが、このような省略は行わないでください。より多くのデータをより迅速に集めても、テストを一時中断して、集めたデータを定期的に見直さなければ、そのデータから有益な情報をすべて引き出すには遅すぎることになります。

アクティビティ 8. アクティビティ 1 ～ 3 の再実行とパフォーマンス受け

入れ基準の検討

各反復処理の間に基本的な情報が変更されていないことを確認します。ユーザーからのフィードバックなどの新たな情報を統合し、必要に応じて方針を更新します。

合格基準、方針、およびタスクを更新し、優先順位を付けたら、中断した位置からパフォーマンス テストのプロセスを再開します。しかし、口で言うほど簡単ではありません。ときには、どんなに一生懸命になっても、この時点で実施する価値のあるパフォーマンス テスト タスクが存在しないこともあります。このような状態は、環境のアップグレード、大幅な再設計やリファクタリング、解決に時間が必要なパフォーマンス上の問題点が検出されたことなどが原因で生じます。

プラス面を見ると、パフォーマンスの専門家がおそらくチーム全体で最も広範なスキル セットを備えていることです。つまり、このような状況が生じた場合、この時点では、パフォーマンス テストを継続することや、開発者や管理者と連携してパフォーマンスを調査することで付加価値が生じることはないため、スモーク テストの自動化、HTML の最適化、開発者と連携した、より包括的な単体テストの開発支援などの他の作業を、パフォーマンスの専門家に担当させることができます。ただし、パフォーマンスの専門家にとって優先順位の高い作業はパフォーマンス テストで、このような他の作業は付加的な役割であることを忘れないことが重要です。

アクティビティ 9. タスクの優先順位の付け直し

テスト結果、新しい情報、および機能やコンポーネントの可用性に基づいて、方針に含まれるタスクの優先順位の付け直し、追加、または削除を行い、アクティビティ 5 に戻ります。

更新の既存の構造ではパフォーマンス テスト関連の調整、レポート、または分析に時間がかかりすぎる場合、アジャイル チームの中には、定期的に "パフォーマンスに限定して" 議論や立ち話を交わすチームもあります。"パフォーマンスに限定した" 特別な集団討議や立ち話であろうと既存のセッションであろうと、優先順位、方針、タスク、および合格基準の主要な調整の大部分はチーム共同で行います。チームがパフォーマンス関連の意思決定を適切に行え、変更が容易になるような頻度で十分な時間を割り当てていることを確認します。

アジャイル パフォーマンス テスト アプローチの実装を成功に導く鍵は、チームのメンバ間で継続的にコミュニケーションをとることです。前の手順で説明したように、タスクや方針についてチームの全メンバに通知し、頻繁にメンバが相互にチェックするだけでなく、テスト スケジ

ュールの中にタスクや優先順位を見直し、更新する時間も予定することもお勧めします。

作業の大きなやり直しを必要としないで変更を導入でき、チームが現在のパフォーマンス テストの合格基準の実現に向けて作業を進めることができる限り、計画、方針、優先順位、および変更を伝達する方法はまったく重要ではありません。

その他の考慮事項

アジャイル パフォーマンス テスト サイクルを管理する場合のその他の考慮事項として、以下の点に留意してください。

- 重要な情報や発見した内容をすべてチームに伝えることを忘れないでください。
- パフォーマンス ビルドが提供される間隔の長短とは無関係に、パフォーマンス テストは常に遅れるものです。パフォーマンス テストのタスクが多すぎると、開発にリアル タイムに追随するには、開発や実行に時間がかかりすぎることになります。次に行うパフォーマンス テストの優先順位を設定するときは、この点に留意し、適切に選択します。
- パフォーマンス テストの目的は、開発ライフ サイクルの大部分向けに、設計、アーキテクチャ、および開発を通じ、パフォーマンスの向上に役立つ情報を収集することであることを覚えておいてください。エンド ユーザーに重点を置いた要件や目標に対する比較は、カスタマ レビュー リリースや製品リリース候補にとってのみ意味があります。残りの時間は、合格や不合格を検証するのではなく、傾向や明らかな問題点を探すことに使います。
- コンポーネント レベルのパフォーマンス テストには既存の単体テスト コードを使用します。その結果、テストを迅速かつ容易に実行でき、開発者がパフォーマンスの傾向を検出できるようになり、優れたスモーク テストを作成できます。
- 単にスケジュールに影響がないという理由で強制的にパフォーマンス ビルドを作成しないでください。現在のビルドがパフォーマンス テストに適していない場合は、適切になるまでできることを続けるか、適切なビルドが準備できるまでパフォーマンス テスタには別の仕事を与えます。
- パフォーマンス テストを 1 つの大きなきっかけとして、アーキテクチャ、コード、ハードウェア、および環境に重要な変更を加えます。パフォーマンス テストを使用して、見つかったパフォーマンスの問題点をチーム全体に広く公開することでメリットを得ます。パフォーマンスに関するレポートを毎日、または 2 日に 1 回報告するだけでは不十分です。チームでレポートを読み、理解し、それに対応しなければ、パフォーマンス テストの価値の大部分が失われます。

まとめ

アジャイル プロジェクト環境でのパフォーマンス テストにより、テストを極めて柔軟に管理できます。特に、このアプローチでは、パフォーマンス テストに追加された価値を基に、任意の時点で、プロジェクトのビジョンを見直し、タスクの優先順位を付け直すことができます。

第 7 章 規制された (CMMI) 環境でのパフォーマンス テスト サイクルの管理

目的

- CMMI の厳しく規制された監査可能なプロジェクトに適切な、パフォーマンス テストの管理アプローチを詳しく理解する。
- 管理やコンプライアンスを犠牲にすることなく効果を最大限に高める方法について学習する。
- 上司や関係者に進捗状況と価値のインジケータを提示する方法について学習する。
- 情報を捕捉するための構造をスケジュールに加えるのではなく、スケジュール内で実現する方法について学習する。
- 作業のやり直し、管理、または監査に関連する懸案事項を必要以上に引き起こすことなく、変化に適応するように設計されたアプローチを適用する方法について学習する。

概要

現在のソフトウェア エンジニアリング業界では、一部のシステムの複雑性と重要性から、規制による監視の必要性が生じています。監視によるプレッシャーを受けながら、システムの効率や効果を高める設計に必要な柔軟性をバランスよく維持することは常に困難です。規制への準拠と柔軟性が両立できないわけではありません。必要なのは、タスク リストを拡張し、スケジュールとエンジニアリング リソースのトレードオフをいくつか受け入れることだけです。

ここでは、Capability Maturity Model® Integration (CMMI) を、一般的に柔軟性があるとはとても言えないプロセスの実例として使用しています。CMMI は、重量級のアプローチと考えられることが多く、一般に、安全性が重要なソフトウェアや、規制標準やプロセス監査の対象となるソフトウェアにより適しています。CMMI は、カーネギーメロン大学のソフトウェア エンジニアリング研究所で開発され、次のように定義されています。

> 「Capability Maturity Model® Integration (CMMI) は、効果的なプロセスの重要な要素を組織にもたらす、プロセス改善アプローチです。CMMI を使用すると、プロジェクト、部門、または組織全体でプロセスの改善が可能になります。」

パフォーマンス テストの性質により、どの種類のテストによって価値がもたらされるか、さらには、どの種類のテストが可能であるかさえも予測することは困難です。このため、計画がなお

さら困難になることは明らかです。ここでは、パフォーマンス テストを計画および管理する、業界で検証済みのアプローチについて説明します。このアプローチは、監査を可能にする必要性、進行状況の追跡、計画を変更する際に難しい承認手順を必要としないことなどに対処しています。

本章の使い方

ここでは、規制された (CMMI) 開発環境でのパフォーマンス テストに対するアプローチ、およびこのアプローチとパフォーマンス テストの主要アクティビティとの関係について理解します。また、これらのアクティビティの実行中に実現できるタスクについても理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「CMMI パフォーマンス テストのアクティビティ」では、CMMI 環境でのパフォーマンス テストのアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、パフォーマンス テストの最も重要なタスクの詳細を理解します。
- さらに、「第 4 章 Web アプリケーション パフォーマンス テストの主要アクティビティ」で、パフォーマンス テストが成功するプロジェクトに関連する一般的な主要アクティビティについて理解します。これにより、主要アクティビティを支える概念をパフォーマンス テストの特定のアプローチに当てはめることができるようになります。

アプローチの概要

アプローチの鍵は、パフォーマンス テストを作業項目レベルで計画すること、およびプロジェクトを実現する際に各作業項目を既存の計画に合わせることにあります。その結果、コンプライアンス、監査の可能性、および承認を行うべきポイントを考慮しながら、実行の詳細は、特定の作業項目の完了を担当する担当者に委ねることができます。

このアプローチでは、線遠近法の観点から、まず、ソフトウェア開発プロジェクト全体を調べてから、関連するプロセスと標準、およびシステムのパフォーマンス受け入れ基準を調べます。この調査の結果には、パフォーマンス テスト作業の合格基準に対するチームの考えが含まれます。

合格基準と受け入れ基準を大まかに把握したら、計画とテストの設計に移ります。その結果として作成する計画とテストの設計では、開発サイクルのさまざまな時点で最も大きな価値をもたらすと予想されるパフォーマンス テストのアクティビティをまとめ、これらの基準を満たす汎用的なアプローチをガイドします。考慮する開発サイクルのさまざまな時点には、主要プロジェク

トの提供時、チェックポイント、反復時、毎週のビルドなどがあります。ここでは、これらのイベントをまとめて "パフォーマンス ビルド" と呼びます。パフォーマンスの専門家やチームは、多くの場合、この計画やテストの設計を進化させながら、テスト対象のシステムを含むパフォーマンス テスト環境と、監視ツールや負荷生成ツールを含む負荷生成環境のセットアップを開始します。

計画、テストの設計、および必要な環境が整うと、そのテストの設計を主要なテストに実装したり、直近のパフォーマンス ビルド向けて作業項目を特定したりします。特定のパフォーマンス ビルドのパフォーマンス テストが完了したら、必要に応じて、レポート作成、データの保管、およびパフォーマンス テストの計画と設計の更新を行い、適切なプロセスに従って承認を得ます。最後に、最終的なパフォーマンス ビルドのテストを行ない、最終的なレポートにまとめます。

CMMI パフォーマンス テストのアクティビティ

ここで説明するアプローチは、次の 12 個のアクティビティを使って表すことができます。

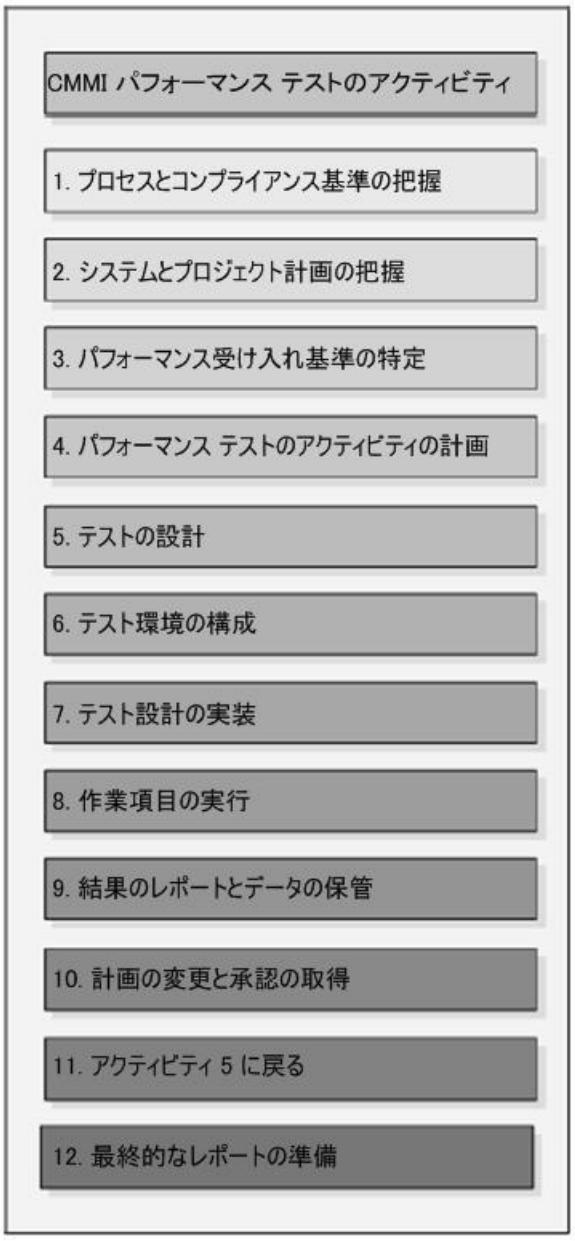


図 7.1 CMMI パフォーマンス テストのアクティビティ

- **アクティビティ 1. プロセスとコンプライアンス基準の把握 :** このアクティビティでは、プロセスとコンプライアンスの要件を把握します。
- **アクティビティ 2. システムとプロジェクト計画の把握 :** テスト対象のシステムと、その

システムの開発に関するプロジェクトの仕様を詳細に理解します。

- **アクティビティ 3. パフォーマンス受け入れ基準の特定 :** このアクティビティでは、パフォーマンスの目標や要件を特定します。また、パフォーマンス テストの対象も特定します。
- **アクティビティ 4. パフォーマンス テストのアクティビティの計画 :** このアクティビティでは、プロジェクト計画への作業項目の割り当て、期間の決定、作業の優先順位の設定、計画への詳細内容の追加などを行います。
- **アクティビティ 5. テストの設計 :** このアクティビティでは、主な使用シナリオの特定、ユーザー間の妥当なばらつきの判断、テスト データの特定と生成、および収集する指標の指定を行います。
- **アクティビティ 6. テスト環境の構成 :** このアクティビティでは、実際のテスト環境をセットアップします。
- **アクティビティ 7. テスト設計の実装 :** このアクティビティでは、テストを作成します。
- **アクティビティ 8. 作業項目の実行 :** このアクティビティでは、パフォーマンス テストの各作業項目を実行します。
- **アクティビティ 9. 結果のレポートとデータの保管 :** このアクティビティでは、結果を集約し、チーム内でデータを共有します。
- **アクティビティ 10. 計画の変更と変更に対する承認の取得 :** このアクティビティでは、必要に応じて、計画を確認して調整します。
- **アクティビティ 11. アクティビティ 5 に戻る :** このアクティビティでは、次のビルド提供時、反復時、チェックポイントのリリースでテストを繰り返し続行します。
- **アクティビティ 12. 最終的なレポートの準備 :** このアクティビティでは、最終的なレポートの作成、提出、および承認を行います。

パフォーマンス テストの主要アクティビティとの関係

次の図は、第 4 章で説明した 7 つの主要アクティビティと上記の 12 個のアクティビティとの関連を示しています。

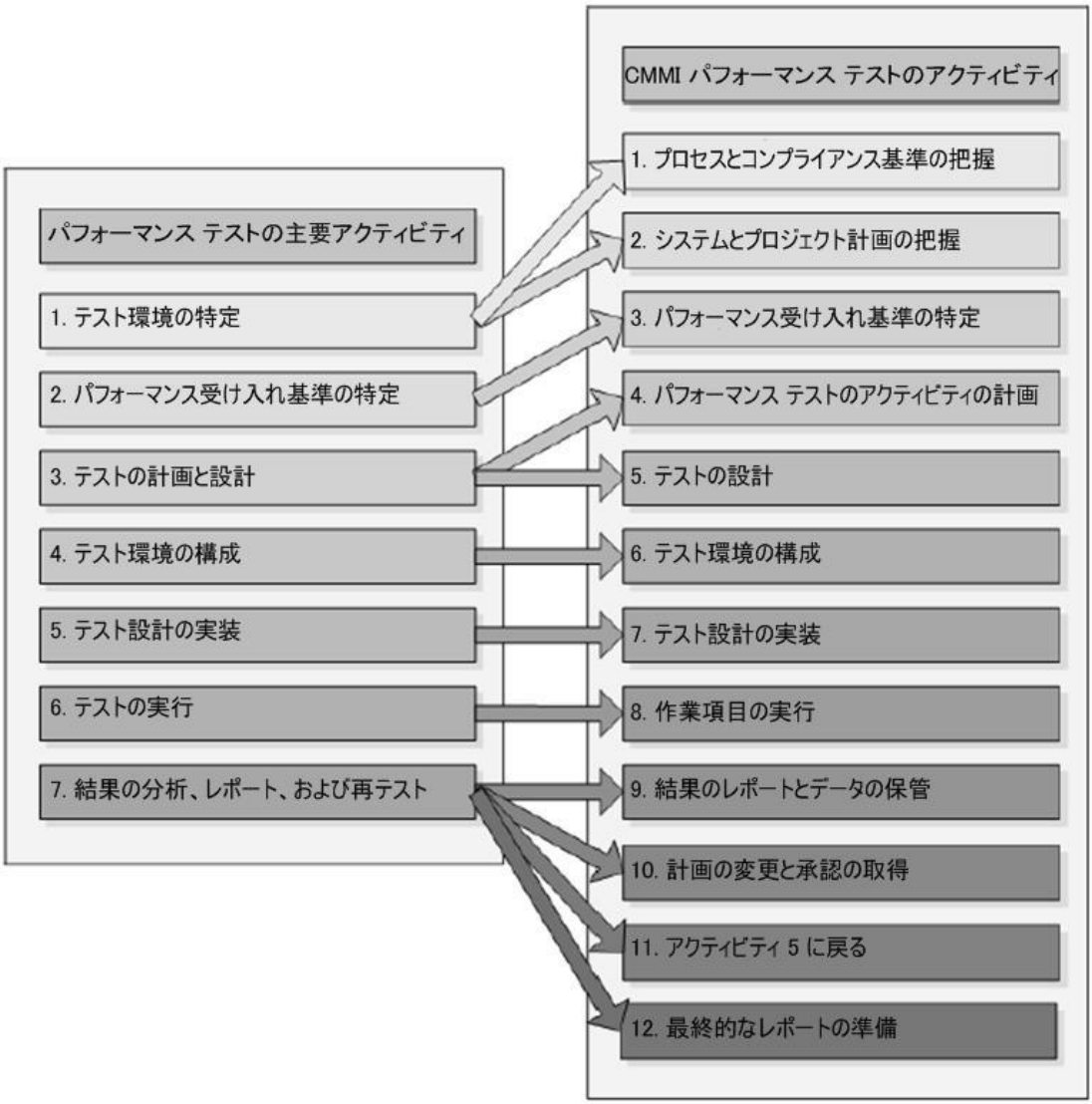


図 7.2 パフォーマンス テストの主要アクティビティとの関係

CMMI パフォーマンス テストのアクティビティ フロー

次の図は、このパフォーマンス テストのアプローチの実際の事例をさらに細かく表したものです。この図では、承認を行うべきポイント、再設計のポイント、およびチェックポイントの位置

関係に、ある程度直列的な構造があることを示しています。アクティビティ 11 からアクティビティ 5 に戻るループは、同じ基本アプローチが反復されることが示されています。

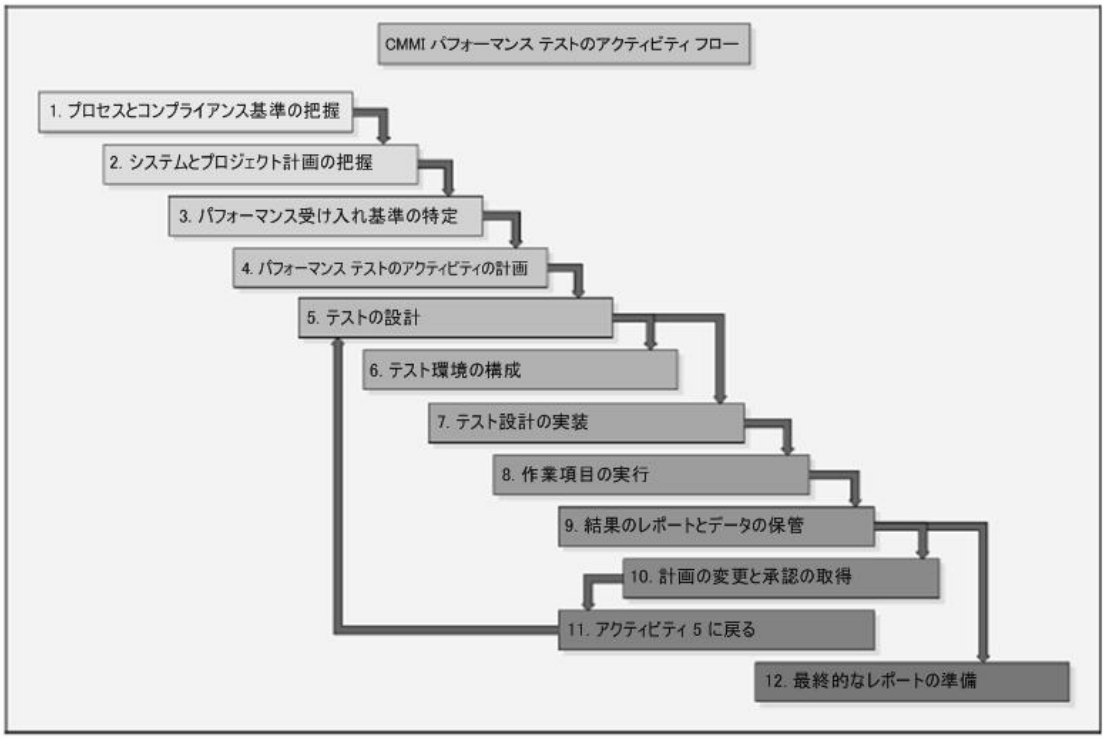


図 7.3 CMMI パフォーマンス テストのアクティビティ フロー

アクティビティ 1. プロセスとコンプライアンス基準の把握

このアクティビティは、パフォーマンス テストにはあまり関係ありませんが、パフォーマンス テストのサブプロジェクト全体を成功に導くには極めて重要です。以前実施したテストのテストデータや結果の再現に必要なプロセスの途中で問題が見つからなくても、パフォーマンス テストの監査を 2 週間以内に実施するようにスケジュールが設定されているため、十分複雑になる可能性があります。

プロセスとコンプライアンスの要件を理解しておくことは、変更要求の承認やサインオフの煩雑なプロセスでテスト作業が失敗したり難儀したりしないようにする唯一の方法なので、パフォーマンス テストの計画を始める前であっても、要件を完全に把握しておく必要があります。さいわい、このような規則や規制は、ほとんどの場合、完全に文書化されているため、このアクティビティは比較的簡単です。ただし、多くの場合、こうしたドキュメントの入手や解釈には課題があります。

プロセスを特定する

プロセス ドキュメントは、ふつう、簡単に入手できます。課題は、そのプロセスをパフォーマンス テストにどのように当てはめるかを把握および解明する点にあります。ソフトウェア開発プロセス ドキュメントでパフォーマンス テストが直接扱われていることはめったにありません。このような場合に該当するプロセスを特定するには、おそらく、ドキュメントを推敲し、可能な範囲でパフォーマンス テストを含めるようにしてから、改訂したプロセスをプロジェクト マネージャやプロセス エンジニアに提示し、承認を得るのが最適な方法です。承認を得る前に作業の反復が必要になる場合もありますが、プロジェクトが始まってからではなく、始まる前にパフォーマンス テスト プロセスの概念を提示しておくことをお勧めします。

コンプライアンス基準を確認する

規制やコンプライアンスに関するドキュメントは経営陣以外には公開されていないことが多く、入手困難なこともあります。それでも、こうした標準を確認することは重要です。コンプライアンス プロセスの確認には、テストに関連する記述に関する特定の言語や文脈も重要です。パフォーマンス テストはその性質上、機能テスト向けに開発されたプロセスには事実上従うことができません。

たとえば、多数のユーザーのシミュレーションを行うパフォーマンス テストを実施した際、このようなユーザーのうち 3 人の応答時間が記載された要件を満たさない場合、この要件には合格、不合格のどちらになるでしょうか。応答時間、量、またはワークロードを設定するテスト ケースのうち、どのテスト ケースが不合格になるでしょうか。テスト全体が不合格になるのでしょうか。テスト中に収集された他の多くの測定値はどうなるでしょうか。平均応答時間が許容範囲内だった場合、3 つの測定結果が不合格となったことで、障害レポートを 1 つだけ提出するのか、3 つについて提出するのか、それともまったく提出しないのでしょうか。具体的な標準がプロジェクトに当てはめられていたとしても、このような疑問点に直面することは多く、標準に基づいて解決する必要があります。

プロセスとコンプライアンス基準の両方を把握したら、時間をとって、適切な関係者が承認した解釈を理解します。コンプライアンスは、パフォーマンス テストと異なり、特殊なものではありません。必要応じて支援を受けてください。

アクティビティ 2. システムとプロジェクト計画の把握

プロセスとコンプライアンスの要件をしっかり把握したら、次に、テスト対象のシステムと、そのシステムの開発に関するプロジェクト仕様書を詳細に把握する必要があります。また、CMMIタイプのプロジェクトには、通常、読んでおくべきドキュメントや参考にするプロジェクト計画が多数あります。たとえば、ユース ケースのドキュメントとモデル、状態遷移図、論理アーキテクチャ図と物理アーキテクチャ図、ストーリーボード、プロトタイプ、契約、要件などがあります。こうしたドキュメントはすべて重要ですが、ひとまとめにしても、適切なパフォーマンステストの計画を作成するために必要なすべての情報が含まれているとは限りません。

システムを理解する

このようなドキュメントに記載されているシステムに関する情報は、多くの場合、各ユーザーやユーザー グループがシステムの操作方法を思い描くことが難しくなるような方法で、エンド ユーザーから集められています。このような場合は、ビジネス アナリストのスキルを使用します。次に、理解しておく必要のある点を示します。

- システムを使用するユーザー。ユーザーがシステムを使用する理由、期待値、動機。
- システムで最も多い使用シナリオ。
- システムでビジネス上重要な使用シナリオ。
- ユーザーがシステムでタスクを実現できるさまざまな方法。
- ユーザーがシステムにアクセスする頻度。
- 時間の経過と共にユーザー グループが実施するタスクの相対分布。
- 別のタイミングでシステムを操作する可能性のあるユーザーの人数。

プロジェクト計画を確認する

システム情報を入手したら、次は、プロジェクト計画にとりかかります。重要なのは、パフォーマンス テストは主たるプロジェクトではなくサブプロジェクトであることを覚えておくことです。そのため、できる限りプロジェクト全体への影響を抑え、計画に組み込む必要があります。組み込むタイミングは、マイルストーン、チェックポイント、ビルド時、および反復時です。

どの項目に注目するかは、パフォーマンス テストに使用できるようになっているハードウェア コンポーネント、サポート ソフトウェア、およびアプリケーション機能に関連します。この情報を、コンプライアンスの基準 (要件、目標、対象) やシステムとその使用方法に関して収集した情報と結び付けて考えると、不要なオーバーヘッドを追加することなく、プロジェクトに適し

たパフォーマンス テスト計画を作成できます。

アクティビティ 3. パフォーマンス受け入れ基準の特定

チームが従っているプロセスに関係なく、少なくとも、開発ライフ サイクルの初期段階でアプリケーションのパフォーマンス特性目標を特定することから開始することをお勧めします。このような特性をそれぞれ検証する方法を記録、説明、場合によっては承認しなければならないというプレッシャーが大きいときは、テストを開始する前に特定を完了しておくことがより重要になります。

パフォーマンスの要件

要件は契約上、法規上、または重要な関係者が必要とする特性であることを思い出してください。契約を確認することを拒否される場合は、コンプライアンスを判断するには、アプリケーションのパフォーマンスに関連する特定の言語や文脈の記述が重要であることを説明する必要があります。たとえば、「取引は」と「概して、取引は」という表現の違いは非常に大きく、前者はすべての取引が毎回準拠することを示しており、後者はこの後の説明で示すように、まったくあいまいです。

要件を確認するには、契約と法的拘束力のある取り決め、または開発中のソフトウェアに関連する標準に注目します。また、運用環境へのソフトウェアのリリースを拒否する原因となり得るパフォーマンス条件を幹部関係者に取り決めてもらいます。最終的な基準が特定のビジネス取引や条件に関連しているかどうかはわかりませんが、関連している場合は、このような取引や条件がパフォーマンス テストに含まれていることを確認する必要があります。

パフォーマンスの目標

パフォーマンス目標の決定はさらに難しくなる可能性があります。パフォーマンスの目標は、関係者、ユーザー、開発者、またはその他の関心のある個人が希望する特性ですが、目標がまったく達成されないからといって製品の出荷が自動的に妨げられることはありません。次に、パフォーマンスの目標を決定する際の適切な情報源を示します。

- プロジェクトのドキュメントと契約
- 関係者との面談
- 競争的分析
- 有用性に関する調査

パフォーマンス テストの対象

パフォーマンス テスタは、明示的または暗黙的な対象に必ずしも簡単にアクセスできるとは限らないため、多くの場合は、こうした対象を体系的に検索する必要があります。パフォーマンス テストの対象を特定して記録する最も簡単な方法は、プロジェクトの特定の時点、または特定のマイルストーンに達した直後にパフォーマンス テストを実施するとどのような価値を得ることができるかを、プロジェクト チームの各メンバに問い掛けることです。

チームの各メンバとの時間を見つけたり調整したりすることは必ずしも簡単ではないかもしれません (特に、プロジェクト チームに幹部関係者、アナリスト、場合によっては代表的なユーザーが含まれる場合は)。しかし、一般的には、重要なパフォーマンス テストの対象を決める際に役立つ情報をすぐに共有できます。

このような対象には、負荷がかかった状況でのリソース使用率データの提供、アプリケーション サーバーのチューニングを支援する特定の負荷の生成、各 Web ページで要求されるオブジェクト数のレポートの提供などがあります。プロジェクトのライフ サイクルの初期段階にパフォーマンス テストの対象を集めておくことが最も重要ですが、このような対象を定期的に見直し、追加された新しい対象を確認するかどうかをチーム メンバに問い掛け、必要に応じて、変更や追加に対する承認を得ることも重要です。

パフォーマンスの要件、目標、およびテストの対象を確認したら、それらをプロセスに適した方法で記録します。記録方法には、正式なドキュメントへの記載や、要件管理システムへの入力などがあります。

アクティビティ 4. パフォーマンス テストのアクティビティの計画

テスト計画はどれも適切に実行することは困難です。現実の願いとして、大幅な見直しを必要としないでプロジェクト期間内のパフォーマンス テストの各アクティビティを多少なりともガイドしていく計画を作成するには、計画のフォワード エンジニアリングとリバース エンジニアリングの両方を実行して、実行した方がよいテスト、実行しなければならないテスト、および実行できる特定のテストの実行タイミングを調整する必要があります。

プロジェクト計画に作業項目を割り当てる

これを行うには、主な提供物、マイルストーン、反復処理、およびチェックポイントに対して、

パフォーマンスの要件、目標、および対象に加え、コンプライアンス基準を割り当てます。次の表は、この割り当ての例を示しています。

	反復 1	反復 2	反復 3	チェックポイント 1
500 人のユーザーが 5 分間以上ログインすることができる (暫定要件と最終要件)。			O	√
すべてのページが 6 秒以内に応答する (目標)。	O	O	O	O
パフォーマンスとスケーラビリティを向上させるためにアプリケーション サーバーのチューニングを行う (目的)。		O	O	O
必要に応じて、後でテストと結果を再現できるように、暫定要件や最終要件の検証に使用されたテストの手順、スクリプト、データ、および結果をすべて保管する (コンプライアンス)。				√

この表の "O" は、プロジェクト計画に従って特定のテスト フェーズ中に実現できる、コンプライアンス タスクまたはテスト ケース (一般には作業項目と呼ばれます) を表します。"√" は、パフォーマンスまたはコンプライアンスの要件により、特定のテスト フェーズ中に実現しなければならない作業項目を表します。

期間を追加する

次に、各フェーズの期間と、各作業項目の予測期間を追加します。

	反復 1 1 週間	反復 2 1 週間	反復 3 1 週間	チェックポイント 1 2 週間
500 人のユーザーが 5 分間以上ログインすることができる (暫定要件と最終要件)。			O (2 日)	√ (2 日)
すべてのページが 6 秒以内に応答する (目標)。	O (2 日)	O (2 日)	O (2 日)	O (2 日)
パフォーマンスとスケーラビリティを向上させるた		O (3 日)	O (3 日)	O (3 日)

めにアプリケーション サーバーのチューニングを行う (目的)。				
必要に応じて、後でテストと結果を再現できるように、暫定要件や最終要件の検証に使用されたテストの手順、スクリプト、データ、および結果をすべて保管する (コンプライアンス)。				√ (3 日)

フェーズごとに作業項目に優先順位を付ける

前のセクションで取り上げたのは、パフォーマンス テストのアクティビティを計画する際のフォワード エンジニアリングの側面です。情報を追加した時点で、リバース エンジニアリングを適用します。どのフェーズ中にどの作業項目が実施されるかを特定し、すべての作業項目が適切に対応されるようにします。次の表に例を示します。

	反復 1 1 週間	反復 2 1 週間	反復 3 1 週間	チェックポイント 1 2 週間
500 人のユーザーが 5 分間以上ログインすることができる (暫定要件と最終要件)。		O (2 日) 計画どおり	O (2 日) 計画どおり	√ (2 日) 計画どおり
すべてのページが 6 秒以内に応答する (目標)。	O (2 日) 計画どおり	O (2 日) 計画どおり	O (2 日)	O (2 日) 計画どおり
パフォーマンスとスケーラビリティを向上させるためにアプリケーション サーバーのチューニングを行う (目的)。		O (3 日) 計画どおり	O (3 日) 計画どおり	O (3 日) 計画どおり
必要に応じて、後でテストと結果を再現できるように、暫定要件や最終要件の検証に使用されたテストの手順、スクリプト、データ、および結果をすべて保管する (コンプライアンス)。				√ (3 日) 計画どおり

計画に詳細を追加する

最後に、この情報を使用して、作業項目ごとに含める計画の詳細を追加することができます。

- 現時点でのこのテストを行う理由
- 現時点での実行の優先順位
- 実行の前提条件
- 必要なツールやスクリプト

- 必要な外部リソース
- 作業項目を完了する際のリスク
- 特別関心のあるデータ
- 懸念する領域
- 合格/不合格の基準
- 完了条件
- テストで計画している変位
- 負荷の範囲
- 収集する具体的なデータ
- データを収集する具体的な方法
- 支援する担当者、方法、およびタイミング
- 後で作業項目を繰り返す際に必要な追加情報 (必要な場合)

こうした情報がそろうと、原案となる最初のパフォーマンス テスト計画が構成されます。多くの場合、計画を実施する前に、適切な上司や関係者がこの原案を確認し、場合によっては強化して、承認します。

アクティビティ 5. テストの設計

パフォーマンス テストを設計する際に、主な使用シナリオの特定、ユーザーの妥当なばらつきの判断、テスト データの特定と生成、および収集する指標の指定を行います。最終的には、これらの項目がワークロードやワークロードのプロファイルの基礎を提供します。

テストを設計および計画する際の目的は、多くの情報を利用してビジネス上の意思決定を促進する信頼性の高いデータを提供できる、現実環境に即したテストのシミュレーションを行うことです。現実環境に即したテスト設計により、結果データの信頼性と有用性が大幅に向上します。

テスト対象のアプリケーションの主な使用シナリオは、通常、アプリケーションの目標パフォーマンス特性を特定する過程で浮かび上がってきます。計画中のテスト プロジェクトにこのような状況が当てはまらない場合は、最も作成する価値の高い使用シナリオを明確に判断する必要があります。主な使用シナリオを特定する際は次の点を考慮します。バッチ プロセスや外部アプリケーションなど、人間としてのユーザーとシステムのユーザーという両面について考慮することを忘れないでください。

- 契約上の義務がある使用シナリオ

- パフォーマンス テストの目標や対象から暗黙に示される、または必要となる使用シナリオ
- 最も一般的な使用シナリオ
- ビジネス クリティカルな使用シナリオ
- パフォーマンスに大きな影響を与える使用シナリオ
- 技術上懸念のある使用シナリオ
- 関係者からの懸念のある使用シナリオ
- 注目度の高い使用シナリオ

主な使用シナリオを特定したら、テストに向けて推敲が必要になります。この推敲プロセスには、通常、次のアクティビティがあります。

- 主なシナリオのナビゲーション パスを判断する。
- 各ユーザーのデータとばらつきを決定する。
- シナリオの相対分布を決定する。
- 対象の負荷レベルを特定する。
- テストの実行中に取得する指標を特定する。

主なシナリオのナビゲーション パスを判断する

人間の操作は予測不能なため、一般に Web サイトには、冗長な機能が用意されています。ユーザー数が比較的少ない場合でも、実在のユーザーがタスクを完了する際に行うと考えられるすべてのパスを用意するだけではなく、いや応なく予定外のパスを作成することになることもほぼ間違いありません。ユーザーがアクティビティを完了するために使用するパスごとに、システムにかかる負荷は異なります。その違いは小さい場合も、大きい場合もあります。結局、その違いはテストするまで確認できません。次のように、ナビゲーション パスを判断する方法は多数あります。

- パフォーマンスに大きな影響をもたらすことが予測され、特定された 1 つ以上の主要シナリオを実現する Web アプリケーション内のユーザー パスを特定する。
- 設計や使用法に関するマニュアルを読む。
- 自身でアクティビティの実施を試みる。
- (新しいユーザーが初めてシステムを使用する前に与えられる指示以外の) 指示なしでアクティビティの実現を試みている他のユーザーを観察する。
- 運用前のリリースと使用の調査時に取得した Web サーバー ログにある履歴データを分析する。

各ユーザーのデータとばらつきを判断する

開発とテストの初期段階では、ほとんどの場合、ユーザーのデータとばらつきは、同様のアプリケーションを操作するユーザーの想定使用法と観察結果から予測されます。一般に、Web サーバー ログに記録される履歴データが使用できるようになるにつれ、このような予測が強化または改訂されます。Web サーバー ログから読み取ったり解明したりできる、有益な指標の一部を次に示します。

- 一定期間あたりのページ表示数。ページ表示とは、依存関係のあるファイル要求 (.jpg ファイル、CSS ファイルなど) をすべて含むページ要求です。ページ表示数を時間単位、日単位、または週単位で追跡して、Web サイト上のユーザー アクティビティの周期パターンやピーク時のバーストを考慮することができます。
- 一定期間あたりのユーザー セッション数。ユーザー セッションとは、既に説明したように、ユーザーからの Web サイトへのアクセスによって生じる、関連する要求のシーケンスです。ページ表示数と同様、ユーザー セッション数を時間単位、日単位、および週単位に追跡します。
- セッション持続時間。この指標は、最初のページ要求から最後のページ要求が完了するまでを測定した、ユーザー セッションの持続時間を表します。これには、ユーザーがページ間を移動する際に一時停止する時間も含まれます。
- ページ要求の分布。この指標は、機能の種類 (ホーム、ログイン、支払いなど) に従ってページのヒット数の分布を比率で表します。分布の比率により、ユーザーによる実際の Web サイトの利用に基づいた、ページ ヒット数の重み付け比率が決まります。
- 対話の速度。この指標は、"ユーザー思考時間"、"ページ表示時間"、"ユーザーによる遅延" などとも呼ばれ、ユーザーが Web サイトを移動する際のページ間の移行にかかる時間 (思考時間から構成されます) を表します。各ユーザーの Web サイトとの対話速度がそれぞれ異なることを覚えておくことが重要です。
- ユーザーによる放棄。この指標は、ユーザーがページの読み込みを始めてから、表示の待ち時間が長くて不満がつのり、サイトの表示を終了して、ユーザー セッションを放棄することを表します。セッションが放棄されることは、インターネット上では極めて通常のことなので、負荷テストの結果に影響します。

シナリオの相対分布を決定する

シミュレーションを行うシナリオと、そのシナリオ用の手順と関連データを決定し、このようなシナリオを 1 つ以上のワークロード モデルに集約します。そこでワークロード モデルの完成

に必要な他のアクティビティと比較して、モデルで表される各アクティビティの実行頻度を確認する必要があります。

1 つのワークロード分布では不十分な場合もあります。調査と経験により、ユーザーのアクティビティは時間の経過と共に大きく変化することが示されています。テストの有効性を確実にするには、時刻、曜日、日付、季節に応じてアクティビティを評価して検証する必要があります。次に、アクティビティの相対分布を決定するための最も一般的な方法を示します。

- 実際の用途、負荷の値、一般的と一般的ではない使用シナリオ (ユーザーのパス)、クリック間やページ間のユーザーによる遅延時間、入力データの変化などは、ログ ファイルから直接取得します。
- 求められる機能、つまり使用される可能性が最も高い機能を知るために、新機能について販売部門やマーケティング部門の担当者と面談します。既存のユーザーと面談すると、こうしたユーザーがどの新機能を使用する可能性が最も高いと考えているかを判断することもできます。
- 代表的なユーザー グループ (予想されるユーザー ベースの約 10 ～ 20% の規模) にベータ リリースを展開し、サイトの使用状況からログ ファイルを分析します。
- 従業員、顧客、クライアント、友人、家族などに依頼して社内で簡単な実験を行い、ユーザーの自然なパスや、新しいユーザーとリピート ユーザーとのページ表示時間の違いなどを確認します。
- 最後の手段として、自身のサイトに関する知識に基づいて直感を使用したり、最適な想定を行ったりして予測を行います。

モデルがパフォーマンス テストに十分適していると確信したら、実際のテストの作成に必要なデータがすべてモデルに含まれるように、これまで収集した個々の使用データでモデルを補強します。

対象の負荷レベルを特定する

顧客から Web サイトへの 1 回のアクセスは、ユーザー セッションという一連の関連要求で構成されます。同じ Web サイトを操作するユーザーでも操作方法が異なれば、セッション中の Web サーバーへの要求が重なり合うことはあまりありません。そのため、ユーザー エクスペリエンスを同時実行ユーザーに基づいてモデル化するのではなく、ユーザー セッションに基づいてモデル化する方が役立ちます。ユーザー セッションは、Web サイトにアクセスする顧客によって行われる、ページの移動フロー内の一連のアクションと定義できます。

ある程度の履歴データが揃わなくても、対象の負荷レベルが特定対象になることは間違いありません。大半の場合、こうした対象は、アプリケーションに関連する目標と、その目標が市場浸透、収入生成、その他のいずれであるかに基づいて、ビジネスによって設定されます。このような対象は、当初、希望する数値を表します。

運用前リリースや現在実装しているアプリケーションの Web サーバー ログを利用できるようになったら、すぐにこうしたログのデータを使用して、上記のリソースを使用して収集したデータを検証または拡張できます。Web サーバー ログの定量分析を行うことで、次の点を確認できます。

- 一定期間内 (月/週/日) でのサイトのアクセス総数。
- 全体の平均負荷とピーク時の負荷の点から見た使用量 (時間単位)。
- 全体の平均負荷とピーク時の負荷に関するセッションの持続時間 (時間単位)。
- 全体の平均負荷とピーク時の負荷 (時間単位)。この値は、実際のスケーラビリティの量をシミュレーションするために、重なり合うユーザー セッション数に変換されます。

量に関する情報を、これまでの手順で特定および決定した、目標、主なシナリオ、ユーザーによる遅延、ナビゲーション パス、およびシナリオの分布と組み合わせると、特定の対象負荷がかかったワークロード モデルの実装に必要な残りの詳細を判断できます。

テスト実行中に取得する指標を特定する

指標を適切に特定、捕捉、および報告することで、アプリケーションのパフォーマンスを目標のパフォーマンス特性と比較する方法に関する情報が提供されます。また、指標により、アプリケーション内で問題のある領域やボトルネックを特定できます。

テストの設計中にパフォーマンス受け入れ基準に関連する指標を特定しておくと、テストの設計を実装する際に、こうした指標を収集する手法をテストに統合する場合に有効です。指標を特定する際は、目標とする明確な特性、またはこうした特性に直接的、間接的に関連するインジケータのいずれかを使用します。

考慮事項

テストを設計する際は、次の点を考慮します。

- 現実に即したテスト設計は、人材、アプリケーションと相互作用する他のシステムなど、システムの制御外にある依存関係から影響を受けます。

- 現実に即したテストの設計は、機械的な手順ではなく、実際の操作とデータに基づきます。
- 現実に即したテストの設計は、結果の信頼性を高めるため、パフォーマンス テストの価値が向上します。
- ユーザーによる遅延や思考時間の現実的なシミュレーションは、テストの正確さに大きく影響します。
- ユーザーがなんらかの理由でタスクを放棄する可能性がある場合、テストの設計では、このような放棄について考慮します。
- 一般的なユーザー エラーを忘れずにシナリオに含めます。
- 現実に即したテストには、コンポーネント レベルのパフォーマンス テストが不可欠です。
- 現実に即したテストの設計は、実装のコストや時間が増加する可能性がありますが、ビジネスや関係者にとっての正確性は大幅に向上します。
- 非現実的なテストで得られたパフォーマンス結果から推測を行うと、システムの範囲が広がるにつれて正確さが低下し、多くの場合、間違った意思決定につながる可能性があります。
- 価値を高める可能性のある指標を判断し、このような指標を捕捉する方法を最も適切にテストに統合する手法を判断するプロセスには、開発者と管理者を関与させます。
- ツールがテストの設計に影響する可能性があることに注意します。ツールやテストを実行できることを前提にテストを設計し、前提に誤りがあることが実証されたらテストやツールを状況に合わせて改定することが、ほぼ常に、適切なテストにつながります。テストを実行する際にツールにアクセスできないことを前提に特定のテストを設計しないでください。

アクティビティ 6. テスト環境の構成

規制の条項によっては、この手順がプロジェクトに当てはまらない場合があります。たとえば、特定機関の管理の下、特定のラボでパフォーマンス テストを実施する必要がある場合があります。これが当てはまるプロジェクトでは、この手順の残りを省略しても構いません。省略しない場合は、次の点を考慮します。

負荷生成ツールやアプリケーション監視ツールを期待どおりに簡単に設計できることはめったにありません。分離したネットワーク環境のセットアップ、ハードウェアの入手、IP スプーフィング用の IP アドレスの専用バンクの調整、監視ソフトウェアとサーバー オペレーティング システムとの間のバージョンの互換性の確立など、問題の原因がどこにあるかは別として、常に、問題が生じるものです。

負荷生成ツールはこうした問題が発生する確率を高めるためのものですが、常に、テクノロジや実践方法の進化に遅れをとることは避けられません。ツールの作成者は、すべてのテクノロジに対するサポートを組み込むことはできません。つまり、ベンダは特定のテクノロジが明確になるまで、そのテクノロジに対するサポートの開発に着手することすらできません。

つまり、多くの場合、テスト対象のアプリケーションがシミュレーションを行ったユーザーと現実のユーザーとの間の差異を正当に指示できない方法で、ユーザーを大まかにシミュレーションするという、ある程度現実的と思われるテストを最初に実装することがパフォーマンス テストプロジェクトにかかわる最大の課題になります。このような状況に対して計画を立てるため、すべてがスムーズに機能するために必要な時間が予定よりも大幅に長くなることに驚かないでください。

アクティビティ 7. テスト設計の実装

実行可能なパフォーマンス テストを作成する細かい手順は、ツールに大きく左右されます。ただし、使用するツールとは無関係に、パフォーマンス テストを作成する場合は、通常、テスト スクリプトの 1 つのインスタンスを取り上げ、時間をかけて徐々にインスタンスやスクリプトを追加します。その結果、コンポーネントやシステムに対する負荷が増加します。多くの場合、テスト スクリプトの 1 つのインスタンスは、シミュレーションを行った 1 人のユーザーまたは仮想ユーザーに相当します。

アクティビティ 8. 作業項目の実行

反復処理の完了時、またはビルドの提供時に、そのビルドに関連し、テストを実施するのに妥当な作業項目のうち、最も優先順位の高い作業項目からパフォーマンス テストを開始します。各作業項目の最後に、見つかった点をチームに公開し、そのフェーズで実施する残りの作業項目の優先順位を付け直した後、次に優先順位の高い実行計画に移ります。できる限り、各作業項目の実行を 1 日から 2 日に制限します。このように制限することにより、特定の作業項目の結果が出ないことが判明する場合や、目的の結果を出すために最初のテスト設計を変更する必要がある場合に無駄な時間がなくなります。

一般に、パフォーマンス テストの作業項目を実行する際は、次の点が重要になります。

- すぐに結果を分析して、必要に応じて計画を変更できるようにする。
- チーム内で頻繁に連絡をとり、情報をオープンにする。

- 結果や見つかった重要な点を記録する。
- 後でテストを繰り返すために必要なデータを記録する。
- 数日おきにパフォーマンス テストの優先順位を見直す。
- 必要に応じてテストの計画やアプローチを調整し、必要な場合は変更に対して適切な承認を得る。

アクティビティ 9. 結果のレポートとデータの保管

各作業項目の完了時にデータや仮の結果を共有している場合でも、パフォーマンス テストの次のフェーズを開始する前に、結果を集約し、傾向分析を実施して、関係者向けにレポートを作成し、開発者、アーキテクト、管理者と連携して結果を分析することが重要です。フェーズとフェーズの間は少なくとも 1 日空けておくべきですが、プロジェクトの完了が近づくにつれて、もう少し時間が必要になる場合があります。こうした短期間の分析とレポート作成で、"大きな中断" が生じることがよくあります。数日おきにレポートを作成することでチームに情報を提供することはできますが、概要レポートだけを発行しても全体的なストーリーが伝わることはめったにないことに注意してください。

パフォーマンス テスタの仕事の 1 つは、データに含まれる傾向やパターンを見つけ出すことですが、これは非常に時間のかかる作業です。また、こうした作業により、パターンが実際に存在するかどうか、特定のテストになんらかの傾向があるかどうかを判断するために、1 つ以上のテストを再実行するきっかけになることもあります。チームでは、時間を節約するためにこうした手順を省きがちですが、このような省略は行わないでください。より多くのデータをより迅速に集めても、テストを一時中断して、集めたデータを定期的に見直さなければ、そのデータが意味することがわかったときには遅すぎることになります。

一般に、パフォーマンス テストの作業項目が完了したら、テスト スクリプト、テスト データ、テスト結果、環境の構成、およびアプリケーションのバージョン情報をすべて、今後の参照用に保管しておく必要があります。こうした情報の保管方法は、チームごとに大きく異なる場合があります。チームで保管標準をまだ設けていない場合は、パフォーマンス テストの計画に保管方法を含めるようにしてください。

通常、テストのセットアップや実行での明らかなミスによって無効と考えられるテスト関連データは保管しなくても構いません。プロジェクトに適用するコンプライアンス基準を確認してください。判断に迷う場合は、とりあえずデータを保管し、ミスやエラーを記述したメモを含めてください。

アクティビティ 10. 計画の変更と変更に対する承認の取得

各テスト フェーズの完了時に、パフォーマンス テストの計画を確認することが重要です。完了した作業項目に印を付け、この完了項目によって計画に段階的な影響がある場合はその影響を評価します。たとえば、完了した作業項目によってその後のフェーズの他のケースや例外ケースがなくなるかどうか、または計画されている作業項目のスケジュールをなんらかの理由で調整する必要があるかどうかを検討します。

計画を調整したら、プロセスやコンプライアンスの規制で定められているように、必要に応じて、その調整の承認を得ることを忘れないでください。

アクティビティ 11. アクティビティ 5 に戻る

計画が更新および承認されたら、アクティビティ 5 に戻り、次のビルド提供時、反復時、チェックポイントのリリース時にテストを続行します。しかし、口で言うほど簡単ではありません。ときには、どんなに一生懸命になっても、この時点で実施する価値のあるパフォーマンス テスト タスクが存在しないこともあります。このような状態は、環境のアップグレード、大幅な再設計やリファクタリング、または完了までに時間が必要なその他の作業が原因で生じます。このような状況に陥った場合は、入手可能な情報に基づいて最終的なレポートをできるだけ準備することなど、時間を有効活用してください。

アクティビティ 12. 最終的なレポートの準備

パフォーマンス テストが完了し、アプリケーションが運用環境または独立検証および有効性確認 (IV&V) 向けにリリースされても、最終的なレポートを完成させ、関係者に送信して、承認されるまでは、仕事は終わりではありません。多くの場合、最終レポートは非常に詳細かつ明確に記述することになります。アクティビティ 1 でコンプライアンス基準が明確になっていれば、多少細かく時間のかかるタスクの場合でも、これは比較的簡単です。

まとめ

CMMI のような厳しく規制された監査可能なプロジェクトのパフォーマンス テストでは、細かく計画され、監視される環境でテストを管理する必要があります。このようなパフォーマンス テストは特に困難です。これは、多くの場合、前のアクティビティで検出された欠陥を解決するま

で、次に計画されているアクティビティを実施できないためです。このような環境でパフォーマンス テストを管理するには、プロジェクト計画に作業項目を割り当て、計画に作業項目の詳細を追加することが重要です。

第 III 部

テスト環境の特定

内容 :

- **パフォーマンス テストの効果を高めるためのシステムの評価**

第 8 章 パフォーマンス テストの効果を高めるためのシステムの評価

目的

- システムの機能を効率的かつ効果的に把握する手法について学習する。
- 予想されるユーザー アクティビティを効率的かつ効果的に把握する手法について学習する。
- システムの論理アーキテクチャと物理アーキテクチャを効率的かつ効果的に把握する手法について学習する。

概要

システムの評価はパフォーマンス テスト作業全体を通じて進めていくプロセスですが、テストプロジェクトの初期段階に実施するとより価値が高まります。システム評価の目的は、プロジェクトの特定のニーズを満たすために、プロジェクト全体、システムの機能、予想されるユーザー アクティビティ、システム アーキテクチャ、およびパフォーマンス テストを進めるのに役立つその他の詳細に関する情報を収集することです。こうした情報により、パフォーマンスの目標と要件の収集、ワークロードの特徴付け、パフォーマンス テストの方針と計画の策定、およびプロジェクトとシステムのリスク査定の基盤が提供されます。

テスト対象のシステムを徹底的に把握することが、パフォーマンス テスト作業を成功させるために重要です。後半の段階で収集される測定値の正確さは、この段階で開発および検証するモデルの正確さによって決まります。評価を行うことによって、受け入れ可能なパフォーマンスの決定、ソフトウェア、システム、またはコンポーネントのパフォーマンス要件の指定、テストが開始される前の作業に対するリスクの特定の基盤が提供されます。

本章の使い方

ここでは、パフォーマンス テスト作業に向けてシステムを評価する方法について学習します。また、システム評価に関連するる主要アクティビティについても説明します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「システムを評価するためのアプローチ」では、システム評価に含まれるアクティビティの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- 残りのセクションでは、システム評価の詳細と重要な説明を理解します。

システムを評価するためのアプローチ

システムの評価には、次のアクティビティが含まれますが、これに限定されるわけではありません。

- ユーザーとのインターフェイスとなるシステムの機能を特定する。
- ユーザーが処理を開始しない (バッチ) プロセスや機能を特定する。
- 予想されるユーザー アクティビティを特定する。
- ユーザー アクティビティが予想を超える可能性があることへの理解を深める。
- テスト アーキテクチャと運用アーキテクチャ両方の正確なモデルを開発する。
- 実際のユーザー環境相応のモデルを開発する。
- 同じアーキテクチャを使用する他のプロセスやシステムを特定する。

上記のアクティビティは、次の手順に従って実現できます。

- システム機能やビジネス プロセスを把握する。
- ユーザー アクティビティを把握する。
- 論理アーキテクチャおよび物理アーキテクチャを把握する。

上記の手順について、以下で詳しく説明します。

システム機能とビジネス プロセスの把握

この手順では、パフォーマンスの受け入れ基準を作成できるように、システムの中核となる機能を特定します。その後、ワークロード モデルを査定して、受け入れ基準とシステム機能のコレクションの両方を検証することができます。

パフォーマンス テストでは、テスト対象のシステムの中核機能を特定することは必要不可欠です。これにより、パフォーマンスの受け入れ基準だけでなく、アプリケーションがこうした受け入れ基準を満たすことができるかどうかを査定するためのユーザー コミュニティ モデルについて、最初の決定を下すことができます。

システムのすべての機能を確実に把握するには、まず、関係者と面談し、システムやアプリケーションの全体的な目的を確認します。システムの目的を完全に把握してから、システムの最適なテスト方法を決定しなければなりません。多くの場合、関係者のビジョンから示されるすべての機能がプロジェクトのドキュメントに明記されているとは限りません。このため、ドキュメントの評価に移る前の手始めとして関係者に面談することをお勧めします。

システムの機能を確認するのに有効なリソースを次に示します。

- 関係者との面談
- 契約書
- 類似アプリケーションの使用方法に関する情報
- クライアントの期待値
- 類似アプリケーションの自身の使用経験
- 設計ドキュメント
- 状態遷移図
- 要件とユース ケース
- マーケティング資料
- プロジェクトの計画
- ビジネス サイクル
- 主要ビジネス プロセス

考慮事項

システム機能やビジネス プロセスを把握する際には、次の点を考慮します。

- 関係者と面談し、システムの全体的な目的を確認します。
- 契約書やドキュメントが、関係者のシステムに対する見解から外れている可能性のあることに留意します。システム機能には、ユーザーが開始するプロセス、スケジュール設定された (バッチ) プロセス、(ウイルス スキャンやデータ バックアップなど) システムに直接関係がないにもかかわらずシステムに影響を与えるプロセスなどがあります。
- 多くの場合、面談、ドキュメント、および計画では、暗黙の機能を多数含む、大まかな機能が説明されます。たとえば、「セキュリティが確保されたログインを行えるようにする」とうい表現には、セッションの追跡、紛失したパスワードの取得、新しいユーザーの作成、ユーザー ID、ユーザーの役割、アクセス許可などの機能が暗黙のうちに示されています。

ユーザー アクティビティの把握

この手順では、テスト対象のアプリケーションの主なユーザー アクティビティを特定します。考えられるユーザー タスクやアクティビティすべてのシミュレーションをパフォーマンス テストで行うことは現実的ではなく、実際には不可能なため、シミュレーションを行うべき最も重要なアクティビティを決定する必要があります。ただし、この決定を行う前に、どのようなユーザー アクティビティが考えられるかを確認する必要があります。

まず、競合他社の Web サイト (競合するアプリケーションが Web ベースでない場合はアプリケーション) を評価します。明示されているかどうかにかかわりなく、競合他社で利用できるアクティビティをすべてユーザーが実行できるようにすることが目標であることが、おそらくプロジェクトのある時点で明確になります。ドキュメントに記載されているかどうかに関係なく、このようなアクティビティを事前に把握しておくと、テスト対象のアプリケーションでこのようなアクティビティが出現したときに驚かずに済みます。

システムの機能を確認するのに有効なリソースを次に示します。

- 類似アプリケーションの使用方法に関する情報
- クライアントの期待値
- 類似アプリケーションの自身の使用経験
- 要件とユース ケース
- 関係者との面談
- マーケティング資料
- ヘルプとユーザー ドキュメント
- クライアントの組織図
- ネットワークまたはアプリケーションのセキュリティ マトリクス
- 履歴データ (請求書、Web ログなど)
- 主要ビジネス サイクル (月次計算、年末処理、5 年間の保管など)

ユーザーが実行できると考えられるすべてのアクティビティのリストを収集したら、「このリストに記載されている以外に、ユーザーがこのアプリケーションで実行できるようなアクティビティが他にありますか」という質問を添えて、このリストをチームで回覧します。

考慮事項

システム機能やビジネス プロセスを把握する際には、次の点を考慮します。

- 事実上、競争相手に負けないことがプロジェクトの目標となる可能性があるため、競合他社の Web サイトを評価します。
- 可能性のあるユーザー アクティビティを考えるように求める際は、すべてのカテゴリのユーザーを考慮に入れてください。顧客、管理者、ベンダ、コール センター担当者は、ドキュメントでは簡単に見つからないかもしれない、アプリケーションのさまざまな側面を使用したり、その側面にアクセスできる可能性があります。
- さらに時間を使って、例外やエラーが発生した場合のアクティビティについての情報を集

めます。こうしたアクティビティは、多くの場合、ドキュメントで示されていたり記載されています。

- 重要と思われるアクティビティまたは競合アプリケーションに現れるアクティビティが存在しないことがわかった場合は、できるだけ早く関連するチーム メンバに相談します。これらは、意図せず見落とされている場合があります。

論理アーキテクチャと物理アーキテクチャの把握

この手順では、ハードウェアやソフトウェアの構造と、アプリケーションとの関係を特定します。この情報は、懸念される特定の分野に対処するためのパフォーマンス テストを設計する際、およびパフォーマンスのボトルネックを特定する際に重要になります。

システム アーキテクチャをきちんと理解しておかないと、プロジェクト後半に行われるパフォーマンス テストに悪影響を及ぼしたり、チューニング プロセスに時間がかかったりする可能性があります。パフォーマンス テスタが論理アーキテクチャと物理アーキテクチャを把握するには、通常、運用環境とテスト環境両方の技術関係者、アーキテクト、および管理者と面談します。パフォーマンス テスタが効率的なテスト方針を設計するためには、システムのどのコンポーネントやどの層が相互に通信するかや、その通信方法を把握しておく必要があるため、こうしたことが重要です。また、コードの基本構造や、アプリケーションに影響を及ぼす外部ソフトウェアも把握しておくと役に立ちます。

"アーキテクチャ" という用語はチームによってさまざまな使い方をされるため、ここでの使い方を明確にするために以下のセクションを用意しました。

論理アーキテクチャ

論理アーキテクチャは、ここで使用されているように、ソフトウェアやコードの構造、相互作用、抽象化を表します。そのコードには、オブジェクト、関数、クラスからアプリケーション全体まであらゆるものが含まれます。コード レベルのアーキテクチャをチームから学習することが必要です。学習する場合は、論理アーキテクチャ層の概念も理解することを忘れないでください。

Web ベースのアプリケーションの最も基本的なアーキテクチャは、3 層アーキテクチャとして知られています。このアーキテクチャでは、多くの場合、各層が次のように定義された役割を備えた物理コンピュータに対応します。

- **クライアント層** (ユーザーのコンピュータ) - 要求されたデータを表示します。

- **プレゼンテーション層** (Web サーバー) - すべてのビジネス ロジックを処理してデータをクライアントに提供します。
- **データ ストレージ層** (データベース サーバー) - システムで使用されるデータを、通常はリレーショナル データベースに保持します。

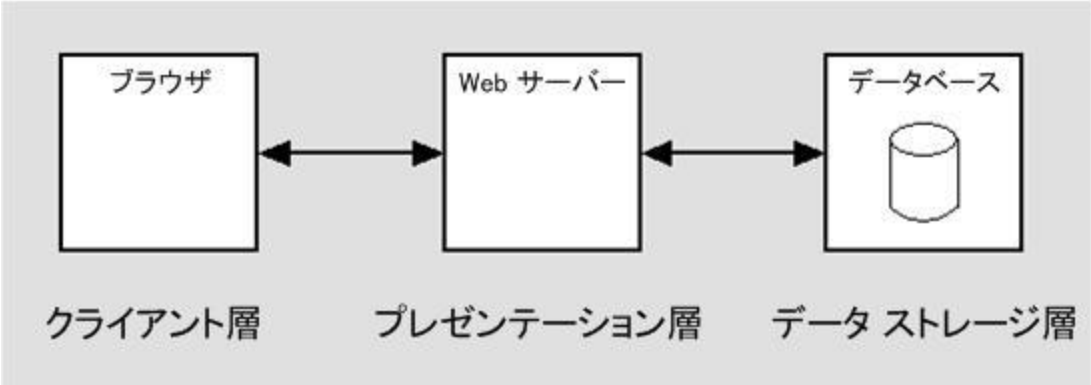


図 8.1 3 層アーキテクチャ

アーキテクチャがより複雑になると、多くの層、同じ役割を果たすコンピュータのクラスタ、複数の論理層のホストとなる 1 台のコンピュータなども含まれる場合があります。

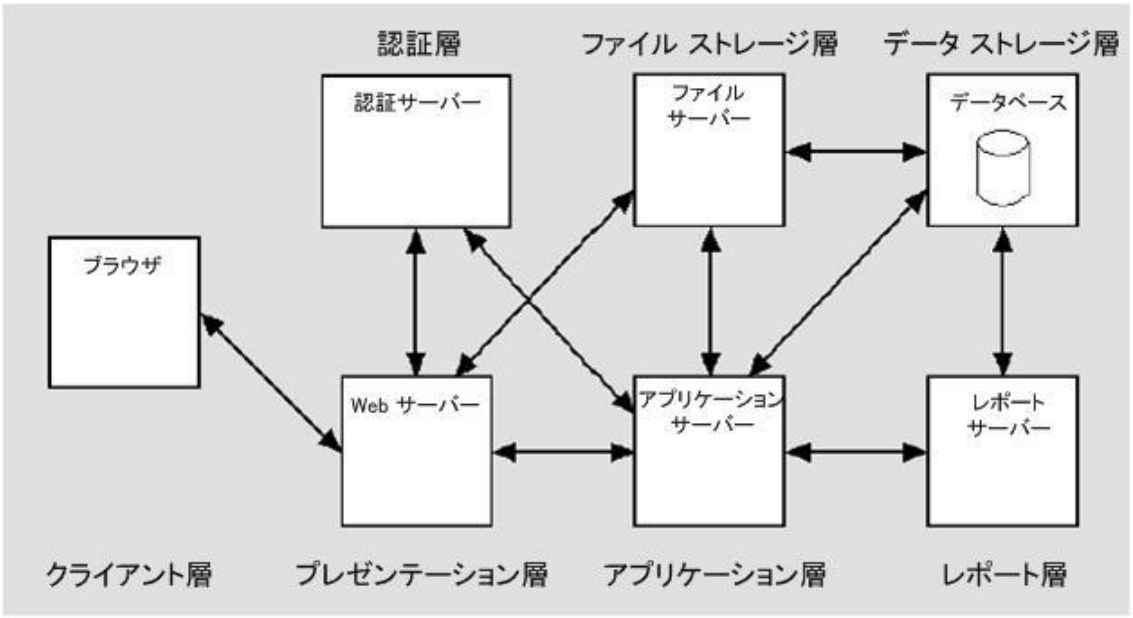


図 8.2 多層アーキテクチャ

具体的には、この複雑なアーキテクチャは次のことを示しています。

- 論理層は、関連する機能のグループと見なして構いません。

- 論理図に示す各層は、複数の物理コンピュータにまたがっている場合、他の 1 つ以上の層と 1 台以上のコンピュータを共有する場合、または専用コンピュータに独占的に関連付けられている場合があります。
- 各論理層を結ぶ矢印は、ネットワーク ケーブルなどの物理接続ではなく、データの流れを表しています。

混乱の原因の 1 つは、実際には "ファイル ストレージ層" といった用語を使用する人がいないことです。"ファイル ストレージ層" は、専用サーバー上に存在するかどうかに関係なく、通常 "ファイル サーバー" と呼ばれます。プレゼンテーション層 (Web サーバー)、アプリケーション層またはビジネス ロジック層 (アプリケーション サーバー)、データ ストレージ層 (データベース サーバー) などにも同じことが当てはまります。

つまり、このようなアーキテクチャでの論理アーキテクチャを理解するための鍵は、他の層とは論理的に区別される一意な機能セットを各層に含めることです。ただし、ある層が通常 "サーバー" と呼ばれていても、各層がそれぞれの専用コンピュータに存在すると考えるのは危険です。

物理アーキテクチャ

環境の物理アーキテクチャ、つまり、ソフトウェアを実行している実際のハードウェアが、少なくとも論理アーキテクチャと同程度に重要であることは明らかです。

多くのチームが実際のハードウェアを "環境" または "ネットワーク アーキテクチャ" と呼んでいますが、実際、どのチームもパフォーマンス テスタが対象とする項目をすべて盛り込んでいるとは限りません。一般に、テスタの関心事項は図で表現されます。このような図では、実在の物理コンピュータが示され、その役割を示すラベルが付いています。また、これらの物理コンピュータが通信する他の実在の物理コンピュータも示されます。このような物理アーキテクチャの例を次の図に示します。

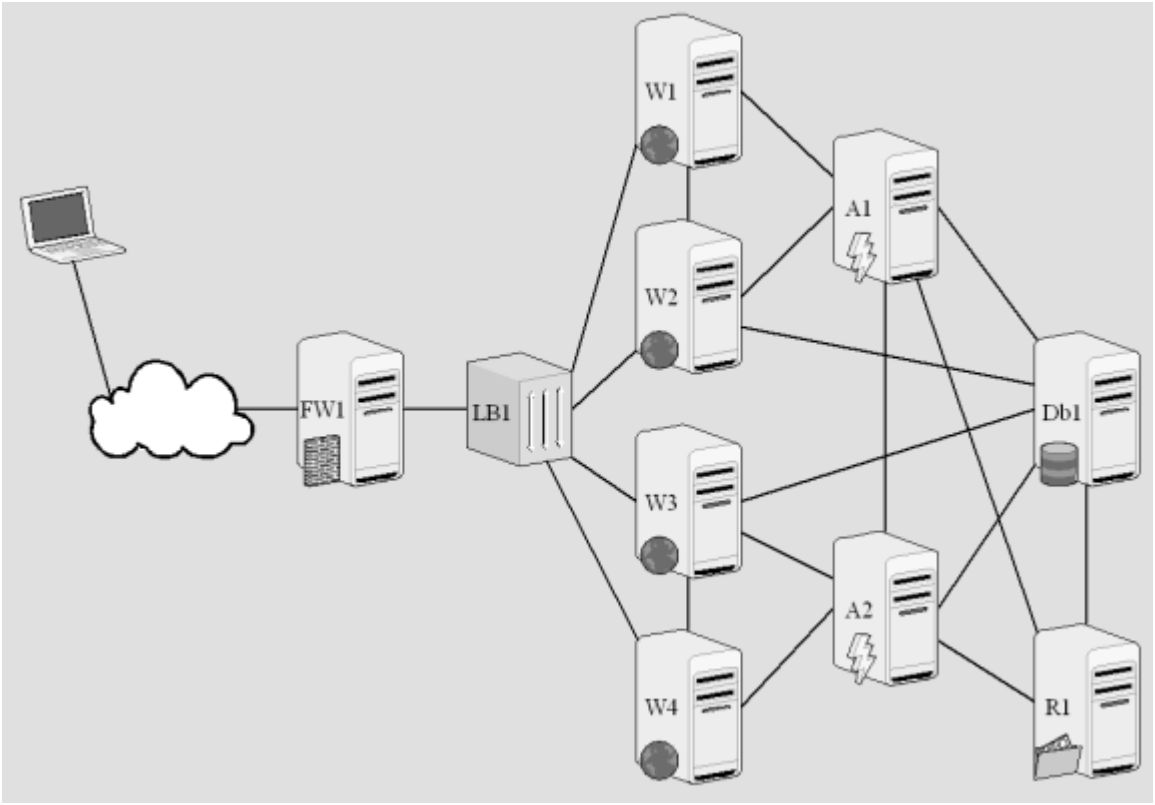


図 8.3 物理アーキテクチャ

システム アーキテクチャ

システム アーキテクチャは、実際には、論理アーキテクチャと物理アーキテクチャをまとめただけです。以下の図は、システム アーキテクチャの例を示しています。明らかに、この例は、アーキテクチャの側面を一部含んでいませんが、この場合は、パフォーマンス テストの対象となるいくつかのポイントを強調するために使用しています。

- 認証層とアプリケーション層の役割は、2 台のサーバーで担当できます。
- マッピング情報により、パフォーマンス テストをより適切に設計できます。
- アプリケーション層などで、パフォーマンス テストを直接対象とすることができます。

このようなパズルのピースが 2 つはまったら、パフォーマンスのテスト作業に最も大きな価値がもたらされます。どの機能やアクティビティがどの層で処理されるかという詳細なコード アーキテクチャとこの情報が手元にあれば、ボトルネックを特定して分離できるテストを設計できます。

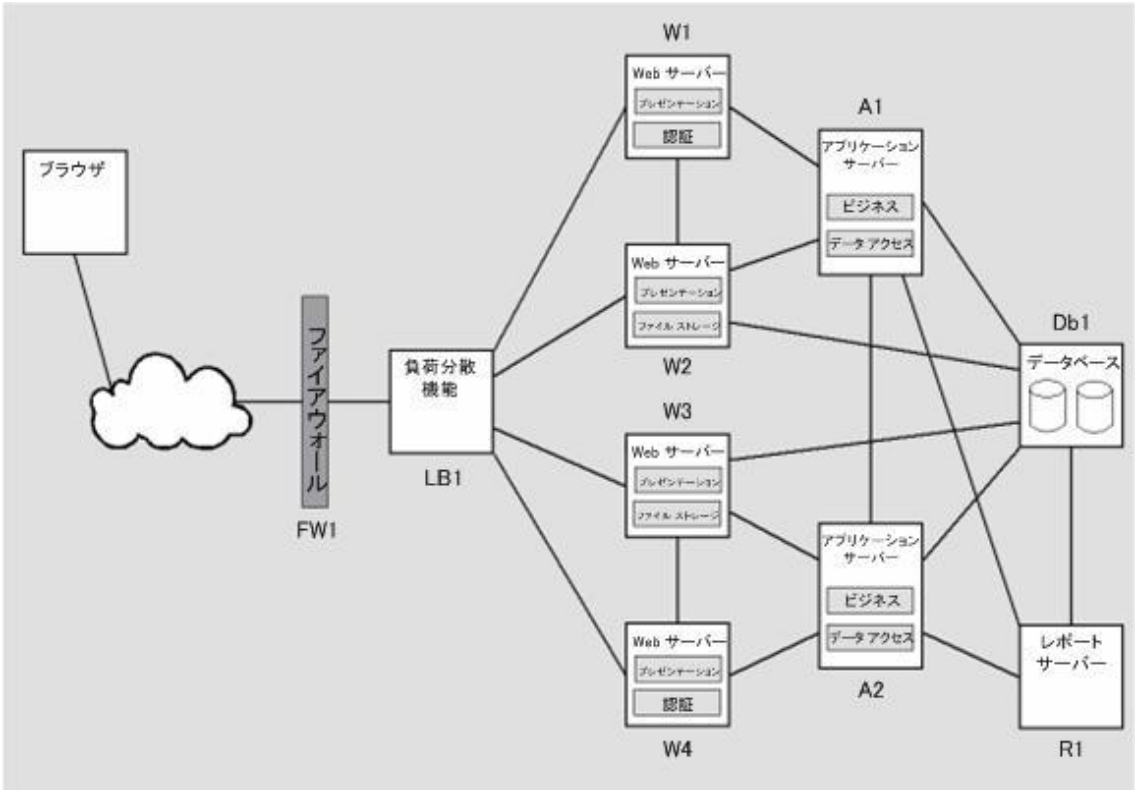


図 8.4 システム アーキテクチャ

考慮事項

システムの論理アーキテクチャと物理アーキテクチャを把握する際は、次の点を考慮します。

- テストを "ブラック ボックス" アクティビティと見なすチームもありますが、パフォーマンス テストを適切に設計し、システム全体の知識 (負荷分散方法からコード オブジェクトのスレッド共有モデルに至るまで) を維持する必要があることをチームに浸透させます。こうすることにより、パフォーマンス テスタが、プロジェクトの初期段階でリスクの高い分野を特定できるようになります。
- サーバーとインターネット プロトコル (IP) アドレスの密接な関係から、Web ファームをテストするには、IP 切り替え手法を使用して運用環境のシミュレーションを正しく行う必要があります。
- アプリケーション サーバーと Web サーバーは、多くの場合、マルチホーム サーバーです。 (つまり、複数のネットワーク インターフェイス カードを備えています)。一方のネットワーク カードは、クライアントまたは Web サーバーに接続され、もう一方は、Web サーバーまたはデータベース バックエンドに接続されます。これは、セキュリティ上の理由によるものです。また、1 枚のネットワーク インターフェイス カードで 2 種類のトラ

フィックに対処するのを避けるためでもありです。このような特性が、パフォーマンス テストの設計、実行、および分析に大きな影響を与える場合があります。

- パフォーマンス テスタが技術要員として開発チームに受け入れられないと、テストの効果が高まりません。パフォーマンス テスタはシステム アーキテクチャを特定することにより、開発者やアーキテクトと同じ技術要員としての立場を確立できます。

まとめ

システムの評価はパフォーマンス テスト作業全体を通じて進めていくプロセスですが、パフォーマンス テスト プロジェクトの初期段階に実施するとより価値が高まります。

システムの評価プロセスでは、プロジェクトの特定のニーズを満たすために、プロジェクト全体、システム プロセスやビジネス プロセスの機能、予想されるユーザー アクティビティ、システム アーキテクチャ、およびパフォーマンス テストを進めるのに役立つその他の詳細に関する情報を収集します。

こうした情報は、パフォーマンスの目標と要件の定義、ワークロードの特徴付け、パフォーマンス テストの方針と計画の策定、およびプロジェクトとシステムのリスクの査定を行う際に役立ちます。

第 IV 部

パフォーマンス
受け入れ基準の特定

内容 :

- **パフォーマンス テストの対象の決定**
- **エンド ユーザー応答時間の目標の定量化**
- **さまざまな種類のパフォーマンス受け入れ基準の確立**

第 9 章 パフォーマンス テストの対象の決定

目的

- パフォーマンス テストの対象を特定および把握する方法について学習する。
- リソース使用量の目標としきい値を把握および推測する方法について学習する。
- リソースの予算 (割り当て) を把握および見積もる方法について学習する。
- プロジェクト全体を通じて利用可能な情報が増えるに従って、さまざまな種類のパフォーマンス テストの対象を見直し、更新する方法、およびチームに更新内容を伝える方法について学習する。

概要

パフォーマンス テスト作業の対象を決定するには、変更点、考えられるリスク、および改善の機会を特定することが重要です。パフォーマンス テストの対象を決定して記録する方法の 1 つは、プロジェクト チームの各メンバに対して、プロジェクトの特定の時点、または特定のマイルストーンを達成した直後にパフォーマンス テストを実施するとどのような価値を得ることができ、どのようなリスクを軽減できるかを質問するだけです。このような対象には、負荷がかかった状態でのリソース使用率に関するデータの提供、アプリケーション サーバーのチューニングを支援する特定の負荷の生成、各 Web ページから要求されるオブジェクト数のレポートの提供などがあります。

プロジェクトのライフ サイクルの初期段階でパフォーマンス テストの対象を収集し始めることが最も大切ですが、こうした対象を定期的に見直し、新しい対象を追加する必要があるかどうかをチーム メンバに問い掛けることも重要です。

パフォーマンス テストの対象を決定する際は、次の大まかな考慮事項に留意します。

- パフォーマンス テストの対象は、パフォーマンスの検証アクティビティと確認アクティビティの出発点となります。
- パフォーマンス テストの対象は、業務量、将来の成長など、ビジネスを開始するときの着眼点に基づいて策定します。こうした情報により、着眼点に対応する技術的な対象を明確に示すことができます。
- パフォーマンス テストの対象はビジネス ニーズと相関関係があるため、実際の顧客が関与する実際のビジネス シナリオを表す必要があります。
- 大まかな対象を決定後、より具体的にテクノロジと対応するように対象を調整できます。

本章の使い方

ここでは、チームに最大の価値がもたらされるよう、チーム内で協力してパフォーマンス テストの対象を確立する方法について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「用語」では、プロジェクトのコンテキスト内で用語を正しく、明確に使用できるように、パフォーマンス テストの対象に関する一般的な用語について理解します。
- 「パフォーマンス テストの対象を決定するためのアプローチ」では、このアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- 残りのセクションでは、パフォーマンス テストの対象の特定と把握、リソース使用量の目標としきい値の把握と推測、およびリソースの予算や割り当ての把握と見積もりをさらに細かく理解します。
- 「ケース スタディ」では、パフォーマンス テストの対象を特定する実際の例を紹介します。

用語

ここでは以下の用語を使用しています。

用語 / 概念	説明
パフォーマンス テストの対象	"パフォーマンス テストの対象" とは、パフォーマンス テストのプロセスから収集されるデータのことです。これらのデータは、製品品質を判断または改善する際に必要な値を持つと予測されます。ただし、こうした対象は定量的である必要はなく、パフォーマンスの要件、目標、または規定したサービスの品質 (QoS) 仕様に直接関連する必要もありません。
パフォーマンス対象	"パフォーマンス対象" は、通常、応答時間、スループット (1 秒あたりのトランザクション数)、およびリソース使用率のレベルに関して指定されます。一般には、ユーザーの満足度に直接つながる可能性のある評価指標に重点が置かれます。

パフォーマンス ターゲット	"パフォーマンス ターゲット" は、特定の条件セットの下でプロジェクトに特定された評価指標の目標値で、通常、応答時間、スループット、リソース使用率のレベルに関連して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク入出力 (I/O)、ネットワーク I/O などがあります。通常、パフォーマンス ターゲットはプロジェクトの目標と一致します。
パフォーマンスのしきい値	"パフォーマンスのしきい値" は、プロジェクトで特定された評価指標の許容最大値で、通常、応答時間、スループット (1 秒あたりのトランザクション数)、およびリソース使用率のレベルに関して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク I/O、ネットワーク I/O などがあります。通常、パフォーマンスのしきい値は要件と一致します。
パフォーマンス予算	"パフォーマンス予算" ("割り当て") は、開発者がコンポーネントに使用できるリソース量に関して課せられる制約です。

パフォーマンス テストの対象を決定するためのアプローチ

パフォーマンス テストの対象の決定は、次のアクティビティの点から考えることができます。

- パフォーマンス テスト作業の対象を決定する。
- リソース使用量の目標としきい値を把握および推測する。
- リソースの予算や割り当てを把握または見積もる。
- 評価指標を特定する。
- 結果を伝える。
- 常に、対象、ターゲット、および予算の変化に注意する。

上記のアクティビティについては、この後詳しく説明します。

パフォーマンス テストの対象を決定する

ここで説明する手法は、パフォーマンス テスト プロジェクトに効果があることが実証されています。このような手法を説明どおり正確に適用するか、特定のプロジェクトや作業環境に合わせて変更するかは重要ではありません。重要なのは、対象について意識的に共同作業できるように留意することです。つまり、対象とは、パフォーマンス テスト作業によって、プロジェクト ライフ サイクルのできる限り早い時期にチーム (特にアーキテクト、開発者、および管理者) に大きな価値がもたらされるようにできる 1 つのツールです。

全体的な対象を決定する

最初の作業として、パフォーマンス テスト作業の全体的な対象を決定します。一般的な対象を次に示します。

- アプリケーションが契約、規制、およびサービス レベル アグリーメント (SLA) に従っているかどうかを判断する。
- チューニングが必要なボトルネックを検出する。
- さまざまな構成オプションに対するパフォーマンス特性の判断について開発チームを支援する。
- スケーラビリティと処理能力の計画作業に関する入力データを提供する。
- アプリケーションを運用環境に展開する準備が整っているかどうか判断する。

プロジェクト計画を確認する

個々のチーム メンバまたは少人数のグループでプロジェクト計画を確認します。プロジェクト計画は必ずしもドキュメントの形式になっているとは限らない点に留意してください。たとえば、ホワイトボード上のスケッチ、一連の電子メール メッセージ、またはさまざまなチーム メンバが思い描いている漠然とした考えかもしれません。ポイントは、プロジェクト計画の形式があまりまとまっていない場合でも、すべてのプロジェクトにはなんらかの基盤となる計画があることです。計画を確認したり聞き取ったりしながら、チェックポイント、反復、またはマイルストーンなどのポイントに到達したら、必ず次のような問い掛けを行います。

- 前回の反復と今回の反復の間で変更されるのは、どのような機能、アーキテクチャ、ハードウェアですか。
- 上記の変更に関するパフォーマンス予算、またはパフォーマンスのしきい値はありますか。あるとすれば、それはどの程度ですか。それはテストできますか。予算内、またはしきい

値内に収まらなかった場合、どのような結果になりますか。

- この変更の結果、チューニングが必要になる可能性がありますか。収集してチューニングに役立てることができる評価指標はありますか。
- この変更は、これまでに評価指標をテストまたは収集した他の領域に影響を与える可能性がありますか。可能性があるとすると、どの領域ですか。すべて想定どおりに機能しているかどうかを判断できるように、どのようなテストを実行したり、どのような評価指標を収集したりすることができますか。
- 上記の変更には、どのようなリスクや重要な懸案事項が関連していますか。変更の効果がない場合は、どのような結果になりますか。

アーキテクチャを確認する

個々のチーム メンバまたは少人数のグループで論理アーキテクチャと物理アーキテクチャの両方を確認します。繰り返しになりますが、この情報はまだドキュメントになっていないかもしれません。しかし、少なくとも概念的なモデルを想定しているメンバがいます。概念的モデルが想定されていない場合は、おそらく、モデルを見つけることが重要です。アーキテクチャを確認したり、聞き取ったりする際は、次のように問い掛けます。

- 以前にこのアーキテクチャ運用または使用したことがありますか。
- プロセスの初期に受け入れ可能なパラメータの下で実行されているかどうかを判断する方法はありますか。いくつかの想定を確認するために使用できる実験やアーキテクチャの検証方法はありますか。
- チューニングが必要になる可能性はありますか。このような決定を行えるように、どのようなテストを実行したり、どのような評価指標を収集したりすることができますか。

チームのメンバに質問する

個々のチーム メンバに、プロジェクトのパフォーマンスに関する最大の懸案事項について、およびできる限り早くこのような問題を検出する方法について質問します。最良の回答を得るには、チーム メンバとの信頼関係を確立する必要があります。各チーム メンバに対してもチーム全体に対しても、質の高い製品の構築にもっと役立てるためにこの情報を求めていると安心させます。

リソース使用量の目標としきい値を把握および推測する

このアクティビティは、誤って使用されることがあります。目標としきい値は、特定のリソースに関連する特定の評価指標である点に留意してください。たとえば、プロセッサの使用率が頻繁

に 80% を超えているとサーバーのパフォーマンスが大きく低下することは、一般的に認められています。これに基づいて、多くのチームはプロセッサの使用率の目標を 70% に、しきい値を 80% に設定します。このように設定することで、プロセッサの使用率が 70% を超えた状態が数秒以上続いたことを確認した場合はチームに通知し、プロセッサの使用率が 80% を超えた状態が数秒以上続いたことを確認した場合は問題を記録することになります。注目すべき点は、このような目標やしきい値を明確にするには、非常に多くの時間が必要になる点です。目標やしきい値が疑わしくなった場合は、それ以上設定を継続しないでください。

非常にまれな状況を除いて、パフォーマンス テスタが目標やしきい値を決定することは適切ではありません。パフォーマンス テスタの役割は、データを取得して、テスト結果を目標やしきい値と比較することだけです。パフォーマンス テスタが目標やしきい値の設定に最も適している場合でも、目標やしきい値の達成を確認する担当者にはしません。代わりに、このような目標やしきい値の達成を確認する担当チーム メンバに情報を提供することを担当し、達成の確認担当者が多くの情報を利用して判断できるようにします。自分で目標を設定しようとする衝動を抑えることが重要です。このアクティビティを実行する際は、次の点を考慮します。

- 運用サポート チームと話し合います。チームが測定する対象と、そのしきい値を決定します。これはこのチームの仕事です。このチームは何年もこの作業を続けており、問題が発生する場所も知っています。
- アーキテクト、または目標としきい値に関する決定事項の適用や決定を担当すると考えられる他のチーム メンバに依頼して、上記の決定を共有します。
- 業界の他の組織が行っていることを調べます。自身が目標やしきい値の設定担当者でない場合でも、Web を検索したり他のドキュメントを参照したりして最新の推奨事項を確認することを常にお勧めします。このような推奨事項がプロジェクトに関連していると思える場合は記録します。この目標としきい値に関するデータは、テスト中に収集する実データに有効なコンテキストを把握できることがあります。
- テクノロジの主要パフォーマンス インジケータ (ネットワーク、ディスク、メモリ、およびプロセッサ) を使って作業します。
- ビジネス要件に対応する主要パフォーマンス インジケータを使って作業します。この作業はエンジニアリングとビジネスを結びつけるのに役立ちます。
- 主要パフォーマンス インジケータとビジネスの評価指標の両方を使って、ビジネスとインフラストラクチャの現在量、および今後の成長のインジケータについて理解を深めます。
- ビジネスの評価指標を使って作業します。多くのパフォーマンスの評価指標には、ビジネスの評価指標と意味的に密接な関係があります。たとえば、1 秒あたりのデータベース トランザクションには 1 秒あたりの注文数が関係し、1 秒あたりの検索数には 1 秒あたり

の Web アクセス数が関係します。

- パフォーマンスの評価指標を明確にして理解する際は、関係者と共同作業を行います。ほとんどの関係者はパフォーマンス テスト、診断、デバッグ、または分析の専門家ではありませんが、関係者の大半はビジネスのパフォーマンス評価指標の要件に関する専門知識を備えています。このような関係者は、システムについての運用関連の評価指標を明確に示すことができます。このため、パフォーマンス評価指標をより直感的な方法で公開するのに役立ちます。

リソースの予算を把握または見積もる

前述のように、パフォーマンス テスタの仕事は予算や割り当てを実施することではなく、予算や割り当てについての情報を収集し、提供することです。リソースの予算や割り当てを決定することは、チームで連携して目標としきい値が現実的であることを確認する方法の 1 つです。たとえば、特定のサーバーでデータベース サーバー ソフトウェアとアプリケーション サーバー ソフトウェアの両方をホストする場合、目標の 1 つがサーバー全体の RAM 使用量が 1 GB を超えないようにすることなら、データベース ソフトウェアに RAM を 600 MB 割り当て、アプリケーション サーバー ソフトウェアに RAM を 400 MB 割り当てることができます。予算内に収めるこのような作業は、各ソフトウェア コンポーネントの開発者と管理者が担当します。パフォーマンス テスタとして予算や割り当ての把握に注意を向ければ、リソースが予算に近づいたり、予算を超えたりしたとき、ほぼすぐにチームに知らせることができるので、チームが対処する時間を最大限に引き出すことができます。このアクティビティを実行する際は、次の実績のある実践方法を考慮します。

- アーキテクト、または目標としきい値に関する決定事項の適用や決定を担当すると考えられる他のチーム メンバに依頼して、上記の決定を共有します。
- プロジェクトのドキュメントを確認します。パフォーマンス テスタは必ずしもデザインやアーキテクチャのドキュメントの確認に明確に参加要請がくるわけではないので、参加を依頼することを忘れないでください。
- 開発者やアーキテクチャの会議に出席します。「メモリの使用量が X の場合にそのオブジェクトを取得できるかどうか確認する」などのコメントを記録します。こうした指示はめったにドキュメントに記載されないため、会議に出席しなければ知ることができません。オブジェクトのメモリ使用量を監視する人数が増えれば開発者の役に立ちます。
- 使用を検討しているテクノロジの状態を示す、主要パフォーマンス インジケータのしきい値を使って作業します。
- 1 秒あたりの注文数、失敗した注文要求数などのビジネス要件を満たしているかどうかを

示す、ビジネス評価指標を使って作業します。

評価指標を特定する

ほとんどの場合、これはあまり意識する必要のないアクティビティです。たとえば、Web サーバーのプロセッサ使用率が 10 秒間のうち 1 秒以上 80% を超えてはならないと目標に規定されている場合、監視が必要な評価指標の 1 つは明らかに、少なくとも 1 秒の間隔をあけてポーリングする Web サーバーのプロセッサ使用率です。すべてのテストで毎回このような測定を行う必要はないかもしれませんが、測定が必要な評価指標であることは疑問の余地がありません。しかし、収集する関連評価指標が明確ではない場合や、単純ではない場合があります。このような場合は、次のアプローチを検討します。

- 目標が達成されているかどうかを示す評価指標と収集する各対象とを対応付けるグリッドや、簡単なスプレッドシートを作成します。
- テストや、同時に収集予定の他のデータに影響を与えずに、各評価指標を収集する方法が明らかでない場合は、調査を行うか、開発チームと連携して最適なアプローチを決定します。
- 開発者、アーキテクト、および管理者と共同作業を行います。こうしたメンバは、特定の目的に役立つ評価指標と、このような評価指標のほとんどを取得する方法を知っています。メンバのアドバイスを受けることで、アプリケーションを、評価指標が最も役立つ状態にする方法を把握していることを確認できます。
- テスト後にこのような情報にアクセスできるように、情報を保存する場所や情報にラベルを付ける方法を検討します。

結果を伝える

パフォーマンスの対象に関するデータを把握するテストの結果を伝えることと、パフォーマンスの全体的な目標や要件に関する結果を伝えることは異なります。対象に関する結果を伝える目的は、リリースに向けてアプリケーションの全体的な適合性を判断することではなく、チームがこの情報を役立てることです。したがって、制限なく情報を共有できると便利です。多くの場合、目標が満たされなかったことはエラー追跡システムに記録する問題ではなく、チームがさらに優れた仕事を行うのに役立つ情報にすぎません。

このアクティビティを実行する際は、次の手法を考慮します。

- 独自の調査だけでなく、ターゲット、予算、および前回の測定値と比較して結果をレポー

トします。どの情報がチームに最も役に立つのかはわかりません。

- チーム全体でレポートを共有します。
- チームが未加工のデータを利用できるようにし、他の方法で解析したり、さらに便利なデータの表示方法を提案したりするようにチーム メンバに要請します。
- 必要に応じてテストの再実行や変更を行う準備を整え、意思を持ち、興味を持って、実際に実行できるようにします。
- 発生する可能性があるすべての結果について責任を取るつもりがあり、責任を取ることができる担当者に指示されたのではない限り、未加工のデータをチーム外部に送信しないでください。
- パフォーマンス低下の考えられる原因はレポートしません。代わりに、現象や状態をレポートします。間違った原因をレポートすると、信用が低下します。

常に対象、ターゲット、および予算の変化に注意する

対象は、プロジェクトの進行中に絶えず変化することに留意してください。要件が変化すると、特定のビルドに機能が追加または削除されたり、ハードウェアの購入が決定したり、コードがリファクタリングされたりします。パフォーマンス テストの対象も絶えず変化します。チームとの緊密な話し合いを維持してください。チーム メンバには変更点と、変更による対象への影響をたずねます。直接話しても、電子メールを使ってもかまいませんが、古くて関連性がなくなった対象をテストすることは、自身の時間の浪費につながることを忘れないでください。

ケース スタディ - パフォーマンス テストの対象の特定

次のケース スタディでは、パフォーマンス テストの対象を特定するアプローチを示します。

ケース スタディ 1

シナリオ

40 年の歴史があり、3,000 人の従業員を抱える金融サービス企業では、新しい運用ハードウェアを含め、エンタープライズ リソース プランニング (ERP) ソフトウェアのアップグレードを実装しようとしています。前回のアップグレードでは期待通りのパフォーマンスが得られず、運用中のチューニングに何か月もかかりました。

パフォーマンスの対象

パフォーマンス テスト作業は、次の全体的なパフォーマンスの対象を基にしました。

- 新しい運用ハードウェアでは前回のリリース以上の処理速度を確保する。
- 新しい運用ハードウェアの構成設定を決定する。
- カスタマイズをチューニングする。

パフォーマンス予算/制約

パフォーマンス テスト作業は、次の予算制限による制約を受けました。

- いずれのサーバーも、予想されるどのような負荷がかかっているときでも、プロセッサ使用率が 80% を超えない (しきい値)。
- 要求されたいずれのレポートによっても、データ キューブ サーバーで 20 MB を超える RAM や 15% を超えるプロセッサ使用率がロックされない。
- 要求されたレポートのいずれの組み合わせでも、データ キューブ サーバーで一度に 100 MB を超える RAM や 50% を超えるプロセッサ使用率がロックされない。

パフォーマンス テストの対象

パフォーマンス テストでは次の最優先対象に重点を置きました。

- 前回リリースよりもパフォーマンスが低下しないことを確認する。
- 応答時間、スループット、およびリソース使用率に関して、アプリケーションの理想的構成を確認する。
- 既存のパフォーマンス不足をデータ キューブ サーバーで解決する。

質問

関連テストの対象を決定するのに、次の質問が役立ちました。

- パフォーマンス テストの実施を決定した理由は何か。
- アップグレードに関して、パフォーマンスの面で最も懸念する問題は何か。
- データ キューブ サーバーについて懸念している理由は何か。

ケース スタディ 2

シナリオ

本社といくつかの支社に 4,000 人のユーザーが分散している金融機関では、融資処理を扱うビジネス アプリケーションにパフォーマンスの問題を抱えています。

企業の IT グループが特定した高いリソース使用率とエラー率に加え、処理速度の低下に関する問題に 6 つの主要業務が影響を受けています。使用率の問題はデータベースのプロセッサ使用率が高いことが原因で、エラーの問題はデータベース クエリで例外が発生することに関連しています。

パフォーマンスの対象

パフォーマンス テスト作業は、次の全体的なパフォーマンスの対象を基にしました。

- システムでは、業務ピーク時にシステムを使用する本社と支社の全ユーザーをサポートする必要がある。
- システムのバックアップを最短時間で実行するという要件を満たす必要がある。
- データベース クエリのプロセッサ使用率は、通常時とピーク時の業務中に 50 ～ 75% を上回らないように最適化する必要がある。

パフォーマンス予算/制約

パフォーマンス テスト作業は、次の予算制限による制約を受けました。

- いずれのサーバーも、本社と支社のユーザーがシステムを使用している間、予想されるどのような負荷 (通常時とピーク時) がかかってもプロセッサ使用率が 75% を超えない (しきい値)。
- システムのバックアップ中の業務の応答時間は 8 秒、またはバックアップが行われていないときの応答時間を超えない。
- 通常時およびピーク時の負荷状態で、全業務の応答時間が 6 秒を超えない。
- ユーザーが送信した融資申し込みが失われる可能性があるデータベースのトランザクション処理中のエラーの発生は許容しない。

パフォーマンス テストの対象

パフォーマンス テストでは次の最優先対象に重点を置きました。

- 融資処理アプリケーションを最適化して、システムが規定したビジネス要件を満たすよう

にする。
- 融資処理アプリケーションの影響を受ける 6 つの業務全体を 100% 対象にしてテストする。
- 不適切なヒントや入れ子になったサブクエリのハッシュを使ってあまり適切ではないと確認されたデータベース クエリを対象にする。
- トランザクション コストを最小限に抑えるため、不要なデータベース クエリを削除できるようにする。
- テストでは、関連コンポーネントの指標 (エンド ユーザー応答時間、エラー率、1 秒あたりのデータベース トランザクション数、データベース サーバーのプロセッサ、メモリ、ネットワーク、ディスクの全状態) を監視する。

質問

関連テストの対象を決定するのに、次の質問が役立ちました。
- パフォーマンス テストの実施を決定した理由は何か。
- プロセッサのボトルネックやトランザクションのエラーの原因になっている可能性があるクエリに関して、パフォーマンス面で最も懸念している問題は何か。
- クエリ関連のどのビジネス ケースによって、プロセッサとトランザクションのエラーが発生するか。
- 業務中のパフォーマンスに影響するデータベース バックアップ操作はどれか。
- バックアップ プロシージャで業務に影響するのはどの時間帯か。その時間帯に関係する最も重要なシナリオは何か。
- 重要な業務を行っている間のユーザー数はどの程度で、ユーザーはどこにいるか (本社、支社)。

このような質問は、パフォーマンス テスタがテスト作業の優先順位を付けるため、最も重要な懸案事項を特定したり、会話やレポートに含める情報の決定にも役立ちました。

ケース スタディ 3

シナリオ

ある Web サイトでは、1 時間に 200 万人のユーザーを対象とするオンライン調査を実施しています。このサイトのインフラストラクチャは、全世界に及ぶ広域ネットワーク (WAN) で構築されています。サイト管理者は、サイトのパフォーマンスをテストして、1 時間に 200 万人の

ユーザーがこのサイトにアクセスできることを確かめたいと考えています。

パフォーマンスの対象

パフォーマンス テスト作業は、次の全体的なパフォーマンスの対象を基にしました。

- Web サイトでは、1 時間に 200 万人のユーザーがアクセスするピーク時の負荷をサポートできる。
- アプリケーション エラーによってアンケートの提出が妨げられない。

パフォーマンス予算/制約

パフォーマンス テスト作業は、次の予算制限による制約を受けました。

- いずれのサーバーも、アンケートの提出中 (ピーク時の負荷は 200 万人) は、予想されるどのような負荷 (通常時とピーク時) がかかってもプロセッサ使用率が 75% を超えない。
- 通常時およびピーク時の負荷状態で、全アンケート提出の応答時間が 8 秒を超えない。
- 提出したアンケートがアプリケーション エラーによって失われない。

パフォーマンス テストの対象

パフォーマンス テストでは次の最優先対象に重点を置きました。

- 1 時間に合計 200 万人の仮想ユーザーを 2 つのデータセンターに分散し、各データセンターに 100 万人のアクティブ ユーザーがいると想定して、1 つのユーザー トランザクションのシミュレーションを行う。
- 1 時間に 200 万人のユーザーがアクセスするピーク時の負荷のシミュレーションを行う。
- 全種類のアンケートを 100% 対象にしてテストする。
- 関連コンポーネントの指標 (エンド ユーザー応答時間、エラー率、1 秒あたりのデータベース トランザクション数、データベース サーバーのプロセッサ、メモリ、ネットワーク、ディスクの全状態) を監視する。
- エラー率をテストして、アンケート システムの信頼性指標を決定する。
- ファイアウォール構成と負荷分散構成を使用してテストする。

質問

関連テストの対象を決定するのに、次の質問が役立ちました。

- パフォーマンス テストの実施を決定した理由は何か。
- 応答時間低下によりデータが失われたり、ユーザーが放棄したりすることにつながる、アンケートの提出に関してパフォーマンス面で最も懸念している問題は何か。

- ビジネス要件に関連して、どのような種類のアンケート提出のシミュレーションを行う必要があるか。
- アンケートを提出するユーザーの地理的所在地はどこか。

まとめ

パフォーマンス テストの対象を決定および記録する場合は、プロジェクトの進行に合わせてチームとコミュニケーションを取り、こうした対象を確立および更新します。チームの各メンバとの時間を調整することは必ずしも簡単ではないかもしれません (特に、プロジェクト チームに幹部関係者、アナリスト、場合によっては代表的なユーザーが含まれる場合は)。しかし、一般的には、重要なパフォーマンス テストの対象を決める際に役立つ情報をすぐに共有できます。このような対象には、ビジネス関連の指標の提供、負荷がかかった状況でのリソース使用率データの提供、アプリケーション サーバーのチューニングを支援する特定の負荷の生成、各 Web ページで要求されるオブジェクト数のレポートの提供などがあります。プロジェクトのライフ サイクルの初期段階でパフォーマンス テストの対象を収集し始めることが最も大切ですが、こうした対象を定期的に見直し、新しい対象を追加する必要があるかどうかをチーム メンバに問い掛けることも重要です。

第 10 章 エンド ユーザー応答時間の目標の定量化

目的

- パフォーマンス要件とパフォーマンス目標の違いを特定する方法について学習する。
- パフォーマンスに関する主観的な要件や目標を把握するため手法をいくつか当てはめる方法について学習する。

概要

エンド ユーザー応答時間の重要な指標は、結局のところ、パフォーマンスの低さに不満を抱くアプリケーション ユーザーの割合しかありません。アプリケーションのユーザーは、パフォーマンス テストの結果値、画面が表示されるときに "長すぎる" と感じる秒数、スループット値などを、認識または意識することはありません。しかし、アプリケーションが遅いかどうかはわかります。こうしたユーザーの印象は、気分的なものから、それまでのアプリケーションの使用経験まで、あらゆるものに基づきます。ここでは、こうしたユーザーの感覚を、テスト可能な数値に変換する方法について説明します。

ユーザーが "満足できる" と考えるパフォーマンスを判断することは簡単ではありません。また、こうしたユーザーの好みは短期間に大きく変わることがあります。ソフトウェア開発会社は時間と費用がかかるという理由で、典型的なユーザー グループを対象とした有用性調査を実施したがりません。こうした会社の大半には、有用性調査を実施したくても、実施するためのリソースもトレーニングもありません。

主要なパフォーマンス テスタからのユーザー エクスペリエンスのレポートは、The Workshop on Performance and Reliability (WOPR、http://www.performance-workshop.org) などのピア ワークショップで共有されています。こうしたレポートでは、アプリケーションのパフォーマンス目標と要件を言葉で表現するだけで、チームはアプリケーションの正常な動作を実現するために、定量化、技術、論理、ロジスティック、管理に関する課題の解決方法を模索できることが示唆されています。同じパフォーマンス テスタからのレポートでは、定量化された目標と要件が、場合によっては満たされ、多くの場合は無視されることが示されています。目標と要件が満たされる場合でも、基準点となる定性的な要件と目標がない限り、目標と要件がユーザー満足度と相互に関係することはほとんどありません。

本章の使い方

ここでは、パフォーマンス テストの目標を確立し、パフォーマンスに関する主観的な要件と目標を把握する手法をいくつか当てはめる方法について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「用語」では、プロジェクトのコンテキスト内で用語を正しく、明確に使用できるように、パフォーマンス テストの目標に関する一般的な用語について理解します。
- 「エンド ユーザー応答時間を定量化するためのアプローチ」では、パフォーマンス テストの目標を決定するアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、エンド ユーザー応答時間の目標を定量化するために最も重要なタスクの詳細を理解します。

用語

ここでは以下の用語を使用しています。

用語 / 概念	説明
パフォーマンス要件	"パフォーマンス要件" は、契約上の義務、サービス レベル アグリーメント (SLA)、ビジネスの固定ニーズなどにより、絶対に調整できない基準です。基準に合格するまでリリースを遅らせる決定に明白にはつながらないパフォーマンス基準は、必ずしも必要ではありません。したがって、要件とはなりません。
パフォーマンス目標	"パフォーマンス目標" とは、製品をリリースする前に満たす必要のある基準です。ただし、このような基準は、特定の環境下では調整可能な場合があります。たとえば、あるトランザクションの応答時間の目標が 3 秒に設定されていて、実際の応答時間が 3.3 秒だった場合、関係者はこのアプリケーションのリリースを選択し、このトランザクションのパフォーマンス チューニングを次回以降のリリースまで延期する可能性があります。

エンド ユーザー応答時間を定量化するためのアプローチ

エンドユーザーの応答時間の定量化は、次のアクティビティの点から考えることができます。

- アプリケーションの機能と使用法を特定する。
- パフォーマンスの要件と目標を言葉で表現して把握する。
- パフォーマンスの要件と目標を定量化する。
- パフォーマンスの要件と目標を記録する。

上記のアクティビティについては、この後詳しく説明します。

アプリケーションの機能と使用法を特定する

アプリケーションの目標パフォーマンス特性を効果的に特定するには、パフォーマンスを際立たせるシナリオを特定する必要があります。パフォーマンスの要件と目標の判断に必要なビジネス シナリオを特定する際、次の 4 つのカテゴリから考えると有効です。

- 頻繁に使用されるシナリオ
- パフォーマンスに大きな影響を与えるシナリオ
- ビジネス クリティカルなシナリオ
- 特別関心のあるシナリオ (場合により、契約上の義務や関係者に起因する)

パフォーマンスの要件や目標の判断に必要なシナリオを特定したら、重役からエンド ユーザーまで、チーム全体がこうした要件や目標の内容を正確に判断するようにできます。一般に必要な作業は、各シナリオまたは一連のシナリオをどのように実行するかをチームに内示するだけです。こうした情報を収集したら、次の作業として、主観的なデータをテスト可能な形式に変換してから、追跡および進行状況の監視のためにそれらのテスト可能な要件や目標を文書にします。

パフォーマンスの要件と目標を言葉で表現して把握する

このアクティビティは、一般に、ソフトウェア開発ライフサイクルの初期段階に実行することが推奨されますが、プロジェクト全体を通じて定期的に見直すことも大事です。このアクティビティをどんなに適切に実行しても、新しい情報が利用可能になるにつれて、契約、感じ方、ビジネスの原動力、優先順位などが変化します。プロジェクト ライフサイクルを通じて、この点に留意してください。たとえば、最終レポートとしてレポートを提示しているときに契約条件が変更されていることに気付いたとすると、そのプロジェクトは最初から契約条件にまったく基づいていなかったかのように見えます。

このアクティビティ全体を通じて、要件と目標を区別することが重要です (上記の「用語」を参照)。要件の特定は、決して難しくありません。要件を判断するには、契約と法的拘束力のある取り決め、または開発中のソフトウェアに関連する標準に注目してください。、また、幹部関係者に、運用環境へのソフトウェア リリースを拒否する原因となる可能性のあるパフォーマンス条件をすべて挙げてもらってください。最終的な基準が、特定のビジネス シナリオやビジネス条件に関連するかどうかはわかりません。しかし、関連する場合は、このようなシナリオや条件を確実にパフォーマンス テストに含める必要があります。

パフォーマンス目標の把握とその後の定量化はさらに困難です。そのため、把握と定量化を個別のアクティビティとして扱うことが重要です。パフォーマンス テスト関連の非常によくあるミスは、最初に目標を主観的または定性的に言葉で表現せずに、定量化を開始することです。

プロジェクト ドキュメントと関連契約書を確認する

このアクティビティの概念は簡単です。規制とコンプライアンスに関するドキュメントは、経営陣以外は簡単には確認できないことが多いため、入手が難しい場合があります。それでも、こうした標準を確認することは重要です。コンプライアンス プロセスの確認には、テストに関連する記述に関する特定の言語や文脈も重要です。たとえば、「取引は」と「概して、取引は」という表現の違いは非常に大きく、前者はすべての取引が毎回準拠することを示しており、後者は、こうした基準の定量化を試みると明らかになるように、まったくあいまいです。

多くの場合、パフォーマンス関連の最も重要な記述は、ビジョンとマーケティングに関するドキュメントにあります。ビジョンに関するドキュメントには、しばしば、「少なくとも以前のリリースと同じくらい迅速に」、「拡大する顧客ベースをサポートできる」、「市場と整合性のあるパフォーマンス」といったパフォーマンス目標が含まれています。しかし、マーケティングに関するドキュメントは、何気なくパフォーマンス要件が含まれていることで有名です。

公表されたマーケティング ステートメント内で行われた宣言には、米国における法的拘束力があります。その結果、パフォーマンス (またはその他) に関して行ったる主張は、調整不能な要件となります。このことはソフトウェア業界全体ではあまり知られておらず、これが原因で、マーケティング資料に「迅速」、「即時」、「市場を牽引するパフォーマンス」などの表現が含まれる場合に重要な課題を引き起こしました。項目ごとに、用語が公的かつ適切に定義およびサポートされる必要があります。ここで、パフォーマンス テストが必要になります。

このアクティビティを完了するために必要な作業は、アプリケーションの処理速度、スケーラビリティ、安定性にほんの少しでも関連する、これらの公開済み資料内の記述を明らかにし、定量化の準備ができるまでそれらを棚上げしておくことだけです。または、後から改訂される可能性があるということを理解したうえで、これらの記述をそのまま要件管理システムに直接記録することもできます。

"リリース" の決定に影響を与える関係者と面談する

関係者は、必ずパフォーマンスに関する意見を持っています。そして多くの場合、こうした意見を既に定量化されて絶対的であるような言葉で表現します。しかし、それが十分に理解されていることはほとんどありません。関係者との面談の鍵は、彼らの意見を把握するだけでなく、そうした意見の背景にある意図を探ることです。

たとえば、電気通信関係出身の関係者が、アプリケーションに "99.999% の可用性" を求めているとすると、その関係者はそれがほぼ不可能な基準であるということを理解していない可能性があります。というのも、その基準では、Web サイトを使用できない期間は 1 年あたり約 5 分 (1 日あたり約 1 秒) です。実際のところ、多くの Web サイトは、ユーザーが気付くこともなく 1 日に 1 時間 ("適切" な時間の場合) は停止している可能性があります。

実際、たとえ 1 秒の遅延が 1 日に 1 回発生したとしても、Web ユーザーがそれに気付くとは思えません。固定電話で会話の途中に毎日 1 秒間無音状態が発生することは、ユーザーには絶対に受け入れられませんが、Web サイトの場合には不必要に厳しい基準となる場合があります。鍵は、パフォーマンスに関する関係者の意見の背景にある真意を探るため、適切な質問をすることです。次に、関係者の意図を把握するのに役立つ、出発点となる質問のサンプルと予測される関連質問を示します。

- 「他の類似アプリケーションや Web サイトと比較して、このアプリケーションはどのように動作する必要がありますか。」どの程度の向上率を想定していますか。10% 程度、それとわかる程度、または劇的ですか。このアプリケーションまたは Web サイトに求めているようなパフォーマンスを示すのは、具体的にどのアプリケーションまたは Web サイトですか。"x" 秒という数値が示されましたが、その数値はどのようにして決めましたか。また、それは何を示していますか。
- 「ダウンタイムによる中断をどの程度受け入れることができますか。」それには、事前にユーザーに通知される定期メンテナンスは含まれますか。ユーザーが単純に Web サイトやアプリケーションにアクセスできなかったり、サイトが停止中であることを知らせるメッ

セージが送信されたりすると、問題になりますか。ダウンタイムの間、ユーザーがタスクを実行できるとしても、処理速度が遅くなる場合にはどうですか。

- 「アプリケーションまたは Web サイトでは、トラフィック量の突然の増加に対してどのように対応する必要がありますか。」サポートされる量を超過する場合、すべてのユーザーに対する動作として、パフォーマンスの大幅な低下と、"システムは一時的に使用できません。後でもう一度やり直してください。" という内容のメッセージの送信では、どちらを選択しますか。アプリケーションまたは Web サイトでは、一貫性があるパフォーマンスと、現在の使用量に基づく平均よりも最大 50% まで高速または低速となる可能性がある可変のパフォーマンスでは、どちらがより適切ですか。

このアクティビティを完了するには、質問と回答を記録し、関係者に明確に説明を求められない限り、回答や回答に対するコメントを定量化しないことが最も重要です。基本的には、定量化を特別に必要としない回答を得られる質問を行い、引き続いて、最初の回答を主観的に修正するのに役立つ質問を行います。関係者が数値を示す場合にはそれらをメモしますが、適切な数値とは見なさないでください。

アプリケーション関連の標準や競争のベースラインがあるかどうか判断する

安全性が重要なデバイスやアプリケーションを除いて、パフォーマンス関連の標準がわずかながら存在します。多くの場合は、市場の期待や競争によって事実上の業界標準が作成されます。あらゆる業界のあらゆるアプリケーションに、競争状況を判断するためのさまざまな手法や情報源があります。つまり、アプリケーションの比較対照の公式標準や業界標準があるかどうかを確認するまで、目標と要件を完全に把握したとは考えないようにしてください。

パフォーマンスの要件と目標を定量化する

要件と目標を把握したら、次にそれらを定量化します。絶対に必要というわけではありませんが、定量化する前に要件と目標を区別しておくと便利です。要件は、目標とは異なり、はるかに慎重かつ完全に定量化する必要があります。たとえば、「要求された Web ページを約 3 秒で表示する」という目標は、パフォーマンス目標としては完璧ですが、パフォーマンス要件としてはまったくテスト不可能です。要件は、「応答時間は 10 秒を超えてはならない」というように、具体的にする必要があります。また、要件には適用先のシステムの条件や状態を指定する必要があります。

要件と目標を区別する

このアクティビティは、一見、純粋な機械的作業のように思われます。把握した項目に法的または契約上の拘束力がある場合や、ソフトウェアのリリース不可を決断できる関係者がその項目を必要とする場合、それは要件です。課題となるのは、要件として特定した項目が、目標として特定した他の項目よりも厳密な場合です。

このような場合、これら競合する項目は関係者に問い合わせ、さらに明確にすることが重要です。場合によっては、目標から要件に変わることがあります。その場合は、目標を項目の一覧から削除するだけです。また、要件が厳密すぎるため変更が必要であると関係者が判断することもあります。とにかく、このような明らかな矛盾を早く解決するほど、後に発生する混乱は少なくなります。

目標と要件間の矛盾は、両方を定量化するまで明らかにならない可能性があることに注意してください。そのため、優先順位が変化していないことを確認するため、定量化後とテスト中 (定期的) に見直すことが重要なアクティビティの 1 つです。

把握したパフォーマンス目標を定量化する

目標によっては、少なくとも概念的には簡単に定量化できるものがあります。たとえば、「以前のリリースよりも遅くない」という目標を定量化するには、最新の運用パフォーマンス監視レポートを参照するか、単一ユーザー、低負荷、高負荷の各テストを以前のリリースで実行し、その結果を記録して比較用のベースラインとして使用します。同様に、「少なくとも競合他社と同程度の速さ」という目標を定量化するには、競合他社のアプリケーションの一連の単一ユーザーのパフォーマンス測定を使用できます。おそらく、アプリケーションに対して一般的なシナリオを 1 週間にわたって 1 時間に 1 回実行するようにパフォーマンス テストのスクリプトをスケジュールすることで実行できます。

多くの場合、定量化する必要がある把握済みの目標の大半は、比較を目標とするものではなく、ユーザーの満足度、つまりサービスの品質 (QoS) を目標とするものです。エンド ユーザーの満足や不満の定量化は困難ですが、決して不可能ではありません。エンド ユーザー満足度の定量化に必要なのは、アプリケーションと数種類の典型的ユーザーだけです。完成したアプリケーションは必要ありません。初回の定量化にはプロトタイプやデモ版で十分です。

たとえば、デモ版やプロトタイプのほんの数行の HTML コードがあれば、各ページ、画面、図、

コントロール、リストの読み込み時間を制御できます。この方法を使用して、応答特性が異なる複数バージョンのアプリケーションを作成できます。その後、各バージョンを試して、そのバージョンの印象を (受け入れられない、遅い、妥当、速いなど、把握した目標に対応するなんらかの表現で) 報告するようにユーザーに求めます。実際の応答時間を把握しているため、その数値とユーザーからレポートされた満足度を対応付けることができます。これは科学的な精密さはありませんが、目標に対しては適切に機能します。特に、このような目標を示すたびに、アプリケーションのパフォーマンス テストに関して同様の質問を繰り返し徹底させていくとさらに適切になります。ユーザーがシステムを操作するときバックグラウンドで応答時間を測定する方法は、機能テスト、ユーザー受け入れテスト、ベータ テストなどに適用できます。その結果、より多くのデータを収集し、アプリケーションの進化に従ってパフォーマンスの目標を強化することができます。

目標を定量化する際に、アプリケーションの使用方法に基づいて目標を区別することを考慮してください。たとえば、1 日あたり 2000 回使用されるデータ入力ページと、4000 万のトランザクションの年 1 回の包括レポートとでは、パフォーマンス目標は大きく異なります。

把握したパフォーマンス要件を定量化する

運が良ければ、把握したパフォーマンス要件の多くは、既に定量化されていてテスト可能です。あまり運が良くない場合は、把握した要件はまったく定量化されていません。その場合は、上記のパフォーマンス目標を定量化するプロセスに従うことができます。もっと運が悪ければ、収集したパフォーマンス要件が部分的にしか定量化されておらずテスト不可能です。

問題は、要件が契約や既存のマーケティングに関するドキュメントから生じている場合にそれを変更できない可能性があることです。「3 秒の平均応答時間」や「2,500 人の同時使用ユーザー」といった要件に直面したら、それらの要件の意味と、それらの要件をテスト可能にするために必要な追加情報を把握する必要があります。

そのための絶対的方法はありません。基本的な考え方としては、記述された要件を共通の言語で正確に解釈し、それらをアプリケーションの最も一般的または予測される状態で補完してから、拡張されたテスト可能な要件について関係者の承認を得ます。関係者は、その後、製品のリリース後に要件への法的準拠が問題になるかどうかを判断する役目を担います。これを例示するため、次の例を考えてみましょう。

要件 : 法的契約からの直接引用「Web サイトの平均応答時間は、3 秒以下でなければならない」。

定量化の拡張 : この要件は特に困難です。文字どおりの (つまり、ほとんどの場合は法的な) 解釈は、「Web サイトの運営中、全応答時間の算術平均は、いずれの時点においても 3 秒を超えない」ということです。その判断も困難ですが、応答時間の意味も定義されていません。応答時間は、「エンド ユーザーが感じる応答時間」、「サーバーの応答時間」、またはまったく別の意味を指す可能性もあります。この要件は次のように体系的に分解できます。

- 矛盾する情報がなければ、3 秒の平均応答時間をテストするのに妥当な方法は、「すべてのページが均等の頻度でアクセスされる」または「最も可能性の高いワークロード分布の下で」のいずれかのみと考えても間違いではないでしょう。
- 繰り返しになりますが、矛盾する情報がなければ、テストの負荷状態が重要です。この場合の最善策は、おそらく、複数の量の平均をとることです。たとえば、低負荷のテストから 30%、予想される負荷のテストから 50%、高負荷のテストから 20% のデータを取得して、その加重平均をレポートします。負荷の分散は、運用環境で予想される負荷プロファイルの適正な近似値になるようにします。または、予想される負荷状態のみでこの要件のテストを実証することも考えられます。

要件 : 販売パンフレットからの直接引用「このアプリケーションは、同時使用ユーザーを最高 2,500 人までサポートします」。

定量化の拡張 : ここでの課題も同じように、「同時使用ユーザー」が Web アプリケーションに関して技術的に正確でないため、いくつかの異なる意味を指す可能性があります。

- 「同時使用」という用語が選択された意図を探ることができる可能性は少ないため、アプリケーションに基づいて最善の判断をする必要があります。一般に、最も無難な解釈は「重なり合うアクティブなセッション」です。「アクティブなセッション」とは、(他の処理を実行するために中断することなく) アプリケーションにアクセスしてからタスクを完了するまでの 1 人のユーザーのアクティビティです。アプリケーションが技術的にセッションを追跡するかどうかは問いません。
- この解釈を使用し、ユーザーの通常のセッション接続時間が 15 分の場合に、2,500 の重なり合うアクティブ セッションのシミュレーションを行うためには、統計的に見ると、ユーザー数が増加と減少を繰り返す現実的なモデルで、30 分間にわたって合計約 5,000 人のユーザーからのアクセスが必要です。
- また、この例では、こうしたユーザーの予想されるアクティビティに関する情報もありません。前の例のように、この要件をテストするのに適した方法は、「すべてのページが均等の頻度でアクセスされる」または「最も可能性の高いワークロード分布の下で」のいずれ

かのみと考えても間違いではないでしょう。ただし、この場合には、「最も可能性の高いワークロード分布の下で」が、本来の執筆者の意図と考えられます。

同時使用ユーザーの定義に関する詳細については、「第 12 章 アプリケーションの使用法のモデル化」を参照してください。

パフォーマンスの要件と目標を記録する

目標と要件を記録するための推奨方法は、常に、チームとツールによって異なります。チームが要件と目標を管理する方法にかかわらず、必ず、定量的なバージョンと定性的なバージョンの目標と要件を同時に記録する必要があります。その結果、プロジェクトの後半になって、リリースできる程度にアプリケーションの動作が適切かどうかを判断するときに、数値だけでなく、数値の背景にある意図を迅速に探ることで、チーム メンバは十分な判断に基づいた判断を下すことができます。

まとめ

応答時間の目標の定量化は、アプリケーションのパフォーマンスに対するユーザーの感覚を表現することに密接に関連しています。ほとんどの場合、アプリケーションのユーザーは、画面へのデータ表示にかかる時間、アプリケーションのスループット、データベースがサポートすべき 1 秒あたりのトランザクション数などを明確に思い描くことはできません。しかし、ユーザーはアプリケーションのパフォーマンス動作はわかります。それは、以前の経験、タスクの最重要項目、期待値を設けた方法など、いくつかの要因から生じる結果としての印象に基づきます。

第 11 章 さまざまな種類のパフォーマンス受け入れ基準の確立

目的

- システム ユーザーの観点、システムのビジネス上の所有者の観点、およびプロジェクト チームの観点だけでなく、コンプライアンス上の想定や技術的考慮事項にも基づいて、パフォーマンス要件とテストの対象を特定し把握する方法について学習する。
- こうした情報を、検証可能で相補的なパフォーマンス受け入れ基準のセット 1 つに集約する方法について学習する。
- プロジェクト全体を通じて利用可能な情報が増えるに従って、パフォーマンス受け入れ基準の見直しと更新を行う方法について学習する。

概要

通常、パフォーマンス要件とテストの対象は、システム ユーザーの観点、システムのビジネス上の所有者の観点、プロジェクト チームの観点、コンプライアンス上の想定、および技術的考慮事項という 5 つの情報源から導き出されます。ここでは、これらの情報源から収集された情報を融合させ、検証可能で相補的なパフォーマンス要件とテストの対象のセット 1 つに集約する方法について説明します。

システムの目標パフォーマンス特性を特定の観点から判断する作業は、そのシステムの全体的なパフォーマンス受け入れ基準を決定する作業の最初の部分にすぎません。目標パフォーマンス特性を限られた観点から確認したら、こうした特性を別の特性と照らし合わせて解決していく必要があります。たとえば、エンド ユーザーはすべてのトランザクションの応答時間が 1 秒以下であることを望むかもしれませんし、ビジネス関係者はシステムが何百万人ものユーザーをサポートすることを望むかもしれません。また、コンプライアンス基準は厳しいセキュリティ ポリシーを要求するかもしれません。

これらの特性はどれも単独では実現可能かもしれませんが、これらすべてを実現しようとすると、時間、テクノロジ、および予算の面からの制約により実現できないこともあります。こうした目標特性を実現する方法を見つけることは、チーム全体の課題です。こうした課題には、テストを通じて矛盾が明らかになってから後追いで対処するのではなく、先手を打って対処する必要があります。

本章の使い方

ここでは、システム ユーザーの観点、システムのビジネス上の所有者の観点、およびプロジェクト チームの観点だけでなく、コンプライアンス上の想定や関連するテクノロジにも基づいて、さまざまな種類のパフォーマンス受け入れ基準を確立する方法を理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「用語」では、プロジェクトのコンテキスト内で用語を正しく、明確に使用できるように、パフォーマンス テストの要件や受け入れ基準に関する一般的な用語について理解します。
- 「受け入れ基準を確立するためのアプローチ」では、パフォーマンス テストの受け入れ基準を決定するアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、システム ユーザーの観点、システムのビジネス上の所有者の観点、およびプロジェクト チームの観点だけでなく、コンプライアンス上の想定や技術的考慮事項にも基づいて、受け入れ基準を確立する方法の詳細を理解します。

用語

以下の用語と定義は、さまざまな種類のパフォーマンス特性を区別するのに役立ちます。

用語 / 概念	説明
パフォーマンス要件	"パフォーマンス要件" は、契約上の義務、サービス レベル アグリーメント (SLA)、ビジネスの固定ニーズなどにより、絶対に調整できない基準です。基準に合格するまでリリースを遅らせる決定に明白にはつながらないパフォーマンス基準は、必ずしも必要ではありません。したがって、要件とはなりません。
パフォーマンス目標	"パフォーマンス目標" とは、製品をリリースする前に満たす必要のある基準です。ただし、このような基準は、特定の環境下では調整可能な場合があります。たとえば、あるトランザクションの応答時間の目標が 3 秒に設定されていて、実際の応答時間が 3.3 秒だった場合、関係者はこのアプリケーションのリリースを選択し、このトランザクションのパフォーマンス チューニングを次回以降のリリースまで延期

	する可能性があります。
パフォーマンスのしきい値	"パフォーマンスのしきい値" は、プロジェクトで特定された評価指標の許容最大値で、通常、応答時間、スループット (1 秒あたりのトランザクション数)、およびリソース使用率のレベルに関して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク I/O、ネットワーク I/O などがあります。通常、パフォーマンスのしきい値は要件と一致します。
パフォーマンス ターゲット	"パフォーマンス ターゲット" は、特定の条件セットの下でプロジェクトに特定された評価指標の目標値で、通常、応答時間、スループット、リソース使用率のレベルに関連して指定されます。リソース使用率のレベルには、アプリケーションが使用するプロセッサの処理能力、メモリ、ディスク I/O、ネットワーク I/O などがあります。通常、パフォーマンス ターゲットはプロジェクトの目標と一致します。
パフォーマンス テストの対象	"パフォーマンス テストの対象" とは、パフォーマンス テスト プロセスから収集されるデータのことです。これらのデータは、製品品質を判断または改善する際に必要な値を持つと予測されます。ただし、こうした対象は定量的である必要はなく、パフォーマンスの要件、目標、または規定したサービスの品質 (QoS) 仕様に直接関連する必要もありません。

受け入れ基準を確立するためのアプローチ

受け入れ基準の確立については、次のアクティビティの観点から考えることができます。

- エンド ユーザーが求める要件を調査する。
- ビジネス要件をまとめる。
- 技術要件を決定する。
- 標準、コンプライアンス、および契約を調査する。
- パフォーマンス テストの対象を決定する。
- パフォーマンス特性を比較および確立する。
- パフォーマンス計画の見直しと更新を行う。

上記のアクティビティについては、この後詳しく説明します。

エンド ユーザーが求める要件を調査する

アプリケーションが運用段階に達したら、問題になる最も重要なパフォーマンス特性は、アプリケーション ユーザーがパフォーマンスの低さに不満を抱くことがあってはならないということです。不満を抱いたユーザーは、そのアプリケーションの代わりを探すことになるでしょう。ユーザーが不満を抱いてしまうと、アプリケーションがどれほど多くのユーザーをサポートし、どれほど大量のデータを処理することができ、また、どれほど効率的にリソースを使用するかは問題になりません。実際、アプリケーションがこれらをすべて実現しており、また、初めて市場に登場する新機能を備えているとしても、不満を抱いているユーザーにとっては何の意味もありません。

アプリケーションのユーザーは、パフォーマンス テストの結果、画面表示にかかる時間を通常 "長すぎる" と感じるしきい値よりも何秒長いかや、どのようなスループットが実現されているかを認識または意識することはありません。主として、アプリケーション ユーザーはアプリケーションが遅いかどうかしかわかりません。また、ユーザーの反応 (反応するかどうかも含め) は、そのときの気分やアプリケーション全般の使用経験など、あらゆるものに基づきます。

エンド ユーザーの満足や不満の定量化は困難な場合がありますが、決して不可能ではありません。エンド ユーザー満足度の定量化に必要なのは、アプリケーションと数種類の典型的ユーザーだけです。完成したアプリケーションは必要ありません。初回には、プロトタイプやデモ版があれば十分です。たとえば、デモ版やプロトタイプのほんの数行の HTML のコードがあれば、各ページ、画面、図、コントロール、リストの読み込み時間を制御できます。この方法を使用して、応答特性が異なる複数バージョンのアプリケーションを作成できます。その後、各バージョンを試して、そのバージョンの印象を (受け入れられない、遅い、妥当、速いなど、把握した目標に対応するなんらかの表現で) 報告するようにユーザーに求めることができます。

実際の応答時間を把握しているため、その数値とユーザーからレポートされた満足度を対応付けることができます。これは科学的な精密さはありませんが、目標に対しては適切に機能します。特に、アプリケーションをテストするたびに、パフォーマンス テストに関して同様の質問を繰り返し徹底させていくとさらに適切になります。テスタがシステムを操作するときバックグラウンドで応答時間を測定する方法は、機能テスト、ユーザー受け入れテスト、ベータ テストなど、あらゆるテストに適用できます。その結果、より多くのデータを収集し、アプリケーションの進化に従ってエンド ユーザーに関するパフォーマンスの目標を強化することができます。

ビジネス要件をまとめる

エンジニアリング上の現実の課題は、単にビジネス要件を満たすだけでなく、期日までに、かつ予算内でこうした要件を実現することです。このバランスを知るには、まず、こうしたビジネス要件を特定する必要があります。たとえば、新しいハードウェアにかける予算や、使用できる既存のハードウェアを特定する必要があります。目標の納期がどれほど融通の利かないものであるか、初回リリースするユーザーの数を限定できるかどうか、プロジェクトのどの側面が優先されるかなどを知る必要があります。たとえば、重要な機能を取り除けば、期日までに、かつ予算内で目標のパフォーマンスを実現できるという状況が生じた場合、このトレードオフは検討する価値があるかどうかを自問する必要があります。

パフォーマンス計画 (パフォーマンス テストの計画ではなく、目標パフォーマンス特性を実現するための計画全体) の作成は、目標のエクスペリエンスを実現するために技術的および金銭的なパフォーマンス予算をどのように使用するかを明らかにすることや、金銭、リソース、およびスケジュールの面からそのパフォーマンスの全体的なコストを特定することを目的として、ほとんどのチームが使用するプロセスです。パフォーマンス計画で考慮する価値の高い領域には、以下のようなものがあります。

- スループットと待機時間 (アプリケーションの展開が、同じネットワークを使用する他のアプリケーションに悪影響を及ぼさないことを確認する必要があるかどうかなど)
- 応答性 (特定の Web サーバーが 200 ミリ秒以上応答しない場合にそのサーバーをスキップする負荷分散機能など、タイムリーに相互作用する必要があるコンポーネントがアプリケーションに存在するかどうかなど)
- 処理能力 (標準状態で最大 500 人のユーザーをサポートするインフラストラクチャを実現できるかどうかなど)
- 導入コスト (エンド ユーザーが求める要件を既存のハードウェアで実現できるかどうかなど)

内部の要件だけでなく、外部のビジネス要件からも影響を受ける可能性があります。外部ビジネス要件には、競合するアプリケーションに "勝るとも劣らない" パフォーマンス特性の実現などがあります。あらゆる業界のあらゆるアプリケーションに、競争状況を判断するためのさまざまな方法と情報源があります。競合他社のアプリケーションに対して負荷テストを実行することができなくても、応答時間についての有益な情報を収集することはできます。これはたとえば、定期的にサイトを閲覧したり、www.keynote.com (英語) などのサイトを調査したりすることによ

って可能です。

要するに、公式に発表された標準が存在しないというだけの理由で、開発したサイトが他のサイトと比較されないとは思い込まないでください。事実上の業界標準は、公式の評価指標よりも、ユーザーの期待値を決める可能性が高いのです。

技術要件を決定する

通常、技術上のパフォーマンス特性は、他のカテゴリに属する要件とは間接的にしか関係しません。このような特性は、多くの場合、特定のテクノロジの使用に関係します。また、通常、ターゲットとしきい値 (特定のリソースに関連する特定の評価指標) で構成されます。

たとえば、サーバーのプロセッサ使用率が 80% に達すると、ユーザーの要求が順番待ちの長い列をなすようになり、そのため、サーバーは依然として問題なく実行されているとしてもユーザーはサービスの低下を経験することが一般に認められています。これに基づいて、多くのチームはプロセッサの使用率の目標を 70% に、しきい値を 80% に設定します。このように設定することで、対象のワークロード状態でプロセッサ使用率が 70% を超えた状態が数秒以上続いたことを確認した場合はチームに通知し、80% を超えた状態が数秒以上続いたことを確認した場合は問題を記録するように、パフォーマンス テスタに指示しています。

パフォーマンス テストの技術要件を把握および管理するには、通常、次の作業が必要です。

- **パフォーマンス テストの対象を決定する。**このアクティビティでは、パフォーマンスに関するプロジェクトの懸案事項に適した形で技術要件に優先順位を付けます。
- **リソース使用量の目標としきい値を把握および推測する。**これらは、おそらく、アーキテクト、管理者、設計ドキュメント、またはハードウェアやソフトウェアの技術ドキュメントから情報を得ることができます。
- **リソースの予算や割り当てを把握または見積もる。**これらは通常、アーキテクトや管理者から開発者に割り当てられるもので、システム全体がターゲットやしきい値から逸脱しないように、特定のコンポーネントが使用する可能性がある特定リソースの量を表します。
- **評価指標を特定する。**多くの場合、リソース使用量を測定する方法は複数存在します。チームでは、コミュニケーションや理解が可能になるように、技術要件ごとに使用する方法と評価指標を決める必要があります。
- **結果を伝える。**収集した技術要件関連のデータは、入手直後からさまざまな形でさまざまなチーム メンバの役に立ちます。データを迅速に共有することで、要件の見直しが必要になる場合に、それを設計、チューニング、および特定することができます。

- **対象、ターゲット、および予算の変化に迅速に対応する。**他の種類の要件と同様、技術要件はプロジェクトを進めていく中で必ず変化します。こうした変化を想定し、迅速に対応します。

標準、コンプライアンス、および契約を調査する

多くの場合、このカテゴリに属するパフォーマンス特性は、エンド ユーザーの満足度、予算、スケジュール、またはテクノロジに大きな影響を与えるかどうかにかかわらず、最も調整しにくいものです。このカテゴリに属するパフォーマンス要件の決定には、通常、次の作業が伴います。

- 法的強制力があるパフォーマンス要件を記述したドキュメントや標準を入手する。
- こうした法的強制力があるパフォーマンス要件を、プロジェクトの枠内で解釈する。
- プロジェクトに適用される、事実上の業界標準や競合比較が存在するかどうか判断する。存在する場合、関連データを収集します。
- 解釈について、関係者の承認を得る。アプリケーションが要件を満たしていなと、企業、さらには個々の関係者でさえ法的責任を問われることがあるので、解釈については関係者の承認を得ることを強くお勧めします。

パフォーマンス テストの対象を決定する

パフォーマンス テストの対象を決定するのは実に簡単です。問題は、パフォーマンス テスタは明示的または暗黙的な対象に必ずしも簡単にアクセスできるとは限らないため、多くの場合、こうした対象を体系的に調査する必要がある点です。パフォーマンス テスト対象の決定には、通常、次の作業が伴います。

- 以下のような、パフォーマンス テスト作業の全体的な対象を決定する。
    - チューニングが必要なボトルネックを検出する。
    - さまざまな構成オプションに対するパフォーマンス特性の判断について開発チームを支援する。
    - スケーラビリティと処理能力の計画作業に関する入力データを提供する。
- 個々のチーム メンバまたは少人数のグループでプロジェクト計画を見直す。その際、次のような問い掛けを行います。
    - 前回の反復と今回の反復の間で変更されるのは、どのような機能、アーキテクチャ、ハードウェアですか。
    - この変更の結果、チューニングが必要になる可能性がありますか。収集してチューニングに役立てることができる評価指標はありますか。

    - この変更は、これまでに評価指標をテストまたは収集した他の領域に影響を与える可能性がありますか。
- 個々のチーム メンバまたは少人数のグループで物理アーキテクチャと論理アーキテクチャの両方を見直す。アーキテクチャを見直す際、次のような問い掛けを行います。
    - 以前にこのアーキテクチャ運用または使用したことがありますか。
    - プロセスの初期に受け入れ可能なパラメータの下で実行されているかどうかを判断する方法はありますか。
    - チューニングが必要になる可能性はありますか。このような決定を行えるように、どのようなテストを実行したり、どのような評価指標を収集したりすることができますか。
- プロジェクトのパフォーマンスに関する最大の懸案事項について、およびできる限り早くこのような問題を検出する方法について、個々のチーム メンバに質問します。

プロジェクトのライフ サイクルの初期段階でパフォーマンス テストの対象を収集し始めることが最も大切ですが、こうした対象を定期的に見直し、新しい対象を追加する必要があるかどうかをチーム メンバに問い掛けることも重要です。

パフォーマンス特性を比較および確立する

ここまで説明した各カテゴリに属するパフォーマンス特性を特定したら、こうした特性を比較し確立することが重要です。多くの場合、これを行う最良の方法は、パフォーマンス計画の策定や強化を行う際に複数の機能チームを利用することです。考慮する主なトピック、アクティビティ、およびトレードオフには、以下のようなものがあります。

- エンド ユーザーが求める要件やコンプライアンス要件を実現するために必要な技術的特性を判断する。
- エンド ユーザーが求める要件やコンプライアンス要件を実現するために必要な技術特性を、既にまとめた技術要件と比較する。プロジェクトに影響を与える重要な違いを書き留めます。
- 改訂した技術要件の実現にかかる (スケジュール、リソース、および金銭面での) コストを見積もります。
- パフォーマンス テストを実施する全体的な対象を見直し、こうしたテストの目標が技術要件やビジネス ニーズをサポートしているかどうかを判断する。

ここで重要なのは、実現しようとしているエクスペリエンスに注目し、そのエクスペリエンスの

実現にかかる関連コストを判断することです。多くの場合、これは、目標のエンド ユーザー パフォーマンス特性のリバース エンジニアリングを行って実用的な技術要件を特定し、その後、こうした技術要件から時間や金銭などの関連コストを見積もることにつながります。

パフォーマンス計画の見直しと更新を行う

ソフトウェア開発の多くの側面と同様に、パフォーマンス計画は変化していきます。前述の 6 つのアクティビティを開発ライフ サイクルの初期段階達成すると暫定的なアプリケーション設計となり、これを技術要件やビジネス要件に照らして評価することができます。要件に照らして評価した際にその設計が意味をなしている限り、その設計を引き続き使用して、プロジェクトを進める過程で段階的に肉付けしていきます。要件に照らして評価した際に設計が意味をなさなくなったら、意味をなすようになるまでその設計を再修正します。

おそらく、開発サイクル全体を通じて何度もこのプロセスを繰り返すことになり、そしてその都度、アプリケーション設計、パフォーマンス計画、およびパフォーマンス要件を更新する可能性があります。一度にすべての肉付けを行おうとはしないでください。また、完璧を求めないでください。重要なのは、設計、計画、および要件がかみ合っている状態を保ち、これらの足並みを揃えておいて、アプリケーションを運用したときにエンド ユーザーや関係者を満足させることです。

まとめ

通常、パフォーマンス要件とテストの対象は、システム ユーザーの観点、ビジネス上の所有者の観点、プロジェクト チームの観点、コンプライアンス上の想定、および技術的考慮事項から導き出されます。

受け入れ基準の完全性に確信を持つには、こうした基準がこれらすべての異なる観点から収集された情報に基づいている必要があります。これにより、検証可能で相補的な、パフォーマンス要件とテストの対象のセットが 1 つ導き出されます。アプリケーション ユーザーがパフォーマンスの低さに不満を抱くことがあってはならないということが最も重要なパフォーマンス特性であると、常に留意しておくことが重要です。

第 V 部

テストの計画と設計

内容 :

- **アプリケーションの使用法のモデル化**
- **各ユーザーのデータとばらつきの判断**

第 12 章 アプリケーションの使用法のモデル化

目的

- 同時使用ユーザーとユーザー セッションの違い、および Web 負荷テストの入力を定義する際にこの違いが重要である理由について学習する。
- 個別の使用シナリオを特定する方法について学習する。
- ワークロードを現実に即して特徴付けする際に役立つ指標について学習する。
- 個別の使用シナリオとその変化をユーザー グループに組み込む方法について学習する。
- 複数のユーザー グループを 1 つのモデルに組み込む場合に特別な考慮事項を特定し、モデル化する方法について学習する。
- アプリケーションを運用環境にリリースする前に入手できるデータ (期待値、ドキュメント、観察結果、ログ ファイルなど) に基づいて Web アプリケーション用の現実的なワークロード モデルを構築する方法について学習する。

概要

Web 負荷テストの最も一般的な目的は、できる限り現実に即してユーザー エクスペリエンスのシミュレーションを行うことです。パフォーマンス テストでアプリケーションの運用時のパフォーマンス特性の把握に直接つながる結果をもたらすには、テスト対象のワークロードが実際の運用シナリオを表す必要があります。現実をかなり正確に表すには、アプリケーションが使用されるビジネス コンテキスト、さまざまな状況での予想されるトランザクション量、予想されるユーザー パスの量など、使用法に関する要素を理解する必要があります。ここでは、ユーザー グループ、およびユーザー グループがどのようにアプリケーションと相互作用するかに重点を置くことで、さまざまなデータ ソースに基づく運用環境での使用法を模したワークロード モデルを作成するためのアプローチについて説明します。

パフォーマンスを正確に予測できるような方法で Web サイトをテストすることは、多くの場合、科学というよりもむしろ芸術の領域に近いものです。パフォーマンスを正確に予測できる負荷モデルや使用モデルを作成するうえで非常に重要なものであるにもかかわらず、このようなモデルを作成するために必要なデータは、通常、テストを実施するユーザーが直接入手できるものではありません。入手できたとしても、通常、不完全であったり、包括的でなかったりします。

確かに、パフォーマンス テストを実施する場合に非現実的なワークロード モデルのシミュレーションを行うことで、有益な情報がチームにもたらされるのも事実です。しかし、運用環境での

パフォーマンスについて正確に予測したり、パフォーマンス最適化の優先順位を決めたりすることができるのは、現実に即したワークロード モデルのシミュレーションを行った場合だけです。

本章の使い方

ここでは、ワークロードの特徴付けをモデル化する方法について理解します。ワークロードの特徴付けをパフォーマンス テストで使用すると、運用環境での特性のシミュレーションを行うことができます。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「アプリケーションの使用法をモデル化するためのアプローチ」では、ワークロードの特徴付けをモデル化するためのアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- さまざまなアクティビティのセクションでは、アクティビティの詳細を理解します。また、これらのセクションには、ワークロードのモデル化に関連するユーザー操作の概念についての重要な説明が含まれています。

アプリケーションの使用法をモデル化するためのアプローチ

パフォーマンス テストで使用するために複合アプリケーションの使用プロファイルを 1 つ以上特定するプロセスを、ワークロードのモデル化と呼びます。ワークロードのモデル化を実現する方法はいくつもありますが、運用環境でのパフォーマンス特性を予測することに成功したほぼすべてのパフォーマンス テスト プロジェクトでは、多かれ少なかれ、以下のアクティビティが明示的または暗黙的に実行されています。

- 対象を特定する。
- 使用法の主なシナリオを特定する。
- 主なシナリオのナビゲーション パスを判断する。
- 各ユーザーのデータと変化を決定する。
- シナリオの相対分布を決定する。
- 対象の負荷レベルを特定する。
- モデルを実装する準備を整える。

上記のアクティビティについては、この後詳しく説明します。

対象を特定する

ワークロード モデルの作成対象の中心となるのは、通常、テストを現実に即したものにするこ

とや、特定の要件、目標、またはパフォーマンス テスト対象に対応したテストを設計することです (詳細については、「第 9 章 パフォーマンス テストの対象の決定」と「第 10 章 エンドユーザー応答時間の目標の定量化」を参照してください)。対象を特定する際は、明示されたビジネス要件を満たすターゲットを使って作業します。対象を考える際は、次の重要な疑問点を検討します。

- 一定時間における、現在の業務量や予測される業務量はどの程度か。たとえば、一定時間内に通常行われる発注数はどの程度でしょう。また、発注をサポートする他のアクティビティ (検索、参照、ログ記録など) の数についてはどうでしょうか。
- 業務量は時間の経過と共にどのように増加すると予測されるか。これを予測する際は、事業の成長、起こりうる合併、新製品の導入など、将来のニーズを考慮に入れる必要があります。
- 現在の、または予測されるピーク時の負荷レベルはどの程度か。この予測には、販売やその他の重要なビジネス プロセス (マーケティング キャンペーン、新たに出荷される製品、外部市場に依存する証券取引などの時間的制約のある活動など) をサポートする活動を反映します。
- どの程度の早さでピーク時の負荷レベルに達すると予測されるか。これを予測する際は、異例の景気急上昇を考慮に入れます (このような事態が発生した場合に、組織はどれほど迅速に需要の増加に適応できるでしょうか)。
- ピーク時の負荷レベルが持続する期間はどの程度か。つまり、リソースの枯渇によりサービス レベル アグリーメント (SLA) に支障を来たすようになるまで、新たな需要にどの程度耐える必要があるかということです。たとえば、為替市場では経済に関する発表が原因で、ほんの数時間の作業ではなく、2 ～ 3 日を要する作業が必要となることがあります。

こうした情報は、Web サーバーのログ、ビジネス要件を反映したマーケティング ドキュメント、または関係者から収集できます。このようなプロセスで特定される対象を、以下にいくつか示します。

- 1 つ以上のモデルが、予測されるピーク時の最大負荷 (1 時間あたりに X 個の注文が処理される) を表すようにする。
- 1 つ以上のモデルが、"3 か月に 1 度の大売り出し" 期間の使用パターンと "標準的な営業日" の使用パターンとの違いを表すようにする。
- 1 つ以上のモデルが、最大 1 年先のビジネス予測とマーケティング予測を表すようにする。

こうした対象は、現時点でのプロジェクトにとってしか意味をなさなくてもかまいません。残りのアクティビティが、目標の達成に必要な詳細を補うのに役立ちます。

考慮事項

対象を特定する際は、次の点を考慮します。

- ワークロード モデルの作成プロセス全体を通じて、自分が考えた仮定と原案をチーム メンバと共有し、フィードバックを求めることを忘れないでください。
- 完璧を求めすぎないようにしてください。また、単純化しすぎるという落とし穴にも陥らないようにしてください。一般に、テスト可能なモデルができたらテストの実行を開始し、結果を収集しながら段階的にモデルを拡張することをお勧めします。

使用法の主なシナリオを特定する

考えられるユーザー タスクやユーザー アクティビティすべてのシミュレーションをパフォーマンス テストで実行するのは、絶対に不可能ではないにしても、非現実的です。そのため、どのような方法を使用して主なシナリオを特定する場合でも、おそらく、パフォーマンス テストの対象として特定するアクティビティや主要シナリオの数を、経験則に基づいて制限することになります。制限するうえで、以下に示す経験則が役に立つでしょう。

- 契約上の義務がある使用シナリオを含める。
- パフォーマンス テストの目標や対象から暗黙に示される、または必須とされる使用シナリオを含める。
- 最も一般的な使用シナリオを含める。
- ビジネス クリティカルな使用シナリオを含める。
- パフォーマンスに大きな影響を与える使用シナリオを含める。
- 技術的懸案事項のある使用シナリオを含める。
- 関係者からの懸案事項のある使用シナリオを含める。
- 注目度の高い使用シナリオを含める。

上記のカテゴリに当てはまる使用シナリオを特定する際に、以下の情報源が役に立ちます。

- 要件とユース ケース
- 契約書
- マーケティング資料
- 関係者との面談
- 類似アプリケーションの使用方法に関する情報
- ベータ版のテスタやプロトタイプのユーザーを観察したりこのようなテスタやユーザーに質問したりすること

- 類似アプリケーションの使用方法に関する独自の経験

アプリケーションの現在の実装 (それが、運用環境での、以前のリリース、代表的なプロトタイプ、ベータ リリースのいずれの実装であれ) の Web サーバー ログにアクセスできる場合は、こうしたログのデータを使用して、上記のリソースから収集したデータの検証や強化を行うことができます。

自分が考える主な使用シナリオの一覧をまとめたら、チーム メンバにコメントを求めます。足りないと思うもの、優先順位を下げてもよいと思うもの、およびその理由 (これが最も重要です) をたずねてください。あるメンバにとっては重要でないように思えるものでも、パフォーマンス テストに含めることが不可欠な場合があります。これは、アクティビティがシステム全体に悪影響をもたらす可能性があるのに、そのアクティビティは重要でないと言ったメンバはこうした影響に気付いていない可能性があるためです。

考慮事項

主な使用シナリオを特定する際は、次の点を考慮します。

- かなり多くの新機能を備えた Web サイトをテストする際は必ず、面談を行ってください。新機能の販売 (売り込み) 担当者と面談すると、どのような機能が求められているかがわかります。つまり、使用される可能性が最も高い機能がわかります。既存のユーザーと面談すると、新機能のうちどれを使用する可能性が最も高いと考えているかを判断することができます。
- 運用前の Web サイトをテストする場合、最適な方法は、ユーザーを代表するグループ (予想されるユーザー ベースの約 10 ～ 20% の規模) に (安定した) ベータ版を展開し、こうしたユーザーが使用したサイトが記録されたログ ファイルを分析することです。
- 従業員、顧客、クライアント、友人、家族などに依頼して社内で簡単な実験を行い、ユーザーの自然なパスや、新しいユーザーとリピート ユーザーとのページ表示時間の違いなどを確認します。この方法は、運用されていない Web サイトのデータを収集する非常に効率的な方法であり、他の方法を使用して収集されたデータの検証方法にもなります。
- さまざまな種類のユーザー、さまざまな役割のユーザー、さまざまな社会的立場のユーザーに、使用法について質問することを忘れないでください。チーム メンバは、明確に質問されない限り、あまり一般的でない種類や役割のユーザーの使用法について伝え忘れることがよくあります。
- 人間のエンド ユーザーだけでなく、システム ユーザーやバッチ プロセスについても考え

てください。たとえば、ユーザーがサイトで操作を行っている間に、注文状態を更新するために実行されるバッチ プロセスがあるかもしれません。こうしたプロセスがリソースを消費している可能性があるため、こうしたプロセスを考慮に入れるよう気を付けてください。

- 一般に、Web サーバーはテキストやグラフィックスの提供に非常に適しています。平均的サイズのグラフィックスを含む静的ページは、おそらく、動的ページ、フォーム、およびマルチメディア ページよりもあまり注意を払われずに提供されます。

主なシナリオのナビゲーション パスを判断する

主なシナリオの一覧が用意できたので、次のアクティビティでは、個々のユーザーが実際にこうしたシナリオに関連するタスクやアクティビティを実現する方法を判断します。

人間の操作は予測不能なため、一般に Web サイトには、冗長な機能が用意されています。ユーザー数が比較的少ない場合でも、実在のユーザーがタスクを完了する際に行うと考えられるすべてのパスを用意するだけではなく、いや応なく予定外のパスを作成することになることもほぼ間違いありません。ユーザーがアクティビティを完了するために使用するパスごとに、システムにかかる負荷は異なります。その違いは小さい場合も、大きい場合もあります。結局、その違いはテストするまで確認できません。次のように、ナビゲーション パスを判断する方法は多数あります。

- パフォーマンスに大きな影響をもたらすことが予測され、特定された 1 つ以上の主要シナリオを実現する Web アプリケーション内のユーザー パスを特定する。
- 設計や使用法に関するマニュアルを読む。
- 自身でアクティビティの実施を試みる。
- 指示なしでアクティビティの実現を試みている他のユーザーを観察する。

指示書なしのユーザー受け入れテスト、ベータ テスト、または運用環境にアプリケーションがリリースされたら、Web サーバーのログを調べることによって、大多数のユーザーがテスト対象のシステムに対してアクティビティを実現する方法を判断することができます。モデルを現実と比較し、見つかった類似点と相違点に基づいて、追加のテストを行うかどうかを決定することが常に推奨されます。

パフォーマンスのシミュレーションに含めるパスを決定する際に適用したものと同じ、制限を行ううえでの経験則をナビゲーション パスにも適用し、結果をチーム メンバと共有します。足りないと思うもの、優先順位を下げてもよいと思うもの、およびその理由をたずねてください。

考慮事項

主なシナリオのナビゲーション パスを判断する際は、次の点を考慮します。

- サイトへの 1 回のアクセス中に複数のアクティビティを完了するユーザーもいます。
- 1 回のアクセス中に同じアクティビティを複数回完了するユーザーもいます。
- サイトへの 1 回のアクセス中に、実際、アクティビティを 1 つも完了しないユーザーもいます。
- 多くの場合、ナビゲーション パスを最も把握しやすい方法はページ タイトルを使用することです。
- アプリケーションのページ タイトルがナビゲーション パスを把握するのに適していなかったり直感的でなかったりする場合は、ユーザーがアクティビティを完了するために実行する手順によってナビゲーション パスを容易に定義できることがあります。
- 多くの場合、アプリケーションを初めて使用するユーザーがタスクを実現するために使用するパスは、アプリケーションを使い慣れたユーザーとは異なります。この違いを考慮し、新しいユーザーのナビゲーション パスとリピート ユーザーのナビゲーション パスとの比率をモデルでどのように表現するかを考慮します。
- ユーザーによって、サイトで費やす時間は異なります。ログアウトするユーザーもいれば、ブラウザを閉じるユーザーもいます。また、セッションがタイムアウトするまで放置するユーザーもいます。セッション持続時間を判断したり、見積もったりする際は、こうした要因を考慮に入れます。
- ナビゲーション パスについてチーム メンバや他のユーザーと議論する際は、使用法を目に見える形にすることが役立つ場合がよくあります。

視覚的表現の例

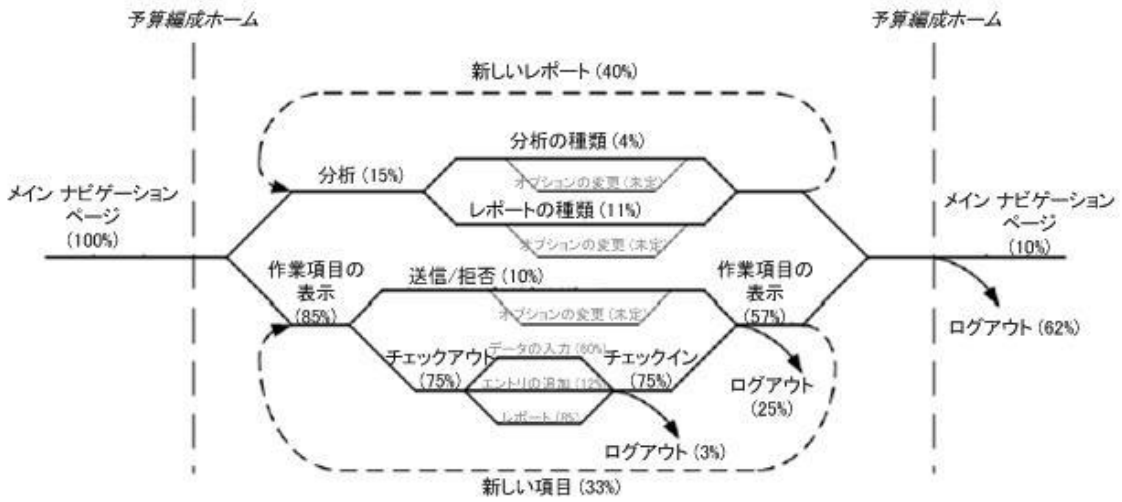


図 12.1 主なシナリオのワークロード

各ユーザーのデータとばらつきを判断する

ナビゲーション パスと使用法のシナリオを表すモデルがどれほど正確でも、各ユーザーが使用するデータや各ユーザーが使用するデータのばらつきが考慮されていなければ、完全なモデルにはなりません。ユーザーを交換可能なエンティティと見なすと、テストの設計と分析がより簡単になり、ある種のパフォーマンスの問題をより検出しやすくもなるのですが、Web サイトが運用環境で直面する可能性がある実社会の複雑さの多くが隠されてしまいます。この複雑さを考慮してシミュレーションを行うことは、現実のユーザーが直面する可能性が最も高いパフォーマンスの問題を見つけるうえで非常に重要であり、運用環境でのパフォーマンス特性についてなんらかの予測を行ううえでの重要な要素でもあります。

ここから先のセクションでは、各ユーザー データとデータのばらつきをモデル化する際に基となる情報源の一部や、モデルの作成やテストの設計を行う際に考慮する必要があるデータとデータのばらつきの一部について詳しく説明します。

Web ログに含まれている Web サイトの指標

ここでの説明における Web サイト指標とは、サイトのトラフィック パターンと負荷パターンをサーバーの観点から理解するのに役立つ可変要素のことです。Web サイトの指標は、一般に、サイトにアクセスするユーザーのフローによって変化する平均値ですが、このような指標によってサイトの使用法の概観を知ることができ、パフォーマンス テストのモデルを作成する際に役

立ちます。こうした指標は結局のところ、Web サーバーのログに含まれています（このようなログを解析してこうした指標を図などの方法で示すソフトウェア アプリケーションは多数存在しますが、ここではこうしたアプリケーションについては扱いません)。(Web サーバーがログを記録するように構成されている場合に) Web サーバー ログから読み取ることができる指標のうち、特に役立つものを以下にいくつか示します。

- **一定期間あたりのページ表示数。**ページ表示とは、依存関係のあるファイル要求（.jpg ファイル、CSS ファイルなど) をすべて含むページ要求です。ページ表示数を時間単位、日単位、または週単位で追跡して、Web サイト上のユーザー アクティビティの周期パターンやピーク時のバーストを考慮することができます。
- **一定期間あたりのユーザー セッション数。**ユーザー セッションとは、既に説明したように、ユーザーからの Web サイトへのアクセスによって生じる、関連する要求のシーケンスです。ページ表示数と同様、ユーザー セッション数を時間単位、日単位、および週単位に追跡します。
- **セッション持続時間。**この指標は、最初のページ要求から最後のページ要求が完了するまでを測定した、ユーザー セッションの持続時間を表します。セッション持続時間では、ユーザーがページ間を移動する際に一時停止する時間も考慮されます。
- **ページ要求の分布。**この指標は、機能の種類 (ホーム、ログイン、支払いなど) に従ってページのヒット数の分布を比率で表します。分布の比率により、ユーザーによる実際の Web サイトの利用に基づいた、ページ ヒット数の重み付け比率が決まります。
- **対話の速度。**この指標は、ユーザーが Web サイトを移動する際のページ間の移行にかかる時間 (思考時間から構成されます) を表します。各ユーザーの Web サイトとの対話速度がそれぞれ異なることを覚えておくことが重要です。
- **ユーザーによる放棄。**この指標は、ユーザーがページの読み込みを始めてから、表示の待ち時間が長くて不満がつのり、サイトの表示を終了することを表します。セッションが放棄されることは、インターネット上では極めて通常のことなので、負荷テストの結果に影響します。

シナリオの相対分布を決定する

シミュレーションを行うシナリオと、そのシナリオ用の手順と関連データを決定し、このようなシナリオを 1 つ以上のワークロード モデルに集約します。そこでワークロード モデルの完成に必要な他のアクティビティと比較して、モデルで表される各アクティビティの実行頻度を確認する必要があります。

1 つのワークロード分布では不十分な場合もあります。調査と経験により、ユーザーのアクティビティは時間の経過と共に大きく変化することが示されています。テストの有効性を確実にするには、時刻、曜日、日付、季節に応じてアクティビティを評価して検証する必要があります。例として、オンラインの請求支払サイトについて考えてみましょう。請求書がすべて毎月 20 日に発行されるとすると、20 日直前にサイトで実行されるアクティビティは、主に、システム管理者によるアカウントの更新や、課金情報のインポートなどでしょう。一方、20 日直後から支払期日である翌月の 5 日までは、顧客が請求書の表示と支払いを行うでしょう。次に、アクティビティの相対分布を決定するための最も一般的な方法を示します。

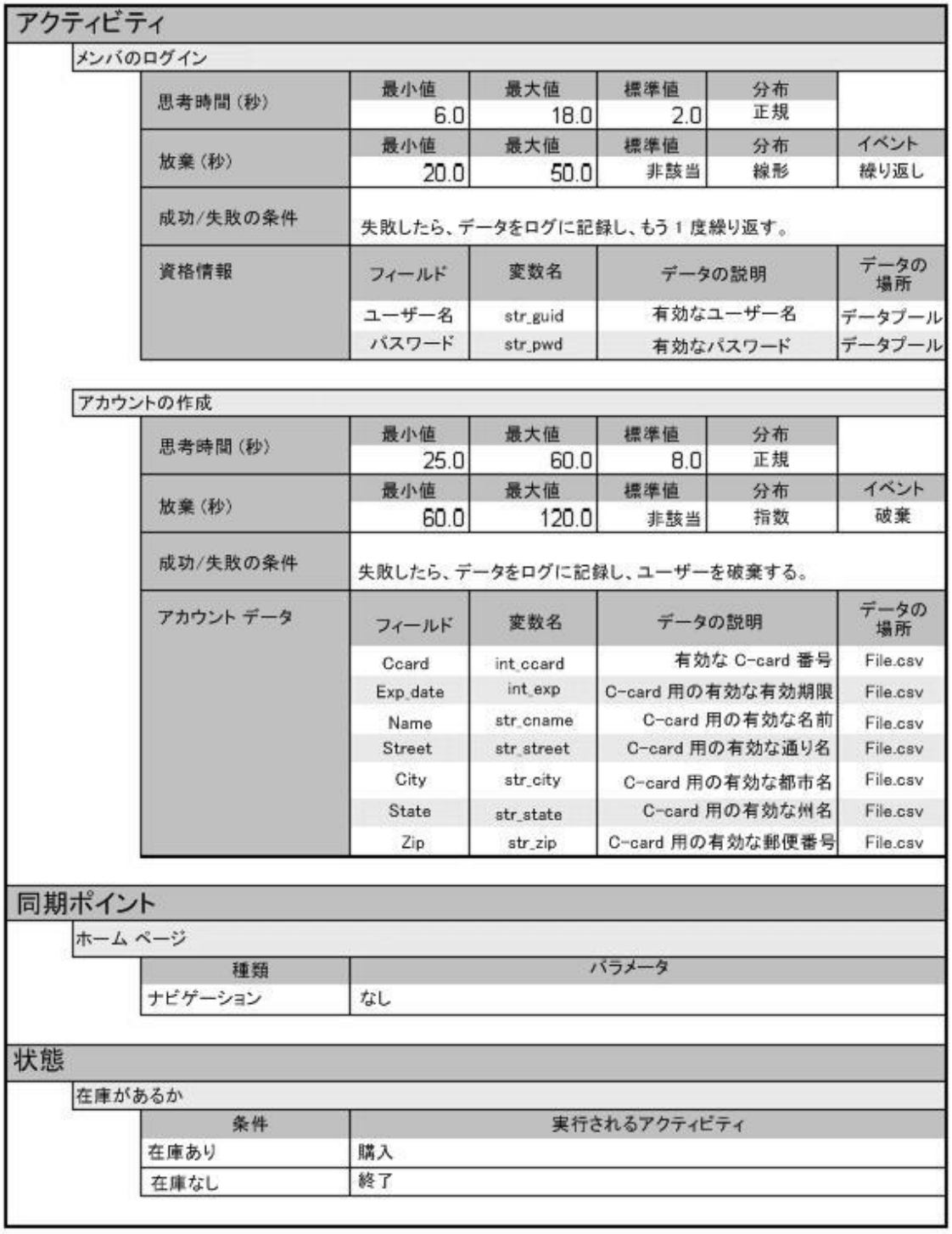
- 実際の用途、負荷の値、一般的と一般的ではない使用シナリオ (ユーザーのパス)、クリック間やページ間のユーザーによる遅延時間、入力データの変化などは、ログ ファイルから直接取得します。
- 求められる機能、つまり使用される可能性が最も高い機能を知るために、新機能について販売部門やマーケティング部門の担当者と面談します。既存のユーザーと面談すると、こうしたユーザーがどの新機能を使用する可能性が最も高いと考えているかを判断することもできます。
- ユーザーを代表するグループ (予想されるユーザー ベースの約 10 ～ 20% の規模) にベータ リリースを展開し、こうしたユーザーが使用したサイトが記録されたログ ファイルを分析します。
- 従業員、顧客、クライアント、友人、家族などに依頼して社内で簡単な実験を行い、ユーザーの自然なパスや、新しいユーザーとリピート ユーザーとのページ表示時間の違いなどを確認します。
- 最後の手段として、自身のサイトに関する知識に基づいて直感を使用したり、最適な想定を行ったりして予測を行います。

チーム メンバは、さまざまな方法を使用して、個々の使用法パターンを 1 つ以上の集約的なモデルにまとめます。こうした方法には、スプレッドシート、ピボット テーブル、説明文、Unified Modeling Language (UML) のコラボレーション図、マルコフ連鎖図、フローチャートなどがあります。どのような方法を採用するにしても、モデル全体をチーム全体にとって理解しやすく、保守しやすく、伝達しやすいものにすることが目的です。

非常に効率の高い方法の 1 つは、ナビゲーション パスと、各アクティビティを実行すると予想されるユーザーの割合とを視覚的に表すモデルを作成することです。このモデルは、エンド ユーザー、開発者、テスタ、アナリスト、および幹部関係者を含むチーム全体が直感的に理解できるようなものにします。ポイントは、チーム メンバが多くのトレーニングを受けなくても理解

できるような言葉や視覚表現を使用することです。実際、最も優れた視覚的モデルは、トレーニングをまったく行わなくても意図する意味が伝わるモデルです。このようなモデルを作成したら、そのモデルをユーザーと関係者の両方に回覧して意見やコメントを求めることが重要です。使用法の主なシナリオをまとめる際に実行した手順に従って、足りないと思うもの、優先順位を下げてもよいと思うもの、およびその理由をチーム メンバにたずねます。チーム メンバは単に新たな割合を視覚的モデルに書き込むという方法もよく使用されます。こうすると、合意に達したアクティビティとそうでないアクティビティを全メンバが非常に理解しやすくなります。

モデルがパフォーマンス テストにふさわしいと確信できたら、"各ユーザーのデータとばらつきを判断する" というアクティビティで各ナビゲーション パス用に収集した各使用法のデータでそのモデルを補完します。その際、実際のテストを作成するために必要なデータがすべてモデルに含まれるようにします。



アクティビティ

メンバのログイン

思考時間（秒）	最小値	最大値	標準値	分布	
	6.0	18.0	2.0	正規	
放棄（秒）	最小値	最大値	標準値	分布	イベント
	20.0	50.0	非該当	線形	繰り返し
成功/失敗の条件	失敗したら、データをログに記録し、もう 1 度繰り返す。				

資格情報	フィールド	変数名	データの説明	データの場所
	ユーザー名	str_guid	有効なユーザー名	データプール
	パスワード	str_pwd	有効なパスワード	データプール

アカウントの作成

思考時間（秒）	最小値	最大値	標準値	分布	
	25.0	60.0	8.0	正規	
放棄（秒）	最小値	最大値	標準値	分布	イベント
	60.0	120.0	非該当	指数	破棄
成功/失敗の条件	失敗したら、データをログに記録し、ユーザーを破棄する。				

アカウント データ	フィールド	変数名	データの説明	データの場所
	Ccard	int_ccard	有効な C-card 番号	File.csv
	Exp_date	int_exp	C-card 用の有効な有効期限	File.csv
	Name	str_cname	C-card 用の有効な名前	File.csv
	Street	str_street	C-card 用の有効な通り名	File.csv
	City	str_city	C-card 用の有効な都市名	File.csv
	State	str_state	C-card 用の有効な州名	File.csv
	Zip	str_zip	C-card 用の有効な郵便番号	File.csv

同期ポイント

ホーム ページ

種類	パラメータ
ナビゲーション	なし

状態

在庫があるか

条件	実行されるアクティビティ
在庫あり	購入
在庫なし	終了

図 12.2 ナビゲーション パスの視覚的モデル

考慮事項

シナリオの相対分布を決定する際は、次の点を考慮します。

- 視覚的モデルを作成し、それをユーザーと関係者に回覧して意見やコメントを求めます。
- 技術系以外のユーザー、技術系の設計者、両者の中間に位置するすべてのメンバのいずれにとっても直感的に理解できるモデルにします。
- パフォーマンス テストでは、多くの場合、大量のテスト データを使用するため、データ ファイルに十分な量のデータを含めるようにします。
- 実際のテストの作成に必要な補足データをすべてモデルに含めるようにします。

対象の負荷レベルを特定する

顧客から Web サイトへの 1 回のアクセスは、ユーザー セッションという一連の関連要求で構成されます。同じ Web サイトを操作するユーザーでも操作方法が異なれば、セッション中の Web サーバーへの要求が重なり合うことはあまりありません。そのため、ユーザー エクスペリエンスを同時実行ユーザーに基づいてモデル化するのではなく、ユーザー セッションに基づいてモデル化する方が役立ちます。ユーザー セッションは、Web サイトにアクセスする顧客によって行われる、ページの移動フロー内の一連のアクションと定義できます。

アプリケーションの使用量を定量化する : 理論

Web ベースのマルチ ユーザー アプリケーションはステートレス プロトコルを使用して通信を行うため、多くの場合、アプリケーションの使用量を特定し表現するのは困難です。"同時実行ユーザー" や "同時ユーザー" といった用語がよく使用されますが、ユーザーによる Web サイトへのアクセスのモデル化に関してこれらの用語を使用すると、誤解を招く可能性があります。以下に示す図 12.3 と図 12.4 では、各線分が 1 つのユーザー アクティビティを表しており、異なるアクティビティは異なる色で表しています。黒で塗りつぶした線分は、"ホーム ページを読み込む" というアクティビティを表しています。グラフ上では、ユーザー セッションを水平方向に表しています。この仮想表記では、各ユーザーが同じアクティビティに費やす時間を同じにしています。"モデルの開始" という直線と "モデルの終了" という直線の間の経過時間は 1 時間です。

図 12.3 サーバーの観点から見たユーザー アクティビティ

上記の図 12.3 は、サーバー (この場合は Web サーバー) の観点から見た使用量を表しています。グラフを上から下、および左から右に見ていくと、ユーザー 1 がまず "黒一色" のページを閲覧し、続いて "白"、"水玉模様"、"黒一色"、"白"、"水玉模様" のページの順に移動していることがわかります。ユーザー 2 も最初に "黒一色" のページを閲覧していますが、続いて “縞模様” のページ、“灰色” のページなどに移動しています。また、グラフを垂直方向にスライスすると、開始時と終了時の間のほとんどどの時点においても、10 人のユーザーがシステムにアクセスしていることもわかります。これは、この分布が 10 人の同時実行ユーザー (同時ユーザー) を表すことを示しています。サーバーは、どの時点でも 10 個のアクティビティが発生しているということは認識していますが、実際に何人のユーザーがシステムと対話しているためにこの 10 個のアクティビティが発生しているかということは認識していません。

以下の図 12.4 も、個々のユーザーによるアクティビティの分布を示しています。これをサーバーの観点から見たものが上記のグラフです。

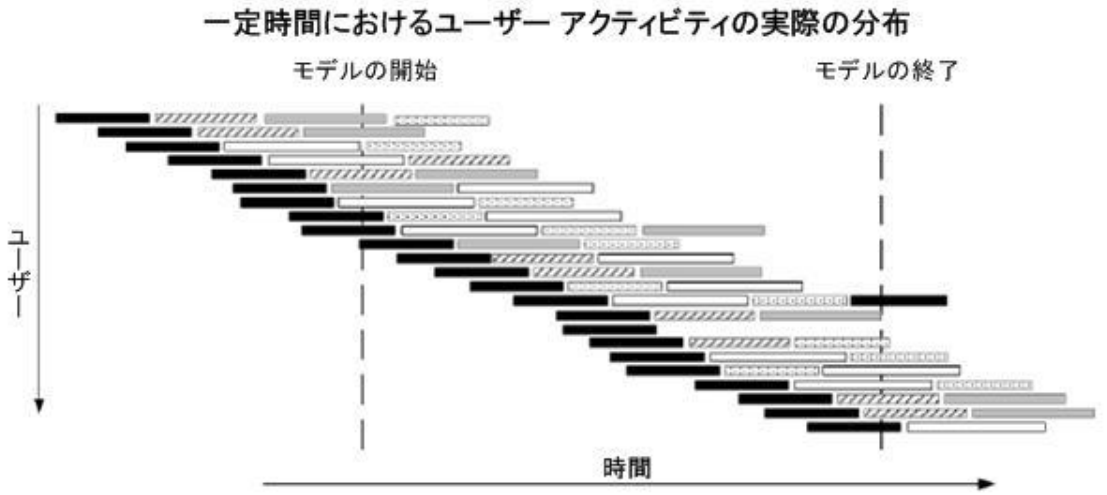


図 12.4 一定時間におけるユーザー アクティビティの実際の分布

このグラフでは、各ユーザーの 23 個のアクティビティが捉えられています。このユーザーそれぞれが、モデル化されている時間中になんらかのアクティビティを実行しており、各自のアクティビティは 23 個のユーザー セッションと見なすことができます。23 人のユーザーはそれぞれ異なる時間帯にサイトとの対話を開始しています。すべてのユーザーが“黒一色”のアクティビティから開始したという点を除けば、アクティビティの順序に特定のパターンはありません。この図で表されている 23 人のユーザーは、実際、図 12.3 に示されているものとまったく同じアクティビティを同じ順序で行っています。しかし、図 12.4 に示すように、同時実行ユーザーはどの時点でも 9 ～ 10 人存在します。上記の場合、量の観点からの使用法のモデル化は、1 時間あたりの総ユーザー数、つまり、"モデルの開始" と "モデルの終了" の間にあるユーザー セッションの数という観点から考えることができます。

ある程度の経験上のデータ (アプリケーションの以前のリリースの Web サーバー ログなど) が揃わなくても、対象の負荷レベルが特定の対象になることは間違いありません。大半の場合、こうした対象は、アプリケーションに関連する目標と、その目標が市場浸透、収入生成、その他のいずれであるかに基づいて、ビジネスによって設定されます。このような対象は、当初、希望する数値を表します。

アプリケーションの使用量を定量化する

アプリケーションの現在の実装 (それが、運用環境での、以前のリリース、代表的なプロトタイプ、ベータ リリースのいずれの実装であれ) の Web サーバー ログにアクセスできる場合は、こうしたログのデータを使用して、上記のリソースから収集したデータの検証や強化を行うことができます。Web サーバー ログの定量分析を行うことで、次の点を確認できます。

- 一定期間内 (月/週/日) でのサイトのアクセス総数
- 全体の平均負荷とピーク時の負荷の点から見た使用量 (時間単位)
- 全体的な平均とピーク時の負荷に関するセッションの持続時間 (時間単位)
- 全体の平均負荷とピーク時の負荷 (時間単位)。この値は、実際のスケーラビリティの量をシミュレーションするために、重なり合うユーザー セッション数に変換されます。
- 使用法に大きな変化をもたらすビジネス サイクルや特別なできごと

対象の負荷レベルの特定に使用する入力と出力を以下に示します。

入力

- Web サーバー ログから取得した使用法に関するデータ
- 対象に対応する業務量 (現在値と計画値の両方)
- 主要シナリオ
- 作業分布
- セッションの特性 (ナビゲーション パス、持続時間、新しいユーザーの割合)

出力

量に関する情報を、これまでの手順で特定および決定した、目標、主なシナリオ、ユーザーによる遅延、ナビゲーション パス、およびシナリオの分布と組み合わせると、特定の対象負荷がかかったワークロード モデルの実装に必要な残りの詳細を判断できます。

モデルのばらつきを統合する

運用データが手に入るまでは使用法モデルは "精一杯の推測" なので、対象の負荷それぞれにつき少なくとも 3 つの使用法モデルを作成することをお勧めします。これにより、パフォーマンスの測定におおよその信頼区間を追加することができます。関係者は、1 つのテストから導き出される、誤りを犯しがちな多くの仮定に基づいた結果と、こうした仮定に含まれる多くの不正確さから、アプリケーションのパフォーマンス特性に影響を与えかねない結果に注目してしまう可能性があります。

一般にチームにとって最も重要な 3 つの使用法モデルを以下に示します。

- 予想される使用法 ("各ユーザーのデータとばらつきを判断する" というアクティビティで作成したモデル)
- 最善のパフォーマンスを想定した使用法 (パフォーマンス コストの低いアクティビティばかりが実行される)
- 最悪のパフォーマンスを想定した使用法 (パフォーマンス コストの高いアクティビティばかりが実行される)

次のグラフは、この 3 つのモデルすべてをテストした場合に得られる情報の例です。ご覧のとおり、この例の場合は、"予想される使用法" と "最善の場合を想定した使用法" のパフォーマンス特性は似ています。しかし、"最悪の場合を想定した使用法" でサポートできる総負荷は "予想される使用法" でサポートできる総負荷よりも 50% 近く少ないことがわかります。このよ

うな情報を基に、場合によっては、使用法モデルを再検討したり、経験上のデータが手に入るまで "最悪の場合を想定した使用法" モデルを一種の安全性要因として引き続きテストすることを決定したりします。

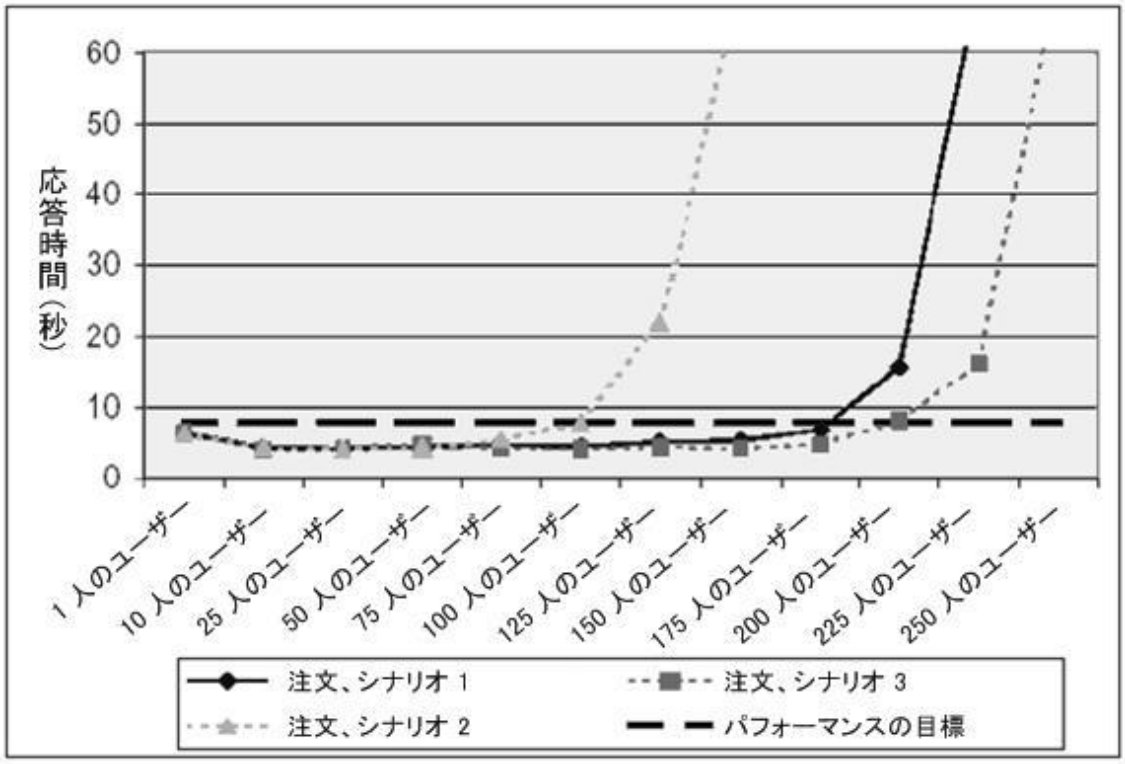


図 12.5 使用法モデル

考慮事項

対象の負荷レベルを特定する際は、次の点を考慮します。

- 上記のアクティビティによる負荷の量は、アプリケーションに実際にかかる負荷と相互に関連している場合も関連していない場合もありますが、開発されたアプリケーションや展開されたアプリケーションが対象の負荷をサポートするかどうか、およびどの程度適切にサポートするかどうかという情報がビジネスには必要です。
- 構築したワークロード モデルでは各アクティビティの頻度が総負荷に対する割合で表されるため、対象の負荷レベルを特定した後でモデルを更新すべきではありません。
- 各ワークロード モデルはさまざまな負荷レベルで実行されることが多く、また、ほとんどの負荷生成ツールを使用すると非常に容易に負荷レベルを実行時に変更できることが多いのですが、それでもやはり、運用環境での状況を予測したり運用環境での状況と比較したりするために、各ワークロード モデルの、予想される対象の負荷レベルとピーク時の対象

の負荷レベルを特定することは重要です。負荷レベルが少し変化しただけでも、結果が劇的に変わることがあります。

モデルを実装する準備を整える

実行可能なテストとしてのワークロード モデルの実装は、実装方法 (通常、負荷生成ツールでのスクリプト作成) と密接に関係しています。テストの実装と検証の詳細については、「第 14 章 テストの実行」を参照してください。

考慮事項

モデルを実装する準備を整える際は、次の点を考慮します。

- 深刻な問題点が存在しない場合は、使用しているツールでの実装が難しいという理由だけでモデルを変更しないでください。
- 設計されたとおりにモデルを実装できない場合は、実際に実装するモデルに関する詳細を記録するようにします。
- モデルを実装する作業には、多くの場合、収集する指標を特定する作業と、こうした指標の収集方法を決定する作業が含まれます。

まとめ

運用パフォーマンスの把握、予測、またはチューニングを目的としてパフォーマンス テストを実施する際は、運用環境での使用法や予測される将来の業務量と同様の、または少なくともこれらに近い条件でテストを行うことが非常に重要です。

正確で予測に役立つテスト結果を得るためには、ユーザーの操作を決定する際に、ユーザーの Web サイトとの対話方法に固有の要因 (ページ フロー、ヒット率、ユーザーがページ間を移動する際に一時停止する時間など) に基づいて顧客のセッションをモデル化する必要があります。

第 13 章 各のユーザーのデータとばらつきの判断

目的

- ユーザーによる遅延時間の現実的な長さと分布パターンを判断する方法について学習する。
- ユーザーによる現実的な遅延をテスト設計やテスト スクリプトに組み込む方法について学習する。
- ワークロードの特徴付けを定義する際に考慮する主な変数について学習する。
- 負荷テスト作成時にユーザー エクスペリエンスのモデル化に役立つ、ユーザーの操作要素について学習する。

概要

ここでは、個別のユーザーによる現実的な遅延、ユーザー データ、およびユーザーによる放棄を判断するプロセスについて説明します。パフォーマンス テストでアプリケーションの運用時のパフォーマンス特性の把握に直接つながる結果をもたらすには、テスト対象のワークロードが実際の運用環境を表す必要があります。現実をかなり正確に表すには、典型的なユーザー群に見受けられるばらつきや不規則性をある程度備えたユーザーをモデル化する必要があります。

本章の使い方

ここでは、ワークロードの特徴付けで現実的な使用法パターンが作成されるように、ユーザーによる遅延、ユーザー データ、およびユーザーによる放棄などの変化をモデル化して、運用環境のシミュレーションの精度を高める方法について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「ユーザーによる遅延」とその後のセクションでは、ユーザーによる遅延のモデル化の主要概念と、ワークロードの特徴付けへの影響を理解します。
- 「個別のユーザー データの決定」では、ユーザー データの主要概念と、ワークロードの特徴付けへの影響を理解します。
- 「ユーザーによる放棄」では、ユーザーによる放棄の主要概念と、ワークロードの特徴付けへの影響を理解します。

ユーザーによる遅延

ユーザー モデルの精度が高いほど、パフォーマンス テスト結果の信頼性も高まります。ユーザ

ーを正確にモデル化する際に見落とされがちな 1 面として、ユーザーによる遅延のモデル化があります。ここでは、ユーザーによる遅延時間を決定する方法について説明します。この遅延時間は、ワークロード モデル、パフォーマンス スクリプトの順に組み込まれます。

ユーザーはセッション中に、参照、システムへのログオンなど、多種多様な状態になります。顧客はさまざまなモードで Web サイトと対話します。サイトに精通していて、あるページから別のページにすばやく移動するユーザーもいれば、実行する操作を判断するのに時間がかかるユーザーもいます。そのため、ユーザーの操作を特徴付けるには、ページ フロー、ヒット率、ユーザーがページを表示するたびに一時停止する時間、および Web サイトとのユーザーの対話方法特有の他の要因に基づいて、顧客のセッションをモデル化する必要があります。

ユーザーによる遅延を適切にモデル化しなかった場合の結果

現実に即した負荷テストにするには、ユーザーによる遅延時間の変化の概念を無視するのではなく、遅延の範囲や分布を当てはめる際に意味のある試みを行います。各ユーザーが各ページにまったく同じ時間を費やすような負荷テストを作成することは、まったく現実的ではなく、誤解を招きかねない結果が生成されます。たとえば、実に簡単に次のような結果に帰結します。

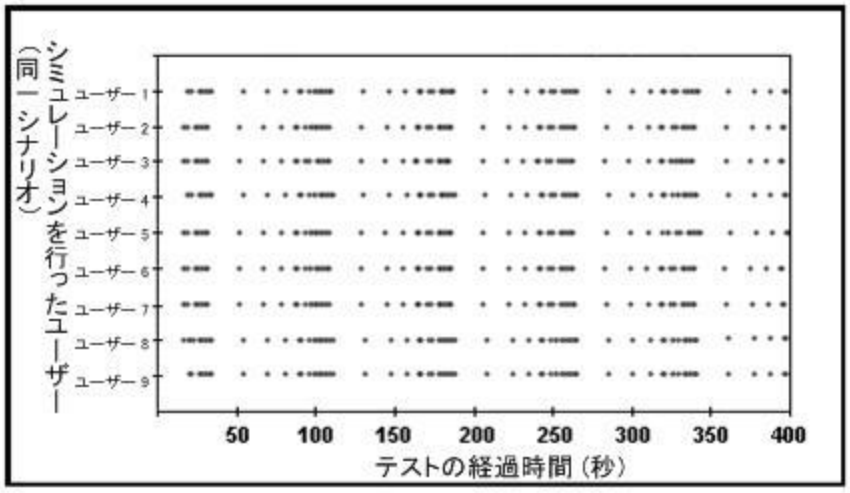


図 13.1 ユーザーによる静的遅延を使用した場合の結果

応答グラフに詳しくない方のために、各ドットはユーザー アクティビティ (この場合はページ要求) を表し、横軸はテストの実行を開始してからの経過時間 (秒単位) を示します。また、縦軸は個々の仮想テスタを示しています。この特定の応答グラフは、"バンド" または "ストライピング" の例です。負荷テストやパフォーマンス テストを実行する際はバンドを回避すべきで

すが、ストレス テストでは役立つこともあります。サーバーの観点から見ると、このテストは、同じ操作 (ホーム ページから、x 秒間待機してページ 1 に移動する) を 10 人のユーザーが同時実行しているのと同じです。

さらに細かく見ていくために、定規を画面に対して垂直に持ち、グラフの左から右へゆっくりと動かします。これがサーバーで認識される状態で、ドットがない状態が続いた後、多数のドットが存在し、またドットがなくなります。これでは、実際のユーザー コミュニティ表しているとはとても言えません。

次の図は、小さい範囲の一様な正規分布の遅延を同じテストに追加することで、実際のユーザーをはるかに適切に表しています。

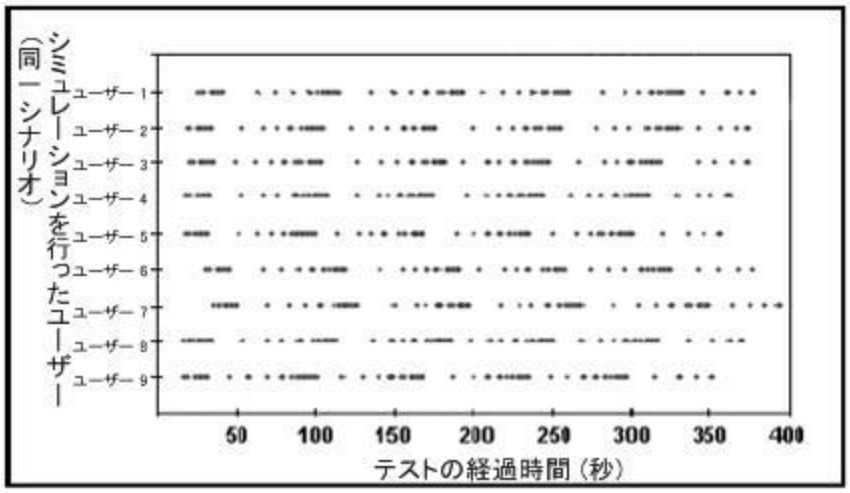


図 13.2 ユーザーによる正規分布の遅延を使用した場合の結果

前回と同じように定規を使ってみると、今回はドットがむらなく分散されていることがわかります。これにより、シミュレーションを行った負荷の現実性と、パフォーマンス テスト結果の正確さが両方とも大幅に向上します。

手順 1 - ユーザーによる遅延を決定する

ユーザーが Web ページのコンテンツを表示中に生じるた遅延 (一般に、思考時間と呼びます) は、「ユーザーがログオン資格情報を入力するのにどのくらい時間がかかるか」や「ユーザーがこのページを読み取るのにどのくらい時間がかかるか」といった質問への回答になります。Web サイトでのユーザー操作に関連する思考時間を推測するには、いくつか異なる方法を使用できます。当然、運用サイトについて収集した実データを使用することが最善の方法です。ただし、通

常、サイトが運用環境にリリースされる前にテストが行われるため、この方法はほぼ不可能です。そのため、サイト上でのアクティビティに関して、経験に基づく推測や推定を行う必要があります。

ユーザーによる遅延の決定に最もよく役立つ方法には、次のものがあります。

- 既に運用中の Web サイトをテストする際は、各ページのログ ファイルから、ユーザーの表示 (または入力) 時間の平均偏差と標準偏差を抽出することで、実際の値や分布を決定できます。こうした情報により、各ページの思考時間を簡単に判断できます。運用サイトには、このような情報を直接提供する Web トラフィック監視ソフトウェアが用意されている場合もあります。
- ログ ファイルがない場合は、従業員、顧客、クライアント、友人、家族などに依頼して社内で簡単な実験を行い、新しいユーザーとリピート ユーザーとのページ表示時間の違いなどを確認できます。この種の簡易有用性調査は、運用されていない Web サイトのデータを収集する非常に効率的な方法であり、他の方法を使用して収集されたデータの検証方法にもなります。
- 自身でサイトを使用したり、同じサイトで同様の操作を行ったりして時間を計測します。明らかに、この方法は、個人的先入観が入り込みやすくなりますが、ユーザー受け入れテスト (UAT) 時に実際のユーザーの時間を計測するか、独自の有用性に関する調査を実施する機会が得られるまでの出発点としては妥当です。
- 適切な情報源がない場合、Nielsen//NetRatings、Keynote、MediaMetrix などの調査会社が既に収集した指標や統計を利用できます。このような統計では、ユーザーと Web サイトの一般的なサンプルに基づいて、平均的なページ表示時間とユーザー セッションの持続時間に関するデータが提供されます。こうした数値は特定の Web サイトからのものではありませんが、第一近似値としては効果的に機能します。

大量のデータを統計的に収集することに長い時間をかけたり、極端に厳密である必要はありません。実際に必要なのは、一般的なユーザーが操作を実行する際にかかる時間 (1 ～ 2 秒ずれてもかまいません) を把握することです。ただし、サイトの性質によっては、初めて使用するユーザーと使用経験のあるユーザーの遅延時間を別々に決定することが必要になる場合もあります。

手順 2 - 遅延の範囲を当てはめる

あるユーザーがページを表示する際にかかる時間や、ユーザー間の時間差を特定するだけでは不十分です。つまり、ユーザーごとに遅延時間を変化させる必要があります。各ユーザーが、ある

ページにまったく同じ時間を費やすことはほとんどありません。すべてのユーザーがあるページに同じ時間を費やすようなパフォーマンス テストを実施すると、非現実的な結果、または少なくとも信頼性の低い結果が出る可能性が極めて高くなります。

手順 1. の遅延時間または遅延の範囲を、ユーザー間のばらつきも表すように変換するには、次の 3 つの情報が必要になります。

- 最短遅延時間
- 最長遅延時間
- この 2 つの時点間のユーザーによる遅延の分布またはパターン

手順 1. の分析から最短時間と最長時間がわからない場合は、次のような経験則を当てはめ、許容可能な推測値を決定します。

- 最短時間は、次のように決まります。
  - ページ移動を目的として、長時間滞在しない使用経験のあるユーザー (次のリンクを探してクリックするためにページを読み込むだけのユーザーなど)。
  - クリックして間違ったページに移動したことに気付いたユーザー。
  - すべての値が事前入力されているフォームをクリックしたユーザー。
  - ユーザーが必要な情報をフォームに入力するのに必要と考えられる最短時間。
  - "一般的" と判断した値の半分。
- 最長時間は、次のように決まります。
  - セッション タイムアウト。
  - ユーザーがフォームの情報を調べるのに十分な時間。
  - 処理速度の遅いリーダーでページ全体を読み取るのにかかる時間を超えない時間。
  - ユーザーが読み取ることを想定するテキストをはっきりと 3 回読み取るための時間 (これは、映画業界でスクリーン上に表示されるテキストに使用されている経験則です)。
  - "一般的" と判断した値の 2 倍。

予測をなるべく現実に近づける必要はありますが、当てはめる範囲に使用経験のあるユーザーの 75% 以上が含まれていれば、結果が意図せず歪曲されるのを防ぐには十分です。

手順 3 - 分布を当てはめる

次の種類の分布には多くの数学モデルがあります。このようなモデルのうち次の 4 つが、ユーザーによる遅延のシナリオの圧倒的大多数に対応しています。

- 直線分布または一様分布
- 正規分布
- 負の指数分布
- ふたこぶ型正規分布

直線分布または一様分布

最短時間と最長時間の間をモデル化する際、最も簡単なのが一様分布です。この分布モデルでは、範囲の上限と下限の間に均等に分散される乱数を選択するだけです。つまり、生成された数値が範囲の中央や片側に偏ることはありません。次の図は、0 ～ 25 の範囲に生成される 1000 個の値の一様分布を示しています。一様分布は、最小値と最大値が非常に明確な場合に使用しますが、このような 2 つの端点間で区別可能なパターンが使用されるか、使用されることが期待されます。

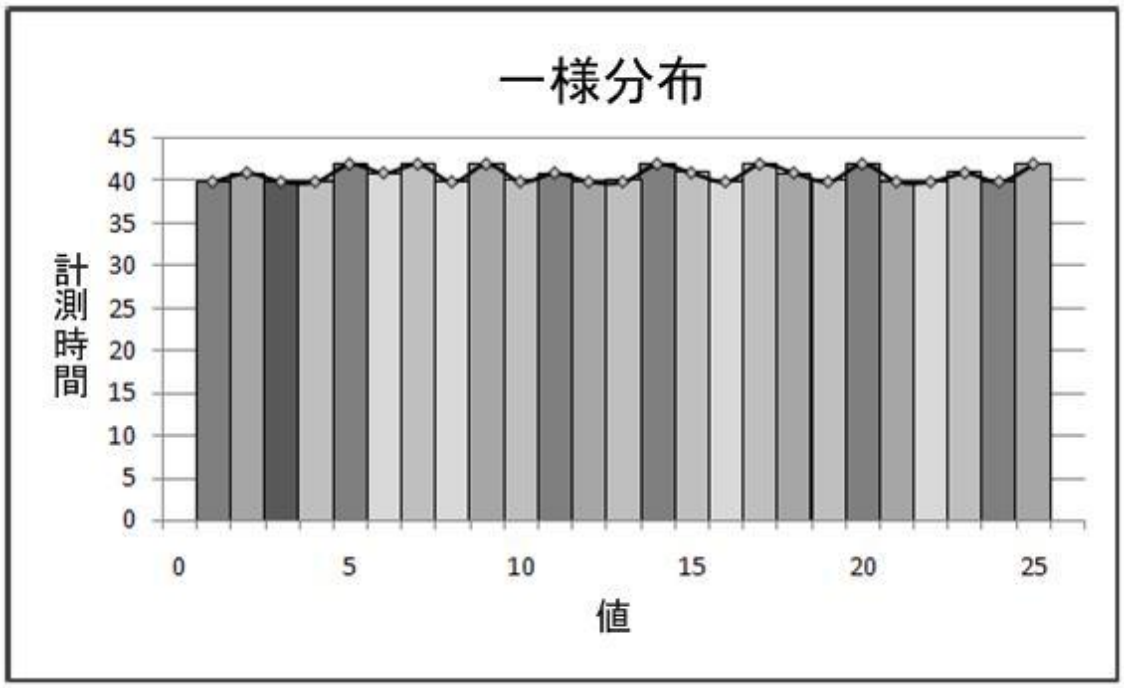


図 13.3 一様分布

正規分布

正規分布 (釣鐘曲線とも呼ばれます) は、モデル化は困難ですが、ほぼすべての場合に、正確さが向上します。この分布モデルでは、選択値が中央 (つまり平均値) に偏るように、数値がランダムに選択されます。次の図では、0 ～ 25 の範囲に生成される 1000 個の値の正規分布を示しています (平均値は 12.5、標準偏差は 3.2 です)。一般に、正規分布は実際のデータが使用できない場合に、定量化できる多数の見本測定値の最も精度の高い数学モデルと考えられます。パターンが両端よりも中央に集中すると想定される場合に正規分布を使用します。標準偏差値の有効範囲は、0 (最大値と最小値の中間の静的遅延に相当) から、最大値から最小値を引いた値 (一様分布に相当) までです。実際の標準偏差を決定する手段がない場合、適正な近似値は、遅延の 25% (または範囲の 0.25 倍) です。

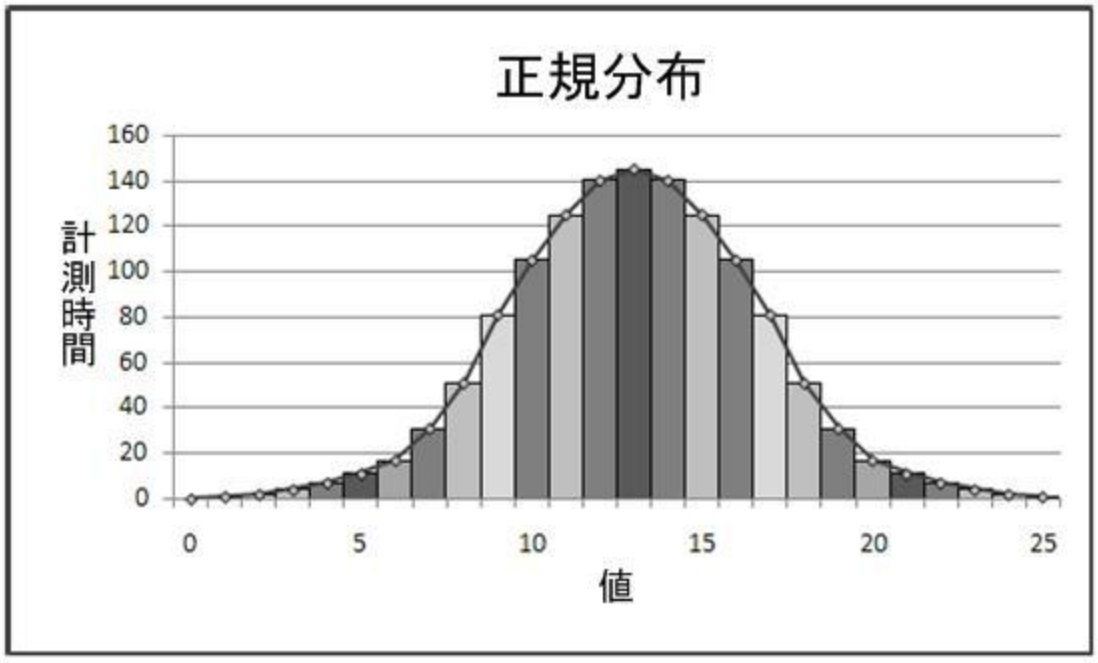


図 13.4 正規分布

負の指数分布

負の指数分布では、次のグラフに示すような分布が作成されます。このモデルでは、遅延時間が範囲の片側に大きく偏っています。このモデルは、マルチメディア コンテンツの再生完了後、その時点でのみ有効になる "もう一度再生" リンクをユーザーがクリックするといった場合に最も役立ちます。次の図では、0 ～ 25 の範囲に生成される 1000 個の値の負の指数分布を示

しています。

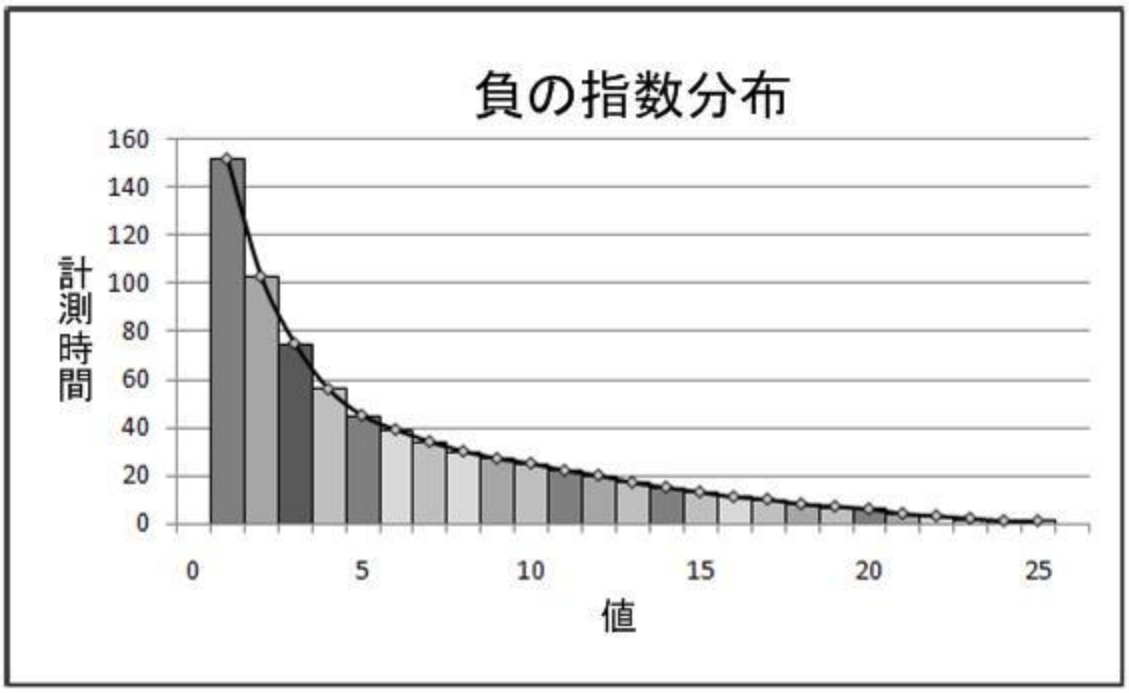


図 13.5 負の指数分布

ふたこぶ型正規分布

ふたこぶ型正規分布では、次のグラフに示すような分布が作成されます。大量のテキストを含む Web ページを初めて参照する場合を考えると、この分布がどのような場合に使用されるかを理解できます。初めて参照するときは、おそらくテキストを読むことになりますが、次に参照するときは、ページの途中を読み飛ばし、ページ下部のリンクをクリックして別のページに移動するだけでしょう。この分布は、まさにこのような種類のユーザー操作を表します。次の図は、このページを参照するユーザーの 60% はそのページで次にクリックするリンクを探すのに約 8 秒しかかけておらず、残りの 40% のユーザーは実際にページ全体を約 45 秒かけて読んでいることを示しています。2 つのこぶがそれぞれ、異なる最小値、最大値、および標準偏差値の正規分布であることがわかります。

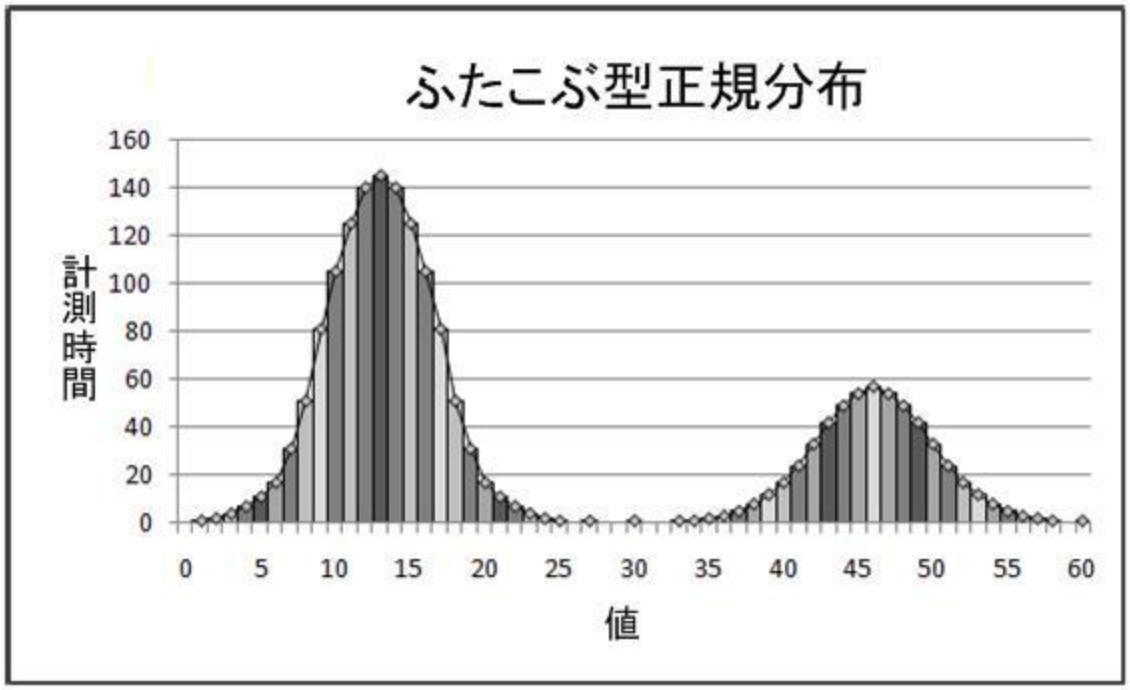


図 13.6 ふたこぶ型正規分布

このパターンを実装するには、ユーザーの比率を表す 1 ～ 100 の数値を生成するコードを記述するだけです。その数値が特定のしきい値を下回る場合 (上記のグラフでは 61 未満) は、最初の分布パターンを使用して遅延を生成するパラメータを指定して正規分布関数を呼び出します。数値がそのしきい値以上の場合は、2 番目の分布パターンを生成するのに適切なパラメータを指定して正規分布関数を呼び出します。

個別のユーザー データの決定

主要シナリオの一覧を用意したら、各ユーザーが実際にこうしたシナリオに関連するタスクやアクティビティを実現する方法と、そのタスクやアクティビティを実現するユーザーに関連するユーザー固有のデータを決定する必要があります。

残念ながら、ナビゲーション パスだけでは、ワークロードのシミュレーションの実装に必要なすべての情報は提供されません。ワークロード モデルを完全に実装するには、さらにいくつかの情報が必要です。これには、次のような情報があります。

- ユーザーが 1 ページに費やす時間
- 各ページに入力する必要のあるデータ
- ユーザーがナビゲーション パスを変更することになる状況

考慮事項

ナビゲーション パスやシミュレーションを行ったユーザーの一意データを特定する際は、次の点を考慮します。

- パフォーマンス テストでは、多くの場合、大量のテスト データが使用されます。効果的なテストを実施できるだけのデータがあることを確認します。
- 同じデータを繰り返し使用すると、多くの場合、有効なパフォーマンス テスト結果が得られません。
- パフォーマンス テストを設計およびデバッグする際は特に、データによってテスト データベースに大きな負荷がかかることがあります。シミュレーション試行時は、非現実的な量のデータがデータベースに格納されているかどうかを定期的に確認します。
- パフォーマンス テストに無効なデータを含めることを検討します。たとえば、1 回目にパスワードを誤入力し、2 回目に正しく入力するユーザーを含めます。
- 通常、初めて使用するユーザーは、使用経験のあるユーザーよりも各ページや操作に費やす時間が長くなります。
- 考えられる最適なテスト データは、運用データベースまたはログ ファイルから収集したテスト データです。
- クライアント側でのキャッシュを検討します。初めて使用するユーザーは、サイトのすべてのオブジェクトをダウンロードしますが、頻繁に参照しているユーザーは、多くの静的オブジェクトや Cookie を自身のローカル キャッシュに格納している可能性があります。ユーザー操作の一意性を把握する際は、そのユーザーが初めて使用するユーザーか、クライアント側で確立されたキャッシュを使用するユーザーかを考慮します。

ユーザーによる放棄

ユーザーによる放棄とは、パフォーマンスの低下が原因で、タスクを完了する前に Web サイトを終了する状況を示します。ユーザーが許容できるパフォーマンスは、そのユーザーの精神面や要求するページの種類によって異なります。ユーザーが放棄することを考慮しないと、非常に非現実的で起こりそうにない負荷が生じることになります。負荷テストでは、パフォーマンスの低下が原因で Web サイトを放棄する可能性のあるユーザーの数をレポートします。

典型的な Web サイトのトラフィック パターンでは、負荷がシステムやアプリケーションで処理できないほど大きくなると、サイトの処理速度が低下します。その結果、ユーザーがそのサイトを放棄するため、システムの速度が許容可能な速度に戻るまで負荷は減少します。一部のユー

ザーを失うことになったとしても、放棄により (過負荷が生じた時点) 以前のレベルのパフォーマンスに回復する自己監視メカニズムが作成されます。したがって、ユーザーによる放棄を正確に考慮する理由の 1 つは、"一部のユーザー" が何人のユーザーであるかを確認することです。もう 1 つは、顧客を失うまでに、アプリケーションが保持できる実際量を確認することです。さらに、現実的には考えられないボトルネックのシミュレーションを行い、そのボトルネックを解消するような状況を避けることです。

放棄をまったく考慮しないと、負荷テストで要求したページまたはオブジェクトを受け取るまでいつまでも待機する可能性があります。最終的にそのオブジェクトを受け取ったら、"結果的に" 実在のユーザーよりも長時間待機しているとしても、何事もなかったように次のオブジェクトに進みます。単にオブジェクトに対する要求が認められなければ、その要求がスキップされ、そのオブジェクトがユーザーにとって重要だったかどうかに関係なく、テスト実行ログに記録されます。放棄を考慮しない状況が現実を的確に表している場合もあることに注意してください。たとえば、必要なタスクを完了する別の方法がないため待機するしかないユーザー向けに専用に作成された Web ベースのアプリケーションなどです。

考慮事項

ユーザーによる放棄に関連する一般的に役立つガイドラインを次に示します。

- 応答時間を評価する前に放棄の割合を確認します。特定のページでの放棄の割合が約 2% 未満の場合は、このような応答時間が生じる可能性はないと考えます。
- 負荷に関する結論を出す前に放棄の割合を確認します。ユーザーが放棄するたびに、負荷をかけるユーザーが 1 人減ることを考えます。応答時間の統計は問題なく思えても、75% のユーザーが放棄すると、テストの想定よりも負荷が約 75% 軽くなります。
- 放棄の割合が約 20% を超える場合、放棄ルーチンを無効にして、問題の原因に関する情報を収集するのに役立つテストを再実行することを検討します。

まとめ

ワークロードの特徴付けで正確な結果を生成するは、現実に即したユーザーによる遅延をテストやテスト スクリプトに設計するプロセスが重要です。パフォーマンス テストでアプリケーションの運用時に予測される将来の業務量やパフォーマンス特性の把握に直接つながる結果をもたらすには、テスト対象のワークロードが現実に即したユーザーによる遅延のパターンを再現する必要があります。

現実をかなり正確に表すには、典型的なユーザー群に見受けられる個別ユーザー データやユーザーによる放棄を考慮に入れることで、ばらつきや不規則性をある程度備えたユーザーによる遅延をモデル化する必要があります。

第 VI 部

テストの実行

内容 :

- **テストの実行**

第 14 章 テストの実行

目的

- パフォーマンス テストの実行に関する一般的な原則と考慮事項について理解する。
- パフォーマンス テストの実行の一般的なアクティビティについて理解する。

概要

パフォーマンス テストの実行は、テスト スクリプトの開発と、テスト結果のレポートおよび分析の間に行うアクティビティです。現在利用できるパフォーマンス テスト関連のトレーニングの多くは、このアクティビティを、テストを開始し、想定どおりに実行されていることを確認するために監視することにすぎないとしています。このアクティビティは、実際には、単にボタンをクリックして、コンピュータを監視することよりもはるかに複雑です。ここでは、実社会の非常に多くのプロジェクトから得た知識に基づいて、このような複雑さを扱います。

本章の使い方

ここでは、パフォーマンス テストの実行と、それに伴うさまざまなアクティビティの基盤となる主要な原則と考慮事項について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「テスト実行のアプローチ」では、パフォーマンス テスト実行のアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- アクティビティに関するさまざまなセクションで、パフォーマンス テストの実行に関連する各アクティビティの詳細を理解します。

テスト実行のアプローチ

パフォーマンス テストの実行には、次のアクティビティが関連します。

- テスト環境の検証
- テストの検証
- テストの実行
- ベースラインとベンチマーク
- テストの保管

以下のセクションでは、これらの各アクティビティについて詳しく説明します。

テスト環境の検証

目標は、できる限り厳密に運用環境をテスト環境に反映することです。通常、テストの設計時に、テスト環境と運用環境の違いをすべて記録および考慮します。テストを実行する前に、期待する構成や設計した構成にテスト環境が一致していることを確認することは重要です。テスト環境と、設計したテスト対象の環境とがわずかでも異なると、そのテストはまったく機能しないか、さらに悪いことには、機能しても誤解を招くデータが提供される可能性が高くなります。

テスト環境を検証する際は、多くの場合、次のアクティビティが役立ちます。

- テスト環境が、指標を収集する目的に向けて正しく構成されていることを確認します。
- ウイルス対策ソフトウェアやスパイウェア対策ソフトウェアがリソースを使用した結果としてデータが無意識のうちに歪曲する副作用の可能性を最小限に抑えるために、テスト中は負荷を生成するコンピュータのウイルス スキャンを停止します。
- 必要な場合は、バックグラウンドで行われるアクティビティのシミュレーションを行うことを考慮します。たとえば、多くのサーバーでは、ユーザーの要求を処理しながら、あらかじめ設定された時間帯にバッチ処理を実行します。このようなアクティビティを考慮しないと、パフォーマンス結果が楽観的になりすぎる場合があります。
- 可能な場合は、Web サーバー層を他の層から切り離し、単純な使用シナリオを実行して、最初に Web サーバー層を検証します。思考時間を考慮しないでスクリプトを実行します。データベース アクティビティを含まないシナリオを実行するようにします。この状態で、Web サーバーのプロセッサを 100% 使用できない場合は、ネットワークの問題が発生しているか、負荷生成クライアントの出力の処理能力が最大限に達している可能性があります。
- データベース シナリオを検証する際は、データの読み取りに限定した単純な使用シナリオを実行します。思考時間を考慮しないでスクリプトを実行します。テスト データ フィードを使用して、データの不規則性のシミュレーションを行います。たとえば、一連の製品をクエリするとします。Web サーバーのプロセッサを 100% 使用できない場合は、ネットワークの問題が発生しているか、負荷生成クライアントの出力の処理能力が最大限に達している可能性があります。
- ビジネス行動のシミュレーションを行う複数のテスト スクリプトを組み合わせて使用し、データベースへの更新や書き込みを行う複雑な使用シナリオを実行して、テスト環境を検証します。

- Web ファーム環境で、負荷テストがインターネット プロトコル (IP) 切り替えを実装しているかどうかを確認します。この確認を行わないと、IP アフィニティが発生することがあります。IP アフィニティが発生すると、負荷生成クライアント コンピュータからのすべての要求が、ファーム内のすべてのサーバー間で均等に配置されるのではなく、1 台のサーバーにルーティングされます。その結果、負荷分散に関係する他のサーバーが使用されなくなるため、負荷テストの結果が不正確になります。
- すべてのサーバーで主要パフォーマンス インジケータ (KPI) を使用して、テスト環境 (プロセッサ、ネットワーク、ディスク、およびメモリ) を評価します。環境を正しく評価するには、クラスタ内のすべてのサーバーを含めます。
- テスト アプリケーション向けのデータ フィードの作成を検討します。たとえば、実際と同様の状況を作り出す、ユーザー数、製品、出荷した商品などの運用データを含むデータベース テーブルを作成し、重要な使用シナリオの問題点を再現できるようにします。多くのシナリオでは、ロック タイムアウトやデッドロックのシミュレーションを行うために、膨大な量のエントリを含むテーブルに対してクエリを実行する必要があります。

その他の考慮事項

パフォーマンス テスト環境のトラブルシューティングを行う際は、次の点を考慮します。

- 負荷のシミュレーションを行う負荷生成クライアントの問題点を探します。クライアント コンピュータでは、プロセッサやメモリ リソースの不足によって、パフォーマンス テストで不正確な結果が生成されることがよくあります。プロセッサ使用率が高くなる原因となる可能性がある処理速度の速いトランザクションを補正するには、クライアント コンピュータを追加することを検討します。また、このトランザクションがボトルネックになる場合は、メモリを増やすことも検討します。テスト データ フィードが負荷生成ツールにキャッシュされたり、負荷テストのスクリプトの複雑さが増したりすると、メモリの使用量が増加する可能性があります。
- 一部のネットワーク インターフェイス カード (NIC) は、自動モードに設定すると、スイッチとのネゴシエーションを全二重モードで正しく実行できません。そのように半二重ネゴシエーションで NIC が動作する場合は、パフォーマンス テストの結果が不正確になります。さまざまな層に Web サーバーとデータベース サーバーを含む代表的な境界ネットワークは、2 枚の NIC を備えた Web サーバーを使用して展開されます。2 枚の NIC のうち 1 枚は、クライアントに接続し、もう 1 枚は別のルートを使用してデータベース層と通信します。しかし、クライアントとデータベース層の両方に接続する Web サーバーに 1 枚しか NIC を使用しない場合は、過密状態によりネットワークがボトルネックにな

る可能性があることに注意してください。

- 運用環境のデータベース サーバーでは、ポリシーの問題として、データベースに関連するログ ファイルとデータ ファイルにそれぞれ個別のハード ドライブを使用することがあります。このような展開構成を再現して、パフォーマンス テストの結果が不正確にならないようにします。DNS を適切に構成しないと、データベース接続を開いたときに、データベース サーバー名を使用してブロードキャスト メッセージが送信されることがある点を考慮します。名前解決の問題が発生すると、接続を開く速度が低下することがあります。
- 間違ったデータ フィードがスクリプトで利用されると、環境に関する問題を見落としてしまうことがよくあります。たとえば、複数のスクリプトがデータベースの同じレコードにクエリすることにより、擬似的なロックがかかり、プロセッサ アクティビティの速度が遅くなることがあります。要求後に送信されるデータの多様性を考慮して、正しいビジネス行動のシミュレーションを行うテスト データ フィードを作成することを検討します。負荷生成ツールは、パフォーマンス テスト データの収集に、ディレクトリ構造のデータベースやファイルなどの中央リポジトリを使用することがあります。このようなデータ リポジトリは、負荷生成ツールが使用するルートにトラフィックが生じないコンピュータ上に配置してください。たとえば、データ収集の管理に使用するコンピュータの仮想ローカル エリア ネットワーク (VLAN) と同じ VLAN にデータ リポジトリを配置します。
- 負荷生成ツールは、負荷を生成するコンピュータと、パフォーマンス データを収集するコンピュータ間に特別なアカウントを使用する必要があることがあります。このような構成が正しく設定されていることを確認します。テスト環境でデータ収集が行われていることを検証します。その際、トラフィックがファイアウォールを通過する必要がある場合を考慮します。

テストの検証

正確さに欠ける負荷シミュレーションは、それ以前のすべての作業を無意味にする可能性があります。テストの実行によって収集されたデータを理解するには、負荷シミュレーションがテストの設計を正確に反映している必要があります。シミュレーションがテストの設計を反映しないと、結果が誤解されやすくなります。テストがテストの設計を正確に反映している場合でも、テストによって無効な結果や誤解を招く結果がもたらされる可能性は数多くあります。テストをたやすく信頼しがちですが、"リリース" の決定をテストの結果に依存する必要があるときは、テストの実行前にテストの正確性を検証することに時間と労力をかける方が常に価値ある結果が得られます。次の 4 つのカテゴリからテストの検証について考えると、有効です。

- **テスト設計の実装 :** テスト設計を正確に実装していることを検証するには、どのような方

法を選択したとしても、テストを実行し、テストの内容を厳密に確認する必要があります。

- **同時実行 :** 1 人のユーザーでテストを実行したときにそのテストがテストの設計と一致することを検証したら、次に複数のユーザーでテストします。各ユーザーが一意のデータをシードし、ユーザーが数秒以内にそれぞれアクティビティを開始する (ただし、同時に開始しない) ようにします。複数のユーザーがアクティビティを同時に開始すると、テスト設計の実装の正確性を検証することが複雑になり、あまり現実に即さないストレスがかかった状況を発生する可能性があります。複数のユーザーでテストが想定どおりに実行されることを検証する 1 つの方法として、テストをそれぞれ 3 人、5 人、および 11 人のユーザーで 3 回実行します。これらの 3 回のテストによって、テスト環境の構成 (アプリケーション コンポーネントにインストールされている制限付きライセンスなど) と、テスト自体 (意図どおりに変化しないパラメータ化されたデータなど) の両方に関する多くの一般的な問題点が見つかる傾向があります。
- **テストの組み合わせ :** 1 人および複数のユーザーでテストが目的どおりに実行されることを検証したら、次の論理手順として、他のテストと組み合わせたときにテストが正しく実行されることを検証します。一般に、パフォーマンスをテストするときは、複数のテストを組み合わせて適合させ、ユーザー、アクティビティ、およびシナリオのさまざまな組み合わせと分布を表すようにします。重要なテスト プロジェクトを実行する前に、テストがこの複雑さに対処するように設計および実装されていることを検証しないと、パフォーマンスに関する有益な情報を収集できるかもしれないときに、テストまたはテスト スクリプトのデバッグで多くの時間を浪費してしまう可能性があります。
- **テスト データの検証 :** テストが正しく実行されていることを確認したら、最後の重要な検証手順として、テスト データを検証します。パフォーマンス テストでは、膨大な量のテスト データが使用される可能性があります。その結果、データセットでエラーが発生する可能性が高くなります。テストで使用されるデータだけでなく、そのデータが意図どおりにテストで共有されること、およびテスト対象のアプリケーションが正しいデータをシードしてテストを有効にすることを検証することが重要です。

動的データ

負荷テスト スクリプトで動的データを正しく使用する技術上の理由を以下に示します。

- システムはメモリ内のコピーからデータを取得するため、同じデータ値を使用すると、キャッシュを擬似的に使用することになります。これは、データベース、オペレーティング システムのファイル キャッシュ、ハード ドライブ、記憶域コントローラ、バッファ マネージャなど、システムのさまざまな層やコンポーネント全体で起こり得ることです。パフォ

ーマンス テスト中にキャッシュのデータが再利用されると、実際よりもテスト結果の方が高速になることがあります。

- ビジネス シナリオによっては、データを比較的狭い範囲から選択しなければならない場合があります。このような場合は、キャッシュを再利用する頻度を上げても、複数のユーザーが同じ項目をクエリすることでタイムアウトが発生することになり、データベースのデッドロックや応答時間の低下など、パフォーマンス関連の他の問題がシミュレーションされます。マーケティング キャンペーンや季節限定販売イベントが、この種類のシナリオの典型的な例です。
- ビジネス シナリオによっては、負荷テスト時に一意のデータを使用しなければならない場合があります。たとえば、特定の一連の資格情報を使用してサイトにログインした後、セッション中にセッション固有の ID がサーバーから返される場合などです。この場合、同じログイン データを再利用すると、無効なセッション ID を示すエラーがサーバーから返されます。別のよくあるシナリオとしては、ユーザーが一連の一意データを入力しないと、システムが選択内容の承認に失敗する場合です。たとえば、登録ページで一意ユーザー ID を入力する必要があるユーザーの新規登録などです。
- ビジネス シナリオによっては、パラメータ化された項目の数を制御しなければならない場合があります。たとえば、キャッシュにさまざまな数の製品が格納されている場合、サーバーの処理能力を評価するためにメモリ占有領域をテストする必要があるキャッシュ コンポーネントなどです。
- ビジネス シナリオによっては、スクリプトのサイズまたはスクリプト数を削減しなければならない場合があります。たとえば、アプリケーションの複数のインスタンスが 1 台のサーバーに存在して、独立系ソフトウェア ベンダ (ISV) がそれらのインスタンスをホストするシナリオを再現する場合などです。このようなシナリオでは、同じビジネス シナリオの負荷テストを実行するときに URL (Uniform Resource Locator) をパラメータ化する必要があります。
- 負荷テストで動的テスト データを使用すると、複雑で時間に依存するバグが再現される傾向があります。たとえば、異なるユーザー アカウントを使用して異なる操作を実行した結果として発生するデッドロックなどです。
- 負荷テストで動的テスト データを使用することで、テスト計画に適していればエラー値を使用できます。たとえば、ハッカーの動作をシミュレーションするテストを行う際、ハッカーは必ず正の数の ID を使用するとします。この場合、無効な値が入力されたときにデータベース テーブルをスキャンするといったアプリケーションのエラーを再現するテストを実行する場合は、0 または負の数を使用することが有効です。

テストの検証

よく使用するテスト検証方法をいくつか以下に示します。これらの方法は、多くの場合、組み合わせて使用します。

- まず、1 人のユーザーのみでテストを実行します。これにより、初期検証の複雑さが大幅に軽減されます。
- 実行中のテストを監視し、普通ではないと感じる動作に十分注意します。直感は正しいことが多く、少なくとも役には立ちます。
- 監視内容と結果データを後から比較できるように、テストの実行中は手動でシステムを使用します。
- テスト結果と収集した指標が、意図した内容を表していることを確認します。
- いずれかの親要求または依存要求が失敗したかどうかをチェックします。
- 負荷生成ツールは、正しいページやデータが返されなかった場合でも、"合格" に見える集計結果をレポートすることがあるため、返されたページの内容を確認します。
- すべてのデータをループするテストを実行し、予期しないエラーをチェックします。
- 該当する場合は、テスト実行後にテスト データやアプリケーション データをリセットできることを検証します。
- テスト実行の最終段階で、テストの設計に応じて、アプリケーション データベースが更新された (または更新されていない) ことを確認します。Web サーバーからコード "200" で成功状態が返される多くのトランザクションは、内部で失敗している可能性があることを考慮します。たとえば、新しいユーザー登録シナリオで以前に使用したユーザー名や、既に使用されている注文番号が原因でエラーが発生することがあります。
- エラーごとにデータベース エントリを消去して、テスト失敗の原因となる可能性があるデータを削除することを検討します。たとえば、その後のテスト実行で再利用できない注文エントリなどです。
- テストを正常に実行するために、さまざまな組み合わせや順序でテストを実行して、別のテストで必要なデータが破損しないことを確認します。

その他の考慮事項

テストを検証する際は、さらに次の点も考慮します。

- 検証テストを実行することで得たパフォーマンスの結果データを、最終レポートの一部として使用しません。
- 検証テスト実行時に明らかになったパフォーマンスの問題点をレポートします。

- 適切な負荷生成ツールを使用し、テスト設計で指定された特性のある負荷を作成します。
- 特定した指標やリソース使用率を対象とするパフォーマンス カウンタが測定中および記録中で、シミュレーションの正確性を損なわないことを確認します。
- パフォーマンス テスト中に他のテストを実行して、シミュレーションがシステムの他の部分に影響しないことを確認します。このような他のテストは、自動化されているか、手動で実行されます。
- テストを繰り返します。その際、ユーザー名や思考時間などの変数を調整して、テストが予想どおりに動作し続けるかどうかを確認します。
- 立ち上げ期間とクールダウン期間のシミュレーションを適切に行うことを忘れないでください。

問いかける質問

- テストの正確性評価に関与させるべき新たなチーム メンバとは。
- 仮の結果に意味があるか。
- 想定したデータがテストに提供されているか。

テストの実行

テストを実行するプロセスと流れは、使用するツール、環境、およびプロジェクトのコンテキストに大きく依存しますが、テストの実行時に留意するかなり普遍的なタスクや考慮事項がいくつかあります。

テスト対象のアプリケーションがパフォーマンス テストの実行に適切な状態であることを確認したら、通常、プロジェクトとアプリケーションの現在状態を基に合理的に完了できる、最も優先順位の高いパフォーマンス テストからテストを開始します。各テストの実行後、テスト中に発生した内容の簡単な概要をまとめ、後からこれらのコメントを参照できるようにテスト ログに追加します。このようなコメントでは、コンピュータのエラー、アプリケーションの例外とエラー、ネットワークの問題、ディスク領域やログの空き領域不足などを扱います。テストの最終実行完了後、すべてのテスト結果とパフォーマンス ログを保存してから、テスト環境を解体するようにします。

できる限り、各タスクを 1 ～ 2 日に制限して、特定のテストや一連のテストの結果が出ないことが判明する場合や、目的の結果を出すために最初のテスト設計を変更する必要がある場合に無駄な時間をなくすようにします。テスト実行時に最も重要なタスクの 1 つは、結果の分析に

よってテストの優先順位を付け直し、テスト、テストの設計、およびその後の方針を変更することを忘れないことです。

次の原則に従うことを強くお勧めします。1 ～ 2 日単位でテスト タスクを実行します。タスクを完了まで確認しますが、新たな価値がもたらされる可能性があれば、回り道をすることも重要です。

効率的かつ効果的にテストを実行するための重要事項

一般に、効率的かつ効果的にテストを実行するには、次の点が重要になります。

- 実行後 2 日以内にパフォーマンス テストの優先順位を見直す。
- パフォーマンス ベースラインを把握し、使用することに留意する。
- アプリケーション エラーの修正、またはテストのデバッグに時間を費やすことを計画する。
- 必要に応じてテスト計画を変更できるよう、すぐに結果を分析する。
- テスト結果を頻繁にチームに伝え、結果をオープンにする。
- 結果や見つかった重要な点を記録する。
- 後でテストを繰り返すために必要なデータを記録する。
- テスト実行中に適切なタイミングで、アプリケーションの最大処理能力または最大ユーザー数までストレスをかける (こうすることで、非常に有益な情報が得られる可能性があります)。
- アプリケーションのチューニングまたは最適化を検証することを忘れない。
- アプリケーションのフェールオーバーと復旧の効果の評価を検討する。
- さまざまなシステム構成の効果を測定することを検討する。

その他の考慮事項

テストを実行する際は、次の点も考慮します。

- 他の業務が中断しないように、分離したネットワーク セグメントでパフォーマンス テストを頻繁に実行します。テスト プロジェクトでこれを実現できない場合は、使用可能なネットワークで特定の時間内に負荷を生成する許可を得るようにします。
- 実際のテストを実行する前に、簡単な "スモーク テスト" を実行し、テスト スクリプトやリモート パフォーマンス カウンタが正しく機能していることを確認することを検討します。
- スモーク テストの実行を選択した場合でも、その結果をテストの公式または正式な結果としてレポートしません。

- シナリオで他に作業を行う場合を除いて、システムをリセットしてから、正式なテストを実行します。
- 可能であれば、各テストを 2 回実行します。生成された結果があまり一致しない場合は、テストを再実行します。差異が生じた要因の判断を試みます。
- 前々からテストのスケジュールが決まっていたとしても、(その日の) テストを開始する 5 分から 30 分前にチームに警告します。1 時間以上連続してテストを実行する予定がない場合はチームに連絡します。
- 負荷を生成しているコンピュータで負荷の生成中にデータの処理、レポートの作成、ダイアグラムの描画などを行いません。このような操作は、データを破損する可能性があります。
- スクリプトのコンパイルなどの理由で、最初の反復テストの結果を放棄しません。この反復テストの結果を個別に測定し、システム規模の再起動が行われた後に最初のユーザーにどのようなことが予想できるかを把握します。
- テストの実行が実際に完了することはありませんが、特定のテストの結果は最終的には収束に向かいます。価値ある情報が収集されなくなったら、テストを変更します。
- テストの実行にかかった 2 倍の時間をかけても開発チームが問題の原因を特定できない場合、1 つ以上の変数や考えられる原因を取り除いてからテストを再実行すると効果が上がることがあります。
- 特定の負荷に関連するパフォーマンスを測定することが目的の場合、負荷を増やすたびにシステムを安定させる時間を設けて、測定値の正確性を保証することが重要です。
- 負荷の生成に使用するクライアント コンピュータ (負荷生成クライアント コンピュータとも呼びます) にストレスをかけすぎないようにします。プロセッサやメモリなどのリソースの使用率を十分低くしておき、負荷生成環境自体がボトルネックにならないようにします。
- すぐに結果を分析し、必要に応じてテスト計画を変更します。
- チームまたはテストとの関連が最も高いサブチームと密接に連携します。
- テスト結果を頻繁にチームに伝え、結果をオープンにします。
- テストを繰り返し実行する場合は、テスト データの復元ポイントを確立してからテストを開始することを検討します。
- ほとんどの場合、テストの実行ごとにメモや監視内容を捕捉するテスト実行ログを管理することは非常に貴重です。
- ワークロードの特徴付けを変化するターゲットとして扱います。思考時間やユーザー数の新しい設定を調整して、通常時とピーク時の負荷状態のユーザーの新しい合計数をモデル

化します。

- 実行中のテストを監視し、通常とは異なると感じる動作に十分注意します。直感は正しいことが多く、少なくとも役には立ちます。
- 特定した指標やリソース使用率を対象とするパフォーマンス カウンタが測定中で、シミュレーションの正確性を損なわないことを確認します。
- 監視内容と結果データを後から比較できるように、テストの実行中は手動でシステムを使用します。
- 立ち上げ期間とクールダウン期間のシミュレーションを適切に行うことを忘れないでください。

問いかける質問 :

- 最近のテスト結果やプロジェクトの更新によって、現在実施できる他のテストより、このタスクの価値が高くなったか、低くなったか。
- このタスクに関与させる新たなチーム メンバとは。
- 仮の結果に意味があるか。

ベースラインとベンチマーク

ベースラインとベンチマークを使用するときは、通常それぞれが、実行する最初と最後のテストになります。プロジェクトの過程で実行するすべてのテストの中で、ベースラインとベンチマークを十分理解および管理し、上記の検証をさらに重要にすることが最も重要です。

ベースライン

ベースラインの作成とは、その後のパフォーマンスを改善するためにシステムやアプリケーションに加える変更の効果を評価する目的で、パフォーマンスの測定データをキャプチャするために、一連のテストを実行するプロセスです。

Web アプリケーションの場合は、ベースラインを使用して、パフォーマンスが向上したか低下したかを判断し、ビルドやバージョン間の差異を見つけます。

たとえば、読み込み時間、一定時間あたりに処理したトランザクション数、一定時間あたりにサービスを提供した Web ページ数、メモリ使用率やプロセッサ使用率などのリソース使用率を計測できます。ベースラインを使用する際には以下のような考慮事項があります。

- システム、コンポーネント、またはアプリケーションに対するベースラインを作成できます。
- データベースや Web サービスなど、さまざまな層でのベースラインを作成できます。
- ベースラインは、将来の最適化や処理能力の低下を追跡する際に比較対象となる標準として使用できます。この目的でベースラインを使用する場合は、ベースラインのテストと結果を十分理解し、再現性を検証することが重要です。
- ベースラインは、傾向分析の出発点として使用でき、開発ライフ サイクルの過程で行う規模縮小や最適化を表す変化を製品チームが明確に示す際に有効です。再利用可能な一連のテスト資産を使用して作成したベースラインが最も価値が高くなります。このようなテストは、再現可能かつ適度の正確さでシュミレーションを行うワークロードの特性を表すことが重要です。
- ベースラインの結果は、応答時間、プロセッサ、メモリ、ディスク、ネットワークなどのパフォーマンスの主要インジケータを組み合わせて使用して、明確に示すことができます。
- チーム内でベースラインの結果を共有すると、パフォーマンスの特性に関する情報の共通基盤が確立され、その後のアプリケーションやコンポーネントのパフォーマンス変化についてコミュニケーションを取ることができるようになります。
- ベースラインをアプリケーション固有にしておくことが、さまざまなビルド、バージョン、またはリリース間でパフォーマンスを比較する際に最も有効です。
- 構成を変更する前にベースラインを確立しておくと、アプリケーションのパフォーマンスに現れる変更の効果をすばやく確認できるため、多くの場合に、時間を節約できます。

ベンチマーク

ベンチマークとは、なんらかの第三者組織が裏付けしている業界標準に対して、システムのパフォーマンスを比較するプロセスです。

Web アプリケーション開発の観点から考えると、ベンチマークでは、アプリケーションのベンチマーク スコアを判断するのに必要なアプリケーションのパフォーマンス測定基準を捕捉するため、業界ベンチマークの仕様に準拠する一連のテストを実行することが必要になります。その後、アプリケーションのスコアを、同じベンチマークで他のシステムやアプリケーションが獲得したスコアと比較できます。ベンチマークの一定のスコアを達成または上回るために、アプリケーションのパフォーマンスのチューニングを選択することもできます。ベンチマークには、以下のような考慮事項があります。

- ベンチマーク スコアは、業界仕様内で作業するか、このような仕様に準拠するように既存

の実装を移植することによって達成します。

- 通常、ベンチマークでは、同時に実行する必要のあるコンポーネント、製品を投入する市場、測定する具体的な評価指標などをすべて特定する必要があります。
- ベンチマーク スコアは公開される可能性があり、競合他社が結果を比較することがあります。ベンチマーク スコアと共に含めることがあるパフォーマンスの評価指標には、応答時間、一定時間あたりに処理したトランザクション数、一定時間あたりにアクセスされた Web ページ数、プロセッサ使用率、メモリ使用率、検索時間などがあります。

テストの保管

テストの実行ごとに行われるスクリプト、シナリオ、データなどの変更を管理し、これらの差異を残りのチーム メンバに通知するため、ある程度の変更管理やバージョン管理が非常に有効です。適用するアプリケーションのビルド時に、テストのスクリプト、結果、およびレポートをバージョン管理システムにチェックインしているチームもあります。また、単に日付別のフォルダに定期的にコピーを保存しているチームや、パフォーマンス チーム専用のバージョン管理ソフトウェアを独自に所有するチームもあります。どのような方法が最も役立つかを判断するのはチーム メンバの役割ですが、テスト、テスト データ、およびテスト結果を保管しておくと、パフォーマンス テスト プロジェクトの過程でかかる時間を大幅に節約できます。

その他の考慮事項

テストを保管する際は、次の点も考慮します。

- 保管したテスト スクリプト、データ、および結果を使用して、製品の次期バージョンのベースラインを作成できます。テストしたソフトウェアのビルドと共にこの情報を保管しておくと、多くの監査可能性標準が満たされます。
- 多くの場合、パフォーマンス テストのスクリプトは新しいビルドで毎回強化または変更されます。スクリプトのコピーを保存し、そのコピーを使用したビルドを特定しておかないと、ビルドをロールバックする際のスクリプトの再実行に多くの追加作業が必要になる可能性があります。
- ほぼすべての負荷生成ツールでは、テストの実装自体は些細なソフトウェア開発作業です。一般に、この作業はチームのソフトウェア開発標準と手順にすべて従う必要はありません。ただし、開発チームが採用するプロセスを補完したり、そのプロセスに匹敵する正しい開発プロセスを採用し、パフォーマンス スクリプト向けに適切に "重み付け" することをお勧めします。

まとめ

パフォーマンス テストの実行には、テストの環境やスクリプトの検証、テストの実行、テスト結果の生成などのアクティビティが関連します。また、パフォーマンス特性のベースラインやベンチマークの作成などもあります。

テスト環境を検証して、テスト環境が運用環境を忠実に表していることを確認することが重要です。

テスト スクリプトを検証して、正しい指標を収集しているかどうか、およびテスト スクリプトの設計がワークロード特性のシミュレーションを正しく行っているかどうかを確認します。

第 VII 部

結果の分析とレポート

内容 :

- **パフォーマンス テストの主要な数学原理**
- **パフォーマンス テスト レポートの基礎**

第 15 章 パフォーマンス テストの主要な数学原理

目的

- 一般的な数学原理や統計原理をパフォーマンス テストの分析とレポートに適用する際、これらの原理の使い方、意味、および基になる概念について学習する。

概要

ソフトウェア開発チームのメンバ、開発者、テスタ、管理者、およびマネージャはすべて、効果的に作業を行うために、数学を当てはめ、統計データを解釈する方法を理解する必要があります。パフォーマンスの分析とレポートでは特に数学を多く利用します。ここでは、パフォーマンス テストで最もよく使用され、誤用や誤解が多い数学的概念と統計概念について、チーム メンバ全員がメリットを得られる方法で説明します。

多くの数学的概念や統計概念を理解する必要がありますが、数学や統計についての経験が豊富だったり、数学や統計を楽しんだりしているソフトウェア開発者、テスタ、およびマネージャはあまりいません。その結果、パフォーマンス テストの結果を大きく間違って表現したり、誤解したりすることになります。ここで紹介している情報が、この分野についての正式なトレーニングの代わりになることは目的としていません。パフォーマンス テストの理解に役立つ数学的手法と統計手法についての共通の言語で、常識的な説明を提供することが目的です。

本章の使い方

ここでは、パフォーマンス データの結果の分析やパフォーマンス結果のレポートの準備に使用するさまざまな指標と計算について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「サンプル データ セット」では、サンプルについて理解します。このサンプルは、ここで説明する主要数学原理を明らかにするために使用します。
- 残りのセクションでは、有意義なパフォーマンス テスト レポートの理解と提示に役立つ主要数学原理を学習します。

サンプル データ セット

ここでは、説明のため次の 3 つのサンプル データ セットを参照しています。

- データ セット A
- データ セット B
- データ セット C

データ セットの概要

データ セット A、B、および C の概要を次に示します。

	サンプル サイズ	最小値	最大値	平均値	中央値	標準値	最頻値	95 パーセンタイル	標準偏差
データ セット A	100	1	7	4	4	4	4	6	1.5
データ セット B	100	1	16	4	1	3	1	16	6.0
データ セット C	100	0	8	4	4	1	3	8	2.6

図 15.1 データ セット A、B、および C の概要

データ セット A

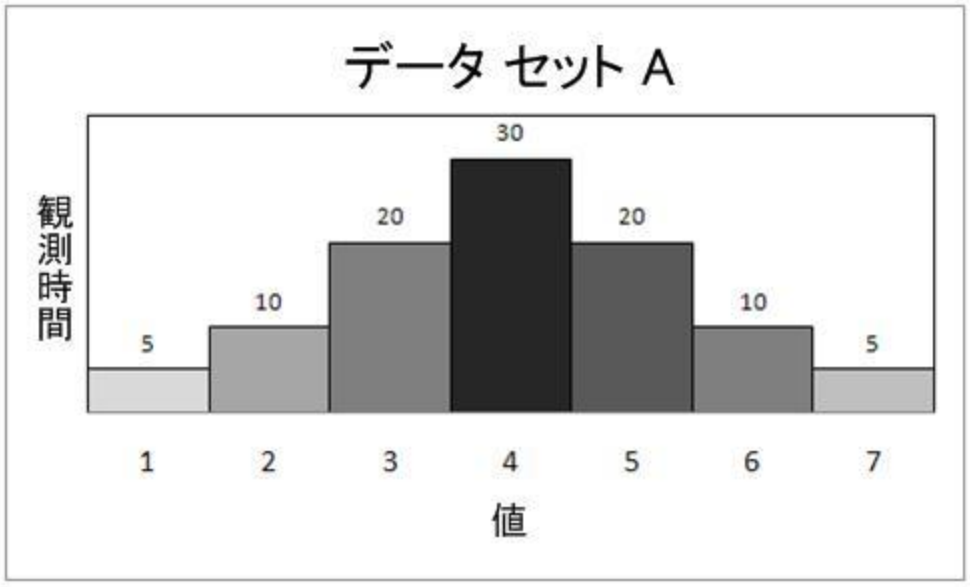


図 15.2 データ セット A

合計 100 個のデータ ポイントの分布は、次のようになります。

- 値が 1 のデータ ポイントが 5 個
- 値が 2 のデータ ポイントが 10 個
- 値が 3 のデータ ポイントが 20 個
- 値が 4 のデータ ポイントが 30 個

- 値が 5 のデータ ポイントが 20 個
- 値が 6 のデータ ポイントが 10 個
- 値が 7 のデータ ポイントが 5 個

データ セット B

図 15.3 データ セット B

合計 100 個のデータ ポイントの分布は、次のようになります。

- 値が 1 のデータ ポイントが 80 個
- 値が 16 のデータ ポイントが 20 個

データ セット C

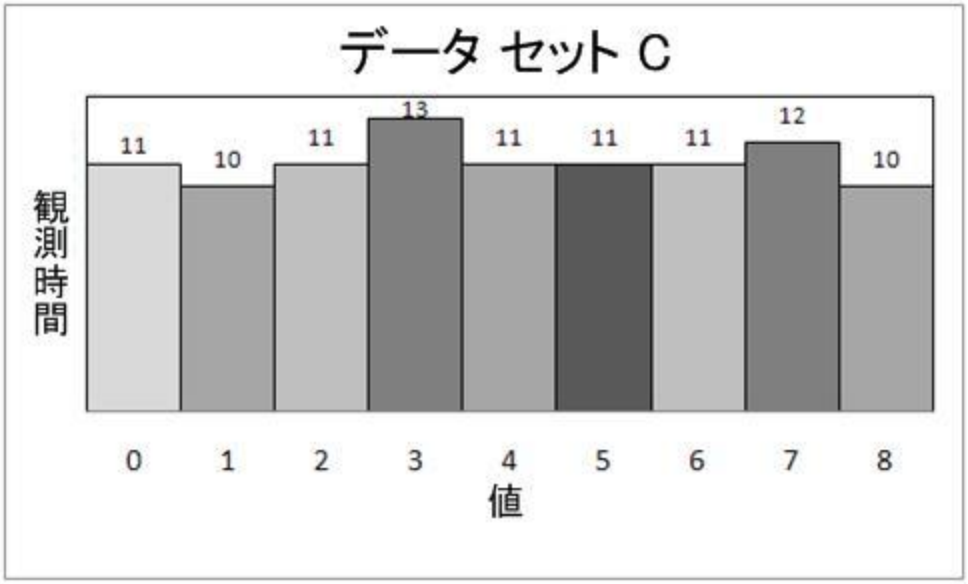


図 15.4 データ セット C

合計 100 個のデータ ポイントの分布は、次のようになります。

- 値が 0 のデータ ポイントが 11 個
- 値が 1 のデータ ポイントが 10 個
- 値が 2 のデータ ポイントが 11 個
- 値が 3 のデータ ポイントが 13 個
- 値が 4 のデータ ポイントが 11 個
- 値が 5 のデータ ポイントが 11 個
- 値が 6 のデータ ポイントが 11 個
- 値が 7 のデータ ポイントが 12 個
- 値が 8 のデータ ポイントが 10 個

平均値

平均値 ("算術平均値" とも呼びます) はすべての統計の中で、おそらく最もよく使用され、最もよく誤解される値です。平均値を算出するには、すべての数値を合計し、それを数値の個数で割るだけです。パフォーマンス テストに関して多くの人が最も混乱する点は、この例の場合、データ セット A、B、および C はすべて平均値がちょうど 4 になっていることでしょう。アプリケーションの応答時間の点から見ると、各データ セットの意味は大きく異なります。応答時間の目標が 5 秒だとすると、各データ セットの平均値だけに注目した場合、3 つのデータ セットすべてが目標を満たしているように見えます。しかし、データに注目すると、構成するすべてのデータが目標を満たしているデータ セットはありません。さらに、データ セット B はおそらくある種のパフォーマンス異常を示しています。平均値を用いて応答時間について議論する際は注意が必要です。可能であれば、レポートする統計情報に平均値だけを使用することは避けてください。平均値をレポートする際は、データ セットのサンプル サイズ、最小値、最大値、および標準偏差を含めてレポートすることをお勧めします。

パーセンタイル

ソフトウェア開発に携わる人の中にパーセンタイルに精通している人はほとんどいません。"パーセンタイル" はわかりやすい概念で、定義よりも例を挙げて説明する方が簡単です。たとえば、ページの応答時間の測定値 100 個で構成されるデータ セットの 95 パーセンタイル値を見つけるには、測定値を最大値から降順に並べ替え、先頭から 6 番目のデータ ポイントを求めるだけです。6 番目のデータ ポイント値がこれらの測定値の 95 パーセンタイル値を表します。応答時間の目的からは、この統計情報を「このシナリオでは、シミュレーションを行ったユーザ

ーの 95% が、"6 番目に遅い値" 以下の応答時間を経験した」と解釈できます。

パーセンタイル統計を単独で使用できるのは、一様分布または正規分布で、外れ値 (下記の「統計上の外れ値」参照) の数が許容範囲に収まっているデータを表す場合のみであることに注意することが重要です。この点を例示するため、サンプル データ セットについて考えます。データ セット B の 95 パーセンタイルは 16 秒です。この値から 5 秒の応答時間目標を達成しているとは思えないことは明らかです。興味深いことに、データ セット B の 80 パーセンタイル値は 1 秒となり、この値は誤解を招く可能性があります。応答時間目標が 5 秒の場合、16 秒の応答時間が存在することはおそらく受け入れられないため、この場合、これらのパーセンタイル値はいずれも応答時間の概要を表すのに有効なデータではありません。

データ セット A は正規分布のデータ セットで、95 パーセンタイル値は 6 秒、85 パーセンタイル値は 5 秒、最大値は 7 秒です。この場合、85 パーセンタイル値または 95 パーセンタイル値をレポートすることは、関係者がおそらくデータについて的確に想定できる方法でデータを表していると言えます。

中央値

"中央値" とは、単に、データ セットを最小値から最大値まで並べたときの中央の値です。データ ポイントの個数が偶数で 2 つの中央値が等しくない場合、中央の 2 つのデータ ポイントの平均値を中央値とすることを推奨する規則と、データ セット全体の平均値に近い方の値を中央値とすることを推奨する規則があります。サンプル データ セットの場合、データ セット A とデータ セット B の中央値は 4 で、データ セット C の中央値は 1 です。

標準値

"標準値" とは、データ セットの中で最も頻繁に現れる単一の値です。データ セット A の標準値は 4、データ セット B の標準値は 3、データ セット C の標準値は 1 です。

標準偏差

定義では、1 つの "標準偏差" とは、一連の測定値の中での分散の量で、データ セットのすべての測定値のうち、出現頻度が高い方から約 68% が含まれます。つまり、データ セットの標準偏差を知ると、データ ポイントが平均値の周囲にどれ程度密集しているかがわかります。簡単に言えば、標準偏差が小さいほど、データの一貫性は高くなります。サンプルでは、データ セ

ット A の標準偏差は約 1.5、データ セット B の標準偏差は約 6.0、データ セット C の標準偏差は約 2.6 です。

この場合の一般的規則は、「標準偏差が平均値の半分を超えているデータは疑わしいデータとして扱うべきで、データが正確であれば、データは正規分布のパターンを示していない」ということになります。この規則を当てはめると、データ セット A は正規分布の妥当な例と考えられます。データ セット B は正規分布を表しているとも、表していないとも言えません。データ セット C は明らかに正規分布を表していません。

一様分布

一様分布 ("直線分布" とも呼びます) は、上限値と下限値の間で均等に配置されている一連の乱数とほぼ同じデータのコレクションを表します。一様分布では、データ セットの各数値が出現する回数はほぼ同じになります。一様分布は、ユーザーによる遅延をモデル化するときによく使用されますが、応答時間の結果データにはあまり見受けられません。実際には、応答時間データの結果が一様に分布している場合は疑わしい結果を示していることがあります。

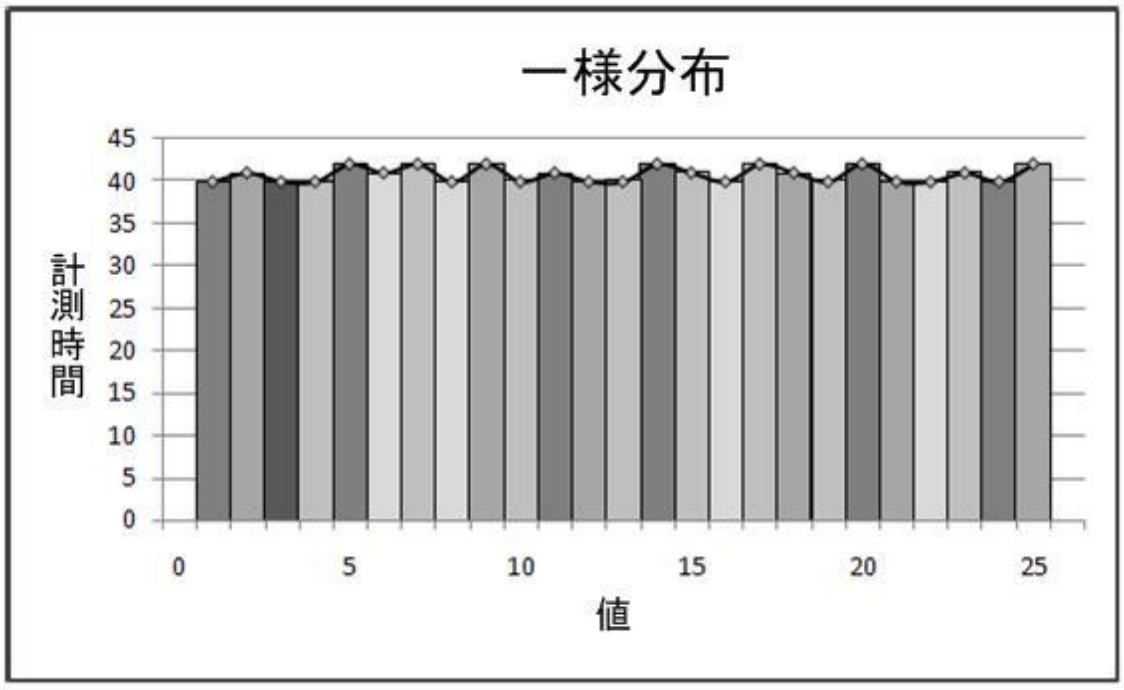


図 15.5 一様分布

正規分布

"釣鐘曲線" とも呼ぶ "正規分布" は、構成データが中央 (中央値) に偏っているデータ セットです。グラフにすると、データ セットの標準偏差により、正規分布のデータの "釣鐘" の形状は高くて細い形から低くてずんぐりした形まで変化します。標準偏差が小さいほど、高くて細い "釣鐘" になります。統計的には、人的変化の測定値は、ほとんどが正規分布を示すデータ セットになります。つまり、多くの場合、Web アプリケーションのエンド ユーザーの応答時間も正規分布になります。

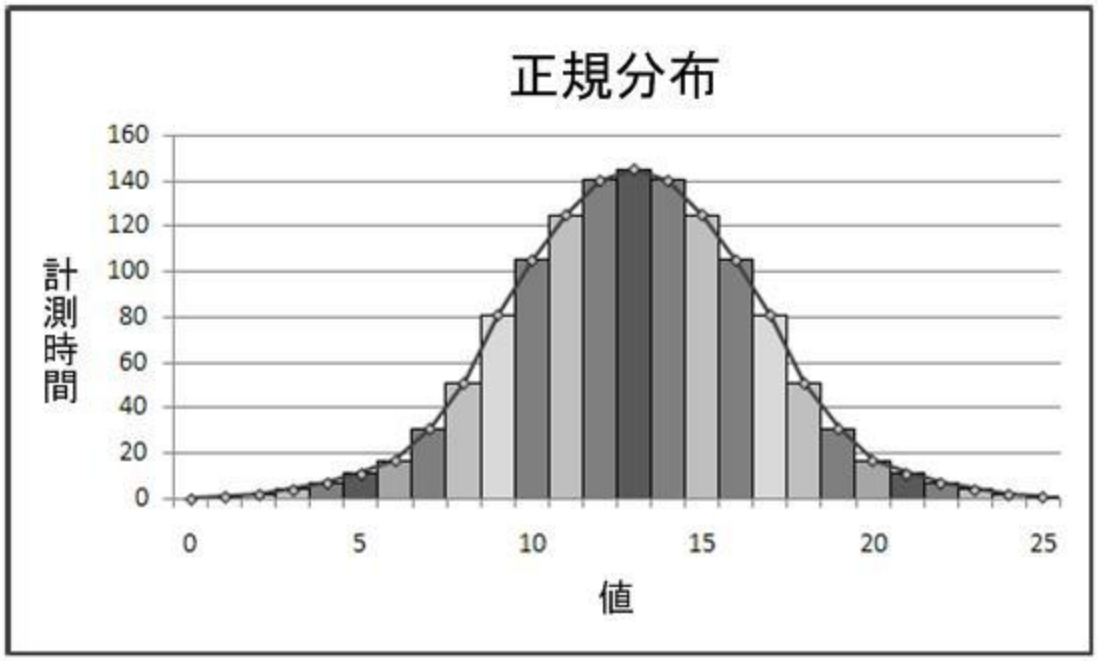


図 15.6 正規分布

統計上の有意性

統計上の有意性 (信頼性) をサンプル サイズに基づいて数学的に算出することは、商業目的のソフトウェア開発プロジェクトにとってはあまりに難しく、複雑な作業です。さいわい、統計上の有意性に関する最も大きな懸案事項を特定できるだけの、効果的かつ正確な常識的アプローチがあります。統計上の有意性を数学的に厳密に計算するもっともな理由がない限り、通常、常識的な推定で十分です。常識的アプローチの裏づけとして、このトピックに関するスタットソフト社 (http://www.statsoft.co.jp) の考察からの抜粋を参考にしてください。

> 実際に "有意" なデータとして扱う有意水準についての最終的判断で、恣意性を避ける方法はありません。つまり、無効な結果として却下しようとして有意水準を選択することは、恣意的です。

通常、パフォーマンス テストに反復処理を追加して、収集する測定値の総数を増やすことは非常に簡単です。統計上の有意性を確保する最適な方法は、収集したデータが現実を表しているかどうか疑わしい場合に、単純にさらに多くのデータを収集することです。できる限り、少なくとも 2 回の独立したテストから、少なくとも 100 個の測定値のサンプル サイズを得るようにしてください。

大量のデータを必要とする複雑な方程式を使わずに、結果が統計的に同じであると判断する方法については厳密な規則は存在せず、このようなデータを収集する時間やリソースも、商業目的のソフトウェア プロジェクトにはめったにありません。それでも、類似するデータになると予測していた 2 つのテスト実行結果を評価した後で、データの有意性や信頼性に疑問がある場合、次のアプローチを適用することは妥当です。少なくとも 5 回分のテスト実行結果を比較し、以下の経験則を当てはめて、信頼できると考えられるほど結果に類似性があるかどうかを判断します。

1. テスト実行結果の 20% (つまり、1/5) 以上が他のテスト結果と異なる場合、一般に、テスト環境、アプリケーション、またはテスト自体に不備があります。
2. すべてのテスト実行結果の 90 パーセンタイル値が、その他のテスト実行結果のいずれかの最大値より大きいか最小値より小さい場合、そのデータ セットはおそらく統計的に同じではありません。
3. グラフを並べてみると、あるテストの測定値が他のテスト実行結果より大幅に高いか低い場合、その測定値はおそらく統計的に同じではありません。

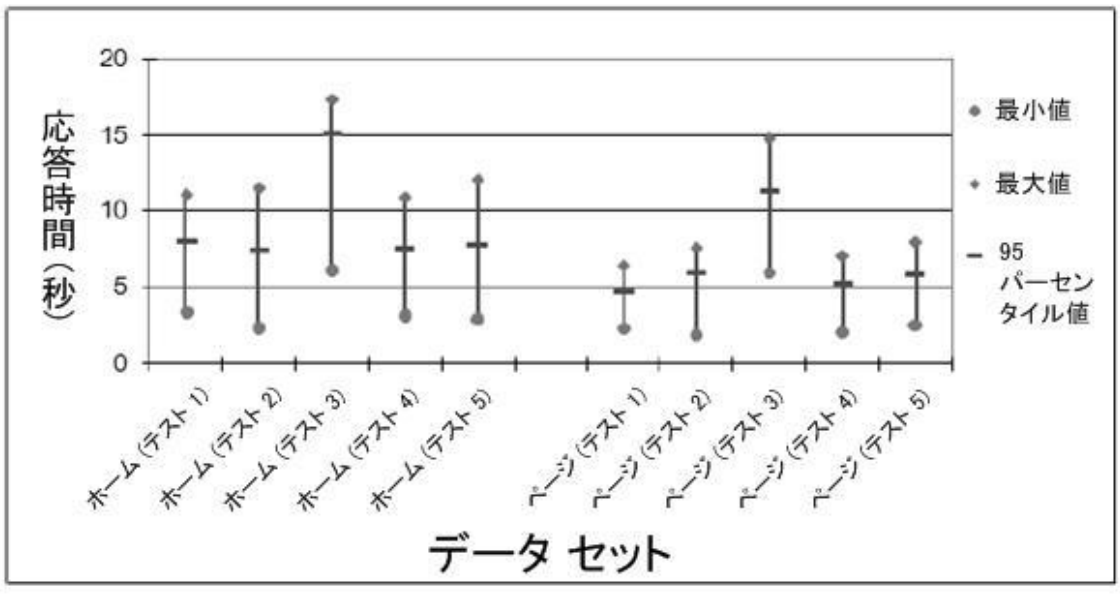


図 15.7 結果の比較

4. テストの特定項目に対するあるデータ セット (1 ページの応答時間など) が大幅に高いか低い場合でも、残りの項目のデータ セットに対する結果が同じように見える場合、テスト自体はおそらく統計的に同じです (ただし、おそらく 1 つのデータ セットだけが異なる理由を時間をかけて調査する価値があります)。

統計上の等価性

統計上の有意性を実際に判断する上記の手法では、統計上の等価性の原則を当てはめています。基本的に、上記で説明した統計上の有意性を判断する手法は、「複数のテスト結果データが等しくなることを想定しているとすると、これらのうちいずれかのテストのデータが、等しくなると想定したすべてのテストの 80% 以上と統計的に等しい場合、そのデータを統計上有意なデータとして扱うことができる」と言い換えることができます。カイ 2 乗、t 検定などの公式手法を使って等価性を数学的に決定することは、商業目的のソフトウェア開発プロジェクトでは一般的ではありません。むしろ、一般的には、統計上の有意性の判断と同様に、グラフを使って等価性を推定してもかまわないと考えられています。

統計上の外れ値

純粋に統計上の観点からは、収集した全測定値の 3 つの標準偏差 (99%) に含まれない測定値はすべて、"外れ値" とみなされます。この定義の問題は、収集した測定値が統計上有意で、しかも正規分布になっていると想定していることです。パフォーマンス テスト データを評価する際、自動的にこのような状態になることはまったくありません。

これを説明するため、スタットソフト社 (http://www.statsoft.co.jp) による、外れ値のより適切な定義を以下に示します。

> 外れ値とは、変則的で珍しい観察結果、つまり残りのサンプルの分布に従っていないように見えるデータ ポイントです。外れ値は一貫性のある中の特異な性質を表すこともあれば、モデル化すべきではない測定値のエラーなどの異常結果を示すこともあります。

この (その他の) 外れ値の説明は、測定値の統計上有意なサンプルと考えられるデータにのみ当てはまることに注意してください。統計上有意なサンプルがなければ、外れ値と代表的な測定値の違いを判断する、一般的に許容できるアプローチはありません。

この説明から、外れ値、つまり適切ではないように思える特異なデータ ポイントの徴候を判断するために結果のグラフを使用できます。明らかな外れ値が本当に変則的で特異な状態かどうかを確認する最適な方法は、テストを再実行してその結果を最初の結果セットと比較することです。外れ値と考えられる値を除いて測定値の大部分が同じ場合、この結果にはおそらく無視できる本当の外れ値が含まれています。しかし、外れ値と考えられる値が同じように結果に現れる場合、この値はおそらく検討する価値のある有効な測定値です。

データ セットに外れ値が含まれていることを特定したら、次の疑問点は、いくつの外れ値を「変則的で特異な観察結果」として却下できるかという点です。

外れ値を無条件に却下できる数に決まりはなく、観察結果の総数に対する最大比率で判断します。上記 2 つの定義の考え方を当てはめると、平均値から 3 つの標準偏差の外側にあり、特定の測定におけるすべての値の 1% までの値が、外れ値と考えられるほど大幅に変則的で特異な場合というのが妥当な結論になります。

以上をまとめると、商業目的のソフトウェア開発を行う場合、特定の項目に対するすべての測定値の 1% 未満で、平均値から少なくとも 3 つの標準偏差より離れている値は、以前のテストや以降のテストに同じ値がないとき (このときのみ)、結果の分析で無視できる値の候補になります。同じ概念をもっとかみ砕いて表現すると、すぐに説明がつかず、結果のごくわずかな部分しか占めない、他のテストの結果と一致しないような、明らかに特異で変わったデータ ポイントは、おそらく外れ値と言えます。

ただし、データ ポイントを外れ値と特定し、結果の集約から除外しても、そのデータ ポイントを無視することにはなりません。除外した外れ値は、その後のテストを実施する中で、懸念するパターンがあらゆる点から考えて各テストの外れ値と言える値の中に特定されるかどうか判断するため、プロジェクトの状況に応じた方法で追跡していく必要があります。

信頼区間

データの信頼レベルを判断することは、統計上の有意性や外れ値の存在を判断することよりもさらに複雑で時間のかかる作業になるため、商業目的のソフトウェア プロジェクトでは非常にまれなことです。特定の統計値の信頼区間とは、ある程度の確実性で "正しい" 統計値が存在する、その統計値近辺の値の範囲です。

関係者の多くは、「これらの結果の信頼区間どうなっている」などと、テスト結果の正確さを推定

するよう求めるため、ここでも常識的なアプローチを採用する必要があります。

パフォーマンス テストを行う場合、この質問の答えはテストするモデルの正確さに直接関係します。多くの場合、ソフトウェアが運用環境にリリースされるまでモデルの正確さを適切に判断することはできないため、これはあまり役立つ依存関係ではありません。しかし、結果の信頼区間を示す方法はあります。

収集した測定値から見て "最良"、"最悪"、"予想される範囲内" とチームが判断できるシナリオなど、さまざまなシナリオをテストすることで、次のように信頼区間を表すグラフを作成できます。

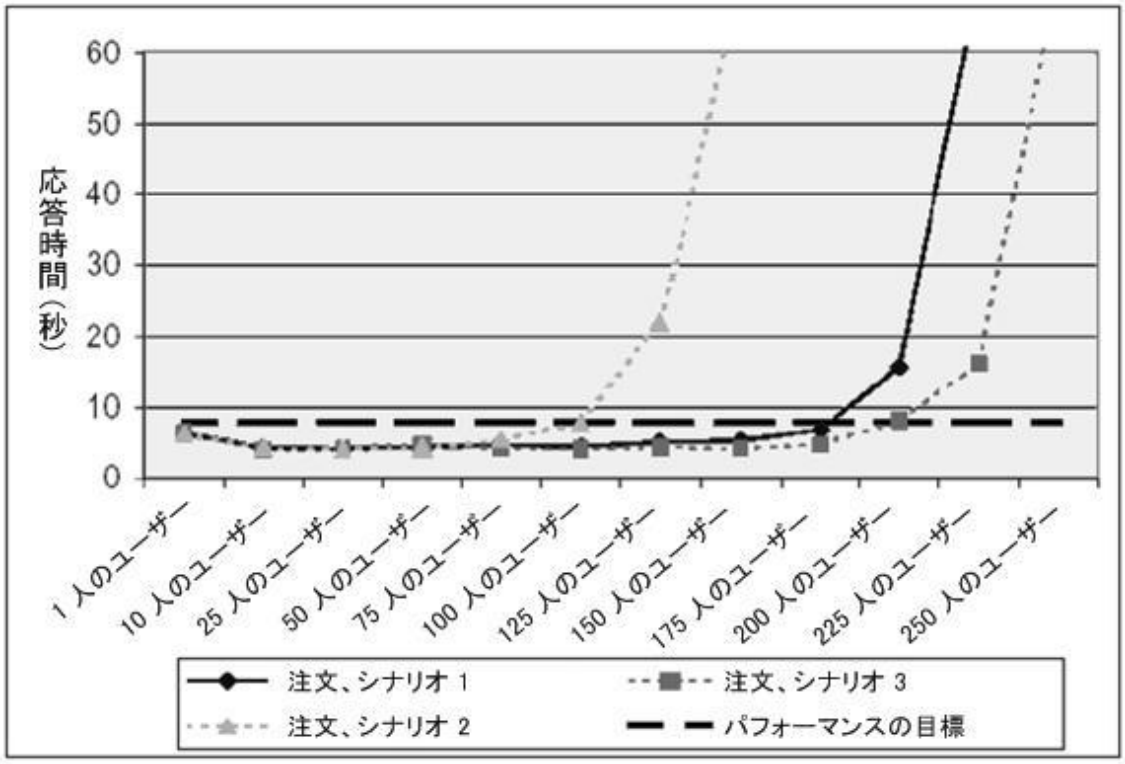


図 15.8 使用法モデル

このグラフでは、破線がパフォーマンスの目標を表し、3 本の曲線が最悪 (パフォーマンスに与える影響が最大)、最良 (パフォーマンスに与える影響が最小)、および予想される範囲内のユーザー コミュニティ モデルを表しています。予想どおり、予想される範囲内の実線の曲線は、最良の場合の曲線と最悪の場合の曲線の間に位置しています。これらの曲線が目標の破線と交差している箇所を観察すると、各場合に指定されたパフォーマンス目標を満たしたままシステムにアクセスできるユーザー数がわかります。チームが最良と最悪の場合のユーザー コミュニティ モ

デルが本当に最良と最悪の場合だと (独自の推定で) 95% の確信がある場合、このグラフから次のように読み取ることができます。「テストでは、満足できるパフォーマンスのまま 100 ～ 200 人のユーザーがシステムにアクセスできる信頼度は 95% であることを示している。」

100 ～ 200 人のユーザーという信頼区間は極めて大きいと思えますが、運用環境での実際の使用経験を表すデータがないと、結果を生成するモデルの信頼度よりも高い信頼度をその結果に期待するのは現実的ではないことに注意することが重要です。テスト結果が正確にテスト対象のモデルを表していることを 100% 確信できれば最高です。

まとめ

ソフトウェア開発チームのメンバ、開発者、テスタ、管理者、およびマネージャはすべて、効果的に作業を行うために、数学を当てはめ、統計データを解釈する方法を理解する必要があります。パフォーマンスの分析とレポートでは特に数学を多く利用します。パフォーマンス テストを正しく分析およびレポートできるように、パフォーマンス テストの数学原理や統計原理を理解することは重要です。

第 16 章 パフォーマンス テスト レポートの基礎

目的

- レポートの効果を高める原則をパフォーマンス テスト データに当てはめる方法について学習する。
- 技術的な結果を共有するタイミングと関係者向けのレポートを作成するタイミングの関係について学習する。
- さまざまなチーム メンバが、パフォーマンス レポートから答えを得ることを期待する質問について学習する。

概要

上司や関係者は、さまざまなテストからの結果だけでは満足しません。これらの結果に基づく結論と、このような結論に至った背景にある整理されたデータを要求します。技術チームのメンバも、単なる結果以上のもの、つまり結果を取得した方法の分析、比較、および詳細を要求します。あらゆる種類のチーム メンバがパフォーマンス結果を共有する頻度を高めることに価値があります。ここでは、さまざまなレポート技法と結果を共有する技法を利用し、各技法が高く評価される模範シナリオを学習することによって、パフォーマンス テストの結果とデータの利用者全員のニーズを満たす方法について学習します。

本章の使い方

ここでは、パフォーマンス テスト結果のレポートの効果を高める原則について理解します。また、効果的なデータ プレゼンテーションの模範として参考にすることもできます。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「レポートの効果を高める原則」では、効果的なレポートを支える主な概念と原則について理解します。
- 「レポートされる頻度の高いパフォーマンス データ」では、パフォーマンス データを表示するさまざまな方法、およびそれらの方法が最も効果的に活用される結果の種類について学習します。
- 「レポートによって解決される質問」では、レポートがさまざまなユーザー向けてどのように設計されるか、およびユーザーが直感的に理解できる形式で、適切なユーザーに適切な情報を提供する方法について学習します。

レポートの効果を高める原則

レポートの効果を高めるには、対象とするユーザーが関心のある情報を、迅速、簡単、かつ直感的な方法で提示することが重要です。レポートの効果を高めるうえで、基盤となる原則の一部を以下に示します。

- レポートを迅速かつ頻繁に提供する
- 見やすいレポートを提供する
- 直感的なレポートを提供する
- 適切な統計を使用する
- データを正しく集約する
- データを効果的にまとめる
- 対象ユーザー向けにレポートをカスタマイズする
- 口頭での簡単なまとめを使用する
- データを利用可能にする

レポートを迅速かつ頻繁に提供する

パフォーマンス テスト プロジェクトの効果を高め、プロジェクト全体を成功させるには、情報とデータを絶えず共有することが重要です。ただし、共有するすべての情報とデータを、正式または準正式なレポートの形式にする必要はありません。効果的な 1 つのアプローチとして、毎日または 2 日に 1 回、要点を簡潔にまとめた電子メール メッセージで関係者に概要のグラフや表を送信する方法があります。次回の正式または準正式なレポートに含める内容を決定する際には、関係者から受け取ったフィードバックや質問を使用します。このようにすれば、単独のドキュメントまたは最終ドキュメントを作成する前に、対象ユーザーのニーズを判断できます。

技術チームと情報やデータを共有するプロセスは、さらに簡単なプロセスです。このプロセスは、新しい結果ファイルの分析を開始する前に、それらのファイルの場所をチームの Wiki に投稿して、分析によって作成されるすべての図とグラフへのリンクを投稿する程度です。

見やすいレポートを提供する

ほとんどの人が、データや統計を図で表す方が理解しやすいと感じます。パフォーマンスの結果データではこのことが特に顕著です。パフォーマンスの結果データの場合、データ量が非常に多くなりがちで、最も重要な結果を検索する場合データのパターンを調べることになります。このようなパターンは、表全体をスキャンしたり、複雑な数学アルゴリズムを使用したりして検索で

きますが、人間の目で見る方がはるかに速く、正確である場合がほとんどです。

パターン、つまり "関心のあるポイント" を目で確認したら、通常、残りの "図内の不要パターン" を削除して、関心のあるパターンを分離します。このような状況では、図内の不要パターンには、関心のないポイントを含む (つまり、関心のないポイントを含むように見える) アクティビティとタイム スライスを表すすべてのデータ ポイントが含まれます。図内の不要パターンを削除すると、関心のあるパターンをより明確に評価でき、レポートがより明瞭になります。

直感的なレポートを提供する

正式かどうかに関係なく、レポートは単独で使用できるものにします。レポートの読者が、その情報がなぜ重要か疑問に思った場合、そのレポートは失敗です。レポートから問題に対する効果的な解答が得られるとは限りませんが、プレゼンテーションによって迅速かつ直感的に明らかになる必要があります。

レポートの直感性を検証する 1 つの方法として、図とグラフからすべてのラベルや ID を削除し、プレゼンテーションからすべての識別情報を削除して、プロジェクトに詳しくない人にレポートを提供してみる方法があります。レポートを見た人が、図やグラフの懸案事項を迅速かつ正確に指摘できる場合、またはレポートの説明で話した懸案事項が問題となる理由を特定できる場合は、直感的なレポートを作成したことになります。

適切な統計を使用する

統計について多くの概念を理解しておくことは幅広いニーズの 1 つですが、統計について経験が豊富だったり、統計を駆使しているソフトウェア開発者、テスタ、マネージャはあまりいません。その結果、レポートを作成する際に、パフォーマンス テストの結果を大きく間違って伝えてしまうことがあります。特定の問題の注目を集めるのにどの統計を使用すればよいかわからない場合は、遠慮しないで支援を仰ぎます。

データを正しく集約する

必ずしも結果を集約することはありませんが、結果が多くのグラフに分散することは避け、1 つか 2 つのグラフに集約すると、結果パターンの説明がはるかに容易になります。ただし、パフォーマンス レポート出力を表やグラフにまとめることができるのは、統計上類似する同一のテストを実行した結果のみである点に留意してください。

その他の考慮事項

結果を集約するには、テストとテスト環境が同一で、かつテスト結果が統計上同一でなければなりません。結果が集約できるほど十分に類似しているかどうかを判断する 1 つのアプローチとして、少なくとも 5 回分のテスト実行結果を比較し、次の規則を当てはめます。

- テスト実行結果の 20% (つまり、1/5) 以上が残りのテスト結果と異なる場合、一般に、テスト環境、アプリケーション、またはテスト自体に不備があります。
- すべてのテスト実行結果の 95 パーセンタイル値が、その他のテスト実行結果のいずれかの最大値より大きいか最小値より小さい場合、その結果は統計的に同じではありません。
- テスト実行の各ページまたは各タイマの結果が、残りすべてのテスト実行結果より大幅に高いか低いことがグラフに示される場合、その結果は統計的に同じではありません。
- テスト実行の 1 ページまたは 1 つのタイマの結果が、残りすべてのテスト実行結果より大幅に高いか低い場合でも、そのテスト実行の残りすべてのページまたはタイマの結果が同じようにグラフに示される場合、それらのテスト実行はおそらく統計的に同じです。

データを効果的にまとめる

多くの場合、結果をまとめることで、テスト結果に含まれる重要なパターンを示すことが大幅に容易になります。データをまとめたグラフや表は、さまざまなテストの実行結果データを並べて示すので、傾向やパターンの特定が容易です。総合的に考えると、このようなグラフや表を使っ

てテスト結果とシステムのパフォーマンス目標とを比較することで、チーム メンバは、システムのリリース、システムのアップグレード、場合によってはプロジェクトの完全な再評価に関する重要な決定を下すことができます。

その他の考慮事項

テスト データをまとめる際は、次の点に留意します。

- 結果が明確にわかるグラフや表を使用します。
- 表やグラフを補足するテキストを使用します。テキストを補足するために表やグラフを使用するのではありません。
- 読者の混乱を招くグラフや表は使用しません。

対象ユーザー向けにレポートをカスタマイズする

パフォーマンス テストの結果を最もよく参照するのは、技術チームのメンバ、技術チーム以外のメンバ、および主要チームに属さない関係者という、3 つのグループのいずれかのユーザーです。これらの 3 つのグループは、パフォーマンス レポートにまったく異なる値を期待し、異なるプレゼンテーション方法を求めがちです。レポートを作成する際は、レポートが対象とするグループ、およびそれらのグループの期待値を見極めてから、収集した結果の最も適切な提示方法を決定してください。

口頭での簡単なまとめを使用する

結果には、口頭での簡単なまとめを少なくとも 1 つ付けるべきですが、最も適切または最も容易に示す方法が記述しかない結果もあります。こうしたまとめに何を含めるかは、対象とするユーザーによって異なります。一部のユーザーは、グラフで示そうとしている要点を押さえた 1 ～ 2 文しか必要としないことがあります。その例を次に示します。

「このグラフを見ると、テスト対象のシステムは、1 時間あたり 150 人のユーザーまで、指定されたすべてのパフォーマンス目標を満たしますが、150 人に達した時点で基本的に使用不可の状態までパフォーマンスが急速に下降することがわかります。」

また、提示されるグラフの細かい説明を求めるユーザーもいます。その例を次に示します。

「このグラフでは、平均応答時間を秒単位で示します。平均応答時間はグラフ左側に縦書きされ、グラフ下部に沿って横書きされた、各テストの実行中にシミュレーションされた 1 時間あたり

のユーザーの合計数に対して曲線で表されています。グラフが交差する位置は...」

データを利用可能にする

パフォーマンス テスト (または他のテスト) のデータは、ユーザーが不適切に使用したり分析したりする恐れがあるので、未加工のまま共有してはいけないと一般に信じられています。この考え方は間違ってはいませんが、それ以上に大きな懸念があります。それは、1 人のユーザーまたは 1 つのチームが、ある特定の時点に一連のデータからすべての値を取得できることを期待するのはまったく不当だという点です。データは、さまざまなユーザーにさまざまな値をさまざまなタイミングで提供するため、チーム内部で常にデータを利用できるようにすることが、データを最大限に活用する唯一の方法です。また、データを利用できるようにしておくと、パフォーマンスの結果が、ユーザー自身が知らない一連のツールとプロセスに基づいて作成された単なる製作物だと考えられてしまうことを最小限に抑えることができます。

レポートされる頻度の高いパフォーマンス データ

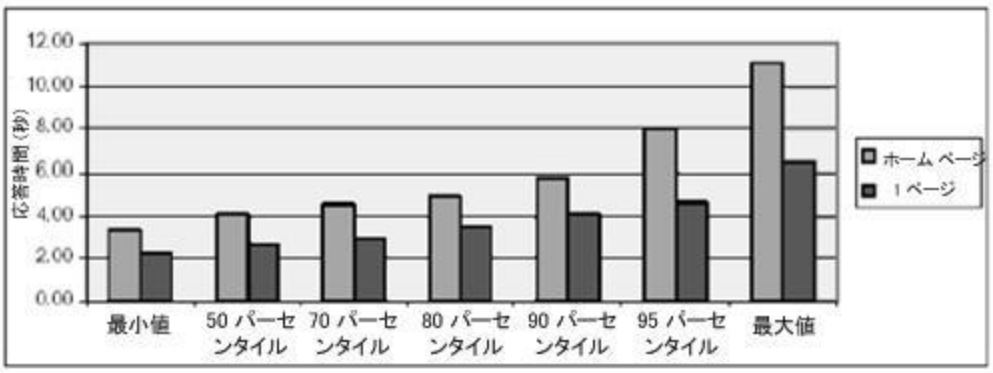
以下に、最もレポートされる頻度の高い結果データの種類を示します。これ以降では、このようなデータが関心を持たれる理由、関心を持つユーザー、およびその種のデータをレポートする際の考慮事項について説明します。

- エンド ユーザーの応答時間
- リソース使用率
- 量、処理能力、および速度
- コンポーネントの応答時間
- 傾向

エンド ユーザーの応答時間

エンド ユーザーの応答時間は、パフォーマンス テストで要求およびレポートされることが圧倒的に多い指標です。事実上、目標と要件を把握していれば、エンド ユーザーの応答時間がシステムまたはアプリケーションのパフォーマンス特性に対するユーザーの満足度の基準になります。関係者がユーザーがアプリケーションにどの程度満足しているかを判断するときは、エンドユーザーの応答時間に興味を持ちます。技術チームのメンバは、ユーザーの観点からパフォーマンスの全体目標を達成しているかどうかを把握し、達成していない場合はその分野を把握する必要があるため、応答時間に関心を持ちます。

サンプル グラフ 1



CmdID	個数	平均値	標準偏差	最小値	パーセンタイル					最大値
					50	70	80	90	95	
ホーム ページ	99	4.53	1.47	3.33	4.08	4.75	4.87	5.77	7.99	11.05
1 ページ	100	2.94	0.85	2.26	2.59	2.91	3.48	4.08	4.65	6.44

図 16.1 応答時間

サンプル グラフ 2

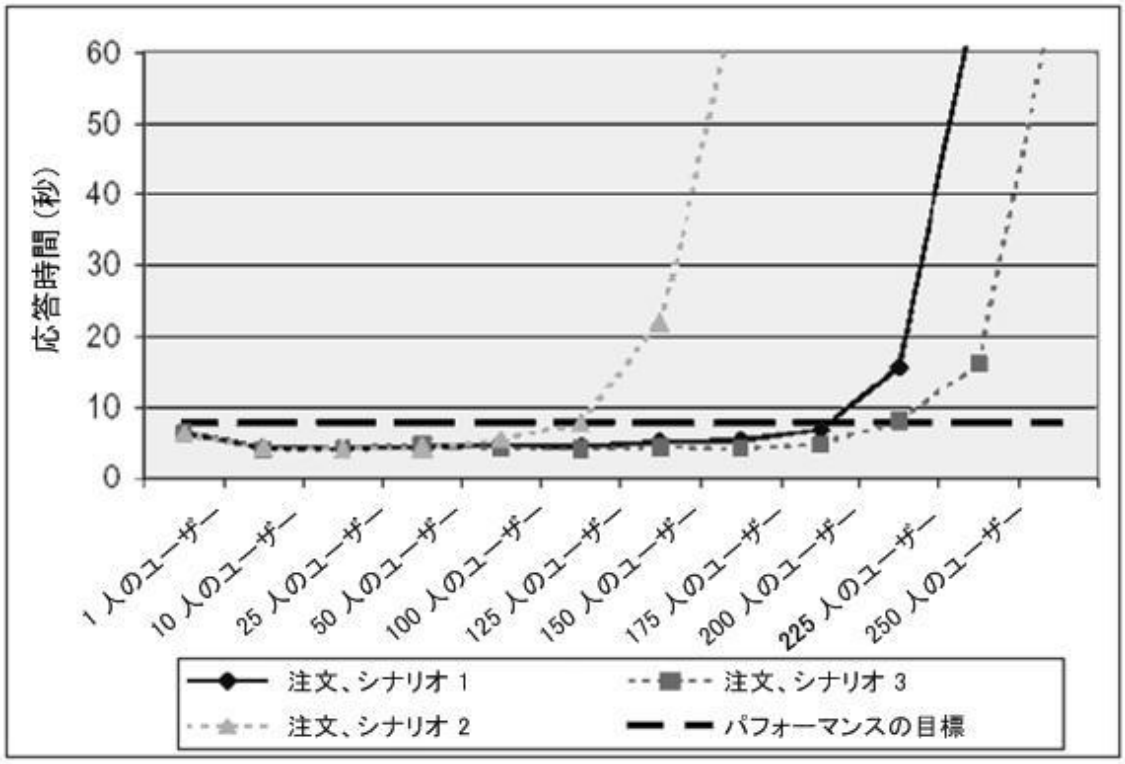


図 16.2 応答時間の低下

考慮事項

エンド ユーザーの応答時間は最もよくレポートされるパフォーマンス テストの指標ですが、考慮すべき重要な点は他にもいくつかあります。

- **レポート前に外れ値を取り除く :** 正当な外れ値であっても、結果データを大幅に歪曲する可能性があります。
- **統計を明確に伝える :** たとえば、平均値と 90 パーセンタイル値の違いが、システムやアプリケーションの "リリース" と "修正" の違いになってしまう可能性があります。
- **放棄を個別にレポートする :** ユーザーによる放棄を考慮に入れている場合、放棄されたページについて収集された応答時間は、放棄されなかったページの応答時間とは同じアクティビティを表さないことがあります。念のため、エンド ユーザーの応答時間のグラフを添えて、放棄されなかったページの応答時間をレポートし、別のグラフや表でページごとの応答時間と放棄の割合をレポートします。
- **各ページまたはトランザクションを個別にレポートする :** あるページが同値類を表すように見えても、気付かない違いがある可能性もあります。

リソース使用率

リソース使用率は、パフォーマンス テストで 2 番目に多く要求およびレポートされる指標です。リソース使用率の指標は、口頭またはナレーション形式でレポートされることがほとんどです。たとえば、「アプリケーション サーバーの CPU 使用率は 45% を超えることはありません。目標は 70% 未満を保つことです」などとレポートされます。伝達方法に問題があるときは、一般に、リソース使用率をグラフでレポートすると有効です。

関係者向けのサンプル グラフ

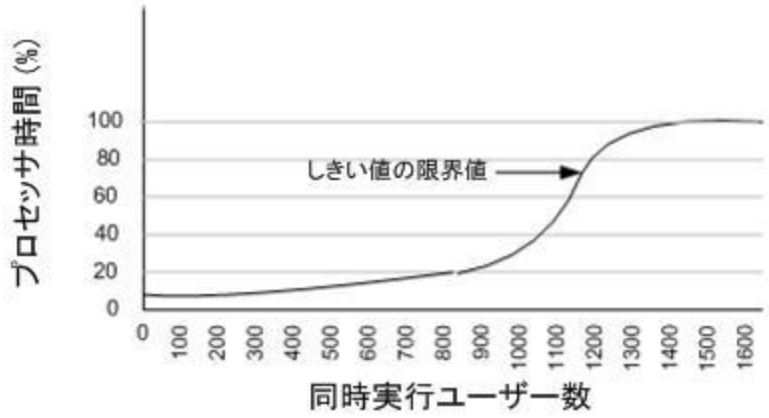


図 16.3 プロセッサ時間

技術チームのメンバ向けのサンプル グラフ

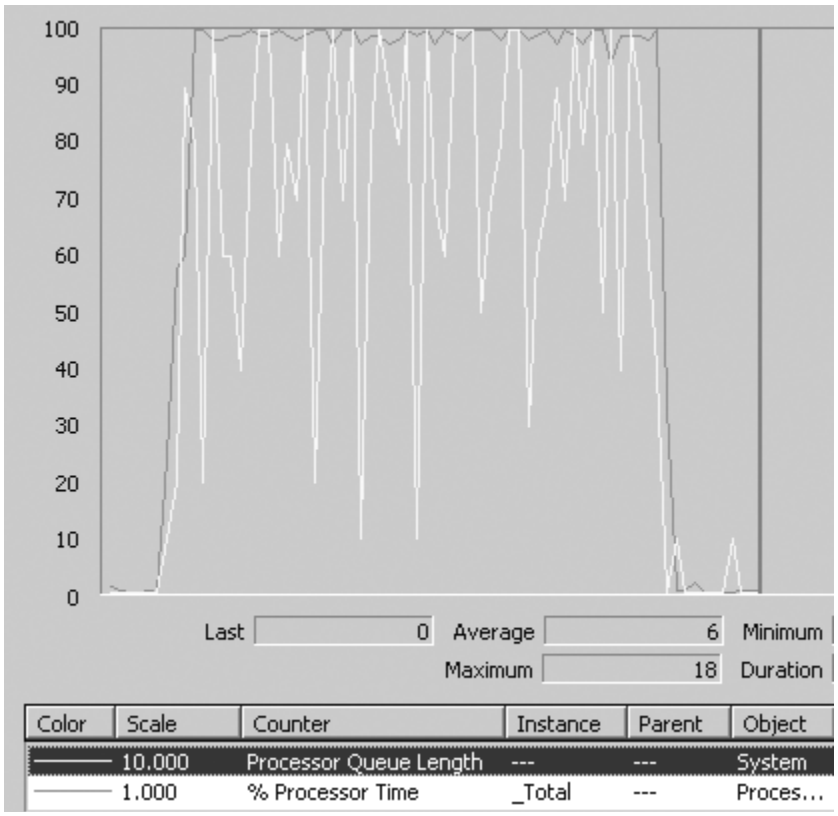


図 16.4 プロセッサ時間とキュー

その他の考慮事項

リソース使用率をレポートする際は、次の点を考慮します。

- **すべてのデータをレポートするタイミングとまとめるタイミングを把握する：** ほとんどの場合、テスト中に監視されたリソースのピーク値をレポートするだけで十分です。問題が検出された場合を除き、レポートではテスト中に問題が存在するかどうかを検出する適切な指標を収集していることを説明する必要があるだけです。
- **リソース使用率の指標と他の負荷データや応答データを一緒に示す：** リソース使用率の指標を、負荷データや応答時間データと同一グラフ上に示すと最も効果が高まります。パフォーマンスに関する問題がある場合、これによりさまざまな指標の関係を特定できます。
- **複数のデータ ポイントを示す場合はすべて示す：** 多くの場合、リソース使用率は測定のたびに大きく変化します。測定ごとの変化のパターンは、少なくとも現在値と同程度に重要です。平均値や傾向線を移動すると、これらのパターンがあいまいになります。その結果、間違った想定が行われ、不適切な決定が下される可能性があります。

量、処理能力、および速度

量、処理能力、および速度の指標の意味合いを解釈するのは困難ですが、これらの指標も関係者から頻繁に要求されます。このため、特定のパフォーマンス基準や特定のパフォーマンスの問題と関連付けてこれらの指標をレポートすることが重要です。一般的に要求される量、処理能力、および速度の指標の例には、以下のようなものがあります。

- 使用された帯域幅
- スループット
- 1 秒あたりのトランザクション数
- 1 秒あたりのアクセス回数
- サポートされる登録済みユーザー数
- データベースに格納できるレコードやアイテムの数

サンプル グラフ

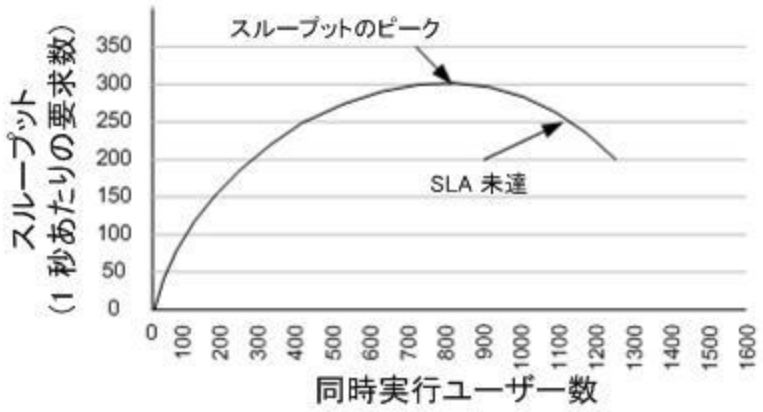


図 16.5 スループット

その他の考慮事項

量、処理能力、および速度をレポートする際は、次の点を考慮します。

- **コンテキストの指標をレポートする：** 通常、量、処理能力、および速度の指標には、単独で意味をなす値がほとんどありません。
- **テスト条件とサポート データを使用できるようにする：** これは一般的に有効ですが、量、処理能力、および速度の指標の場合は特に重要です。
- **説明にナレーションによるまとめを含める：** これも一般的に有効ですが、実際には量、処理能力、および速度の指標を理解することが重要です。

コンポーネントの応答時間

コンポーネントの応答時間は、エンド ユーザーの応答時間やリソース使用率の指標ほど一般的に関係者にレポートされませんが、頻繁に収集され、技術チームと共有されます。これらの応答時間は、開発者、アーキテクト、データベース管理者 (DBA)、および管理者が、大部分のエンド ユーザーの応答時間に関与するシステムのサブパートを判断するのに役立ちます。

サンプル グラフ

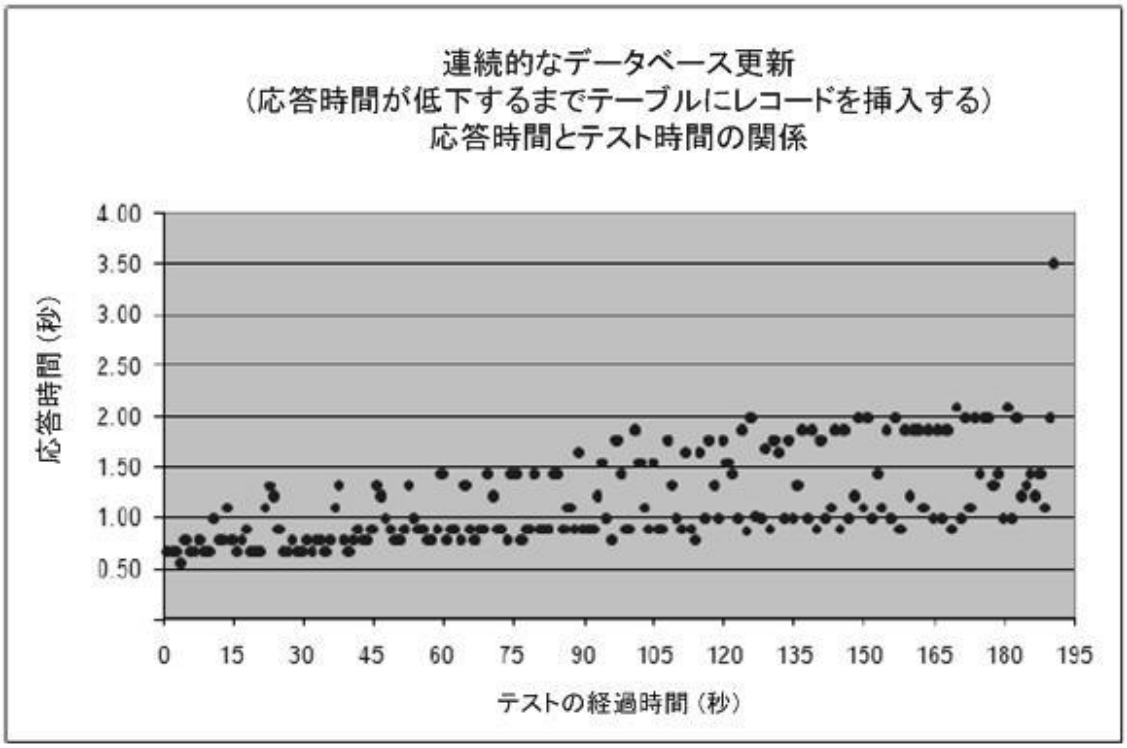


図 16.6 連続的なデータベース更新

その他の考慮事項

コンポーネントの応答時間をレポートする際は、次の点を考慮します。

- **コンポーネントの応答時間をエンド ユーザーのアクティビティに関連付ける：**コンポーネントの応答時間の影響を受けるエンド ユーザーのアクティビティは必ずしも明確ではないため、これらの関係をレポートに含めることをお勧めします。
- **コンポーネントの応答時間がどの程度重要かを説明する：**コンポーネントの処理速度が非常に遅いため、負荷がかかったときにそのコンポーネントがボトルネックになるのではないかという懸念が生じることがあります。また、そのコンポーネントによって、エンド ユーザーの応答時間が非常に低下するのではないかという懸念が生じることもあります。どちらの状況がプロジェクトに当てはまるかがわかれば、効果的な決定を下すことができます。

傾向

傾向は、強力であっても使用頻度の低いデータ レポート方法の 1 つです。傾向は、ビルドごとにパフォーマンスが向上しているか、低下しているかを示したり、負荷の増加に伴うパフォーマンスの低下率を示したりすることができます。技術チームのメンバは傾向を利用することで、最近行った変更が目的のパフォーマンスに影響を与えたかどうかをすばやく判断できます。

サンプル グラフ

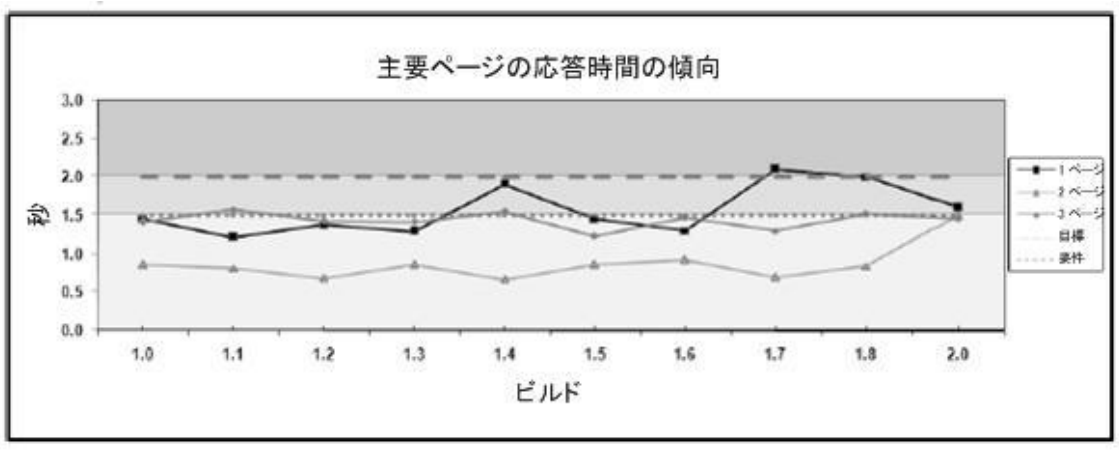


図 16.7 主要ページの応答時間の傾向

その他の考慮事項

傾向をレポートする際は、次の点を考慮します。

- **通常、少なくとも 3 つの測定値が得られるまで傾向は価値をもたらさない：** 場合によっては、4 つ以上の測定値が得られるまで傾向を効果的に検出できないこともあります。最初のデータ セットを使用して傾向グラフの作成を始めますが、実際の傾向をレポートするのに十分なデータを収集するまで、正式なレポートに傾向グラフを含めることには注意が必要です。
- **技術チームのメンバと傾向を共有してから、正式なレポートに傾向を含める：** 一般に優れた方法ですが、特に傾向に関しては優れた手法です。それは、レポートをまとめる前に、開発者、アーキテクト、管理者、および DBA は、傾向が誤った方向に進む原因となった変更を既に解消していることが多いためです。このような場合は、傾向レポートを含める必要がないと決定したり、原因を説明し、問題が解決されたことを示す注釈を付けたりすることができます。

レポートによって解決される質問

パフォーマンス テストから得るデータや結果のレポートに関しては、ほぼすべてのチーム メンバがそれぞれ独自の望み、ニーズ、および期待を持っています。このため、パフォーマンス テストから得た情報は共有されますが、さまざまなチーム メンバが期待していることや価値をあらかじめ理解しておくと、適切な詳細レベルおよび適切なタイミングで適切な人に有益な情報を提供することがはるかに容易になります。

すべての役割

チーム メンバからの一般的な質問には、次のようなものがあります。

- パフォーマンスは向上しているか、低下しているか。
- 要件やサービス レベル アグリーメント (SLA) を満たしているか。
- 使用できるレポートはどれか。
- レポートはどのくらいの頻度で入手できるか。
- より詳しい (または詳しくない) レポートを入手できるか。

幹部関係者

幹部関係者は、レポートに対して非常に具体的なニーズや期待を抱きがちで、他のチーム メン

バのニーズや期待とはまったく異なる傾向があります。こうした関係者は、要点を明確に強調する、理解しやすい少量の情報を好む傾向があります。また、一目で直感的に理解できるデータの視覚的表現と、こうした視覚的表現の "サウンドバイト" 程度の簡単な説明を好みます。最後に、他のチーム メンバより (大幅に少なくはありませんが) 少ない頻度で集約され、まとめられた情報を好む傾向があります。以下に、幹部関係者がパフォーマンス テスト レポートで解決されることを望む一般的な質問を示します。

- 出荷の準備は整っているか。
- これらの結果が運用にどのように関係するか。
- これらの結果はどの程度信頼できるか。
- 出荷準備が整うまでに、何を行う必要があるか。
- パフォーマンス テストは予定どおりに進行しているか。
- パフォーマンス テストから価値を得られているか。

プロジェクト レベルの上司

プロジェクト マネージャ、開発リーダーやマネージャ、テスト リーダーやマネージャなど、プロジェクト レベルの上司は、幹部関係者とまったく同じニーズや質問を抱えています。ただし、プロジェクト レベルの上司は幹部関係者よりも頻繁に、より詳細な回答を求めます。また、通常、次のような質問の答えを望みます。

- パフォーマンスの問題点は効果的に検出されているか。
- パフォーマンスの問題点は効果的に解決されているか。
- 現在実施していないパフォーマンス テストのどれを実施すべきか。
- 現在実施中のどのパフォーマンス テストが価値をもたらさらないか。
- 現在、ブロッカーは存在するか。あるとすれば、それは何か。

技術チームのメンバ

技術チームのメンバは、上司や幹部関係者からのすべての質問にもある程度関心がありますが、テスト結果、監視データ、監視内容、および分析と改良を行う機会に関連する情報を継続的に受け取ることに、より関心があります。技術チームのメンバは、次のような質問の答えを求める傾向があります。

- これらの結果は自分の専門分野や注目分野にどのような意味を持つか。
- 前回のテスト結果はどこで確認できるか。
- 未加工のデータはどこで入手できるか。

- 次回のテスト実行時に指標 X を把握できるか。

結果を共有する方法

最も基本的な観点では、結果を共有する方法には、未加工データの表示、技術レポート、および関係者向けレポートという 3 つの異なる種類があります。これらの結果共有方法はすべて、結果、監視内容、懸案事項、および推奨事項のタイムリーで正確かつ適切なコミュニケーションに基づいて行われますが、それぞれ異なるユーザーを対象とし、データを最も効果的に伝達する方法は大きく異なります。

未加工データの表示

レポートのシナリオは明確ではありませんが、共同作業を目的とした未加工データの共有には、共同作業の効果を高めるためにレポートに適用される、データ プレゼンテーションと同じ原則が数多く関係します。

一般に、大半の人が、表ではなくグラフでデータや統計を表示することを好みます。ただし、場合によっては、すべてのデータの計算結果を示すのに表が最も効果的な方法になることもあります。レポート内ではあまり表を使用せず、図やグラフの作成に使用した表形式のデータは、レポートの付録や添付ファイルとして含め、関心のある関係者が参照できるようにしておくことをお勧めします。

次の種類のテストの結果は、表形式で適切に表現できます。

- ベースライン
- ベンチマーク
- スケーラビリティ
- その他のユーザー エクスペリエンス ベースのテスト

データ量を整然と整理された形式で示し、最終結果を導き出すには、表が優れています。ただし、レポートでは表を使いすぎないように注意してください。表を飛ばして周りのテキストだけを読んだり、表に付属しているグラフだけを確認したりする人はたくさんいます。以下に示す表を使用する場合も、他の種類の表を使用する場合も、レポートでは明らかに重要な表のみを提示するようにしてください。サポートするすべてのデータを含む大きな表は、ごく一部の人が関心を持つかもしれませんが、ほとんどの人の興味は引かないため、レポートに付録としてみ含めます。未加工のデータは、次の形式で共有するのが最も一般的です。

- スプレッドシート
- テキスト ファイル (および正規表現検索)
- データ収集ツール ("構成済み" レポート)

技術レポート

一般に、技術レポートは未加工データの表示よりも改まった形式ですが、形式的すぎるわけではありません。技術レポートは単独で使用できる必要がありますが、現在プロジェクトで作業しているチームの技術メンバを対象とするため、関係者向けのレポートに通常含まれる補足的な詳細は一切含める必要がありません。単純な意味では、技術レポートは次の内容で構成されます。

- ワークロード モデルやテスト環境など、テストの説明
- 最低限の事前処理で容易に理解できるデータ
- 完全なデータ セットとテスト条件へのアクセス
- 監視内容、懸案事項、疑問点、要請などの共同作業のための簡単な説明

技術レポートには、次の形式のデータを含めるのが最も一般的です。

- 散布図
- パレート図
- 傾向図
- 概要スプレッドシート

関係者へのレポート

関係者へのレポートは、最も改まった形式のパフォーマンス データの共有形式です。このようなレポートは単独で使用でき、プロジェクトで日常的に技術を担当していない人が直感的に理解できる必要もあります。通常、こうしたレポートは受け入れ基準とリスクに重点を置きます。効果を高めるため、通常、次の内容を含める必要があります。

- 結果が関係する受け入れ基準
- 多くの関連データの直感的で見やすい表現
- 基準に関連した図やグラフの口頭での簡単なまとめ
- ワークロード モデルやテスト環境の直感的で見やすい表現
- 関連技術レポート、完全なデータ セット、およびテスト条件へのアクセス
- 監視内容、懸案事項、および推奨事項のまとめ

関係者へのレポートを準備する際は、ほとんどの関係者へのレポートが次の 3 つの感想のうち

1 つ (または 2 つ以上) を満たせるよう考慮します。3 つの感想はすべてそれなりに肯定的ですが、最初は肯定的でないように思われるかもしれません。これらの感想と、感想に対するお勧めの返答を以下に示します。

- **「すばらしいレポートですが、サポート データはどこに記載されていますか。」:** これは、技術関係者が最もよく示す反応です。人や組織の多くは、独自の結論を導き出せるように、すべてのデータを保持したがります。さいわい、この質問には簡単に対処できます。つまり、レポートの付録としてこうしたサポート データを記載した全スプレッドシートを含めます。
- **「見事ですが、これらのレポートが何を意味するのかわかりません。」:** この場合は、文章による説明が役立ちます。多くの場合、パフォーマンス テストやパフォーマンスの結果に慣れていない人には、結果の意味を詳しく説明する必要があります。関係者が予期していなかった悪い知らせをパフォーマンス テスタから伝えられる確率は、90% を超えることに留意してください。テスタは、関係者に結果を信頼させ、情報を前向きに捉えさせるようにする責任があります。
- **「すばらしい。これこそ、まさに求めていたレポートです。最終レポートのことは気にしないでください。これらのレポートで十分です。」:** この反応は喜ばしいことのように思われますが、そうではありません。遅かれ早かれ、上記の 2 つのうちいずれかの質問をした人、さらに悪い場合は、データの取得方法を質問した人に表やグラフが提示されます。残りのデータを記載した場所を示す最終レポートが 1 つもなければ、こうした質問に対する回答を提示していないことになるため、必ず結果をたずねられます。

技術レポートの作成

技術レポートの 6 つの主要コンポーネントを以下に示しますが、技術レポートによっては当てはまらないコンポーネントもあります。同様に、レポートで伝えようとしているメッセージによっては、追加情報を含める必要がある場合もあります。ほとんどの場合、以下の 6 つのコンポーネントを含めると優れた技術レポートになりますが、メッセージを明瞭かつ直感的に理解できるものにするには創造性が必要になることもある点に留意してください。

技術レポートを準備する際は、次の主要コンポーネントを含めることを検討します。

- 結果グラフ
- 1 つのインスタンスの測定値を示す表 (例 : 達成した最大スループット)
- ワークロード モデル (図)
- テスト環境 (注釈付きの図)

- 監視内容、懸案事項、疑問点、要請などの共同作業のための簡単な説明
- 「参考資料」セクション

結果グラフのサンプル

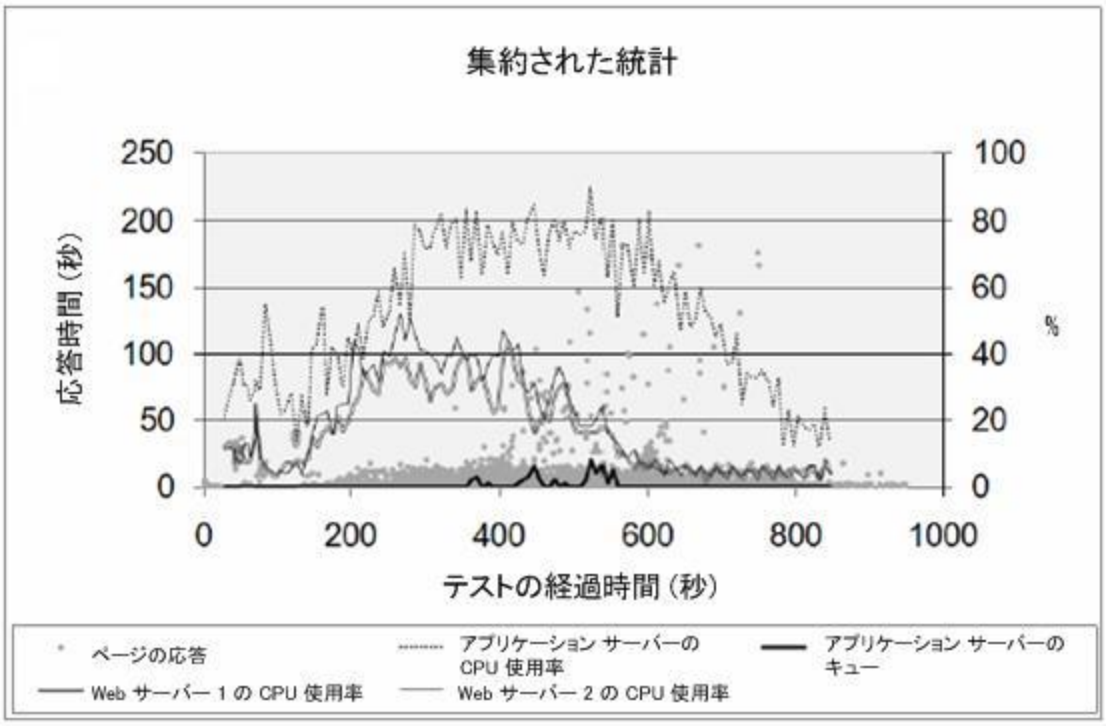


図 16.8 集約された統計

1 つのインスタンスの測定値を示す表のサンプル

その他の指標	
同時実行可能な最大仮想ユーザー数 :	125
合計スループット (バイト数):	179,746,398
平均スループット (バイト/秒):	322,126
アクセス総数 :	26,500
1 秒あたりの平均アクセス数 :	47.491

図 16.9 1 つのインスタンスの測定値

ワークロード モデル図のサンプル

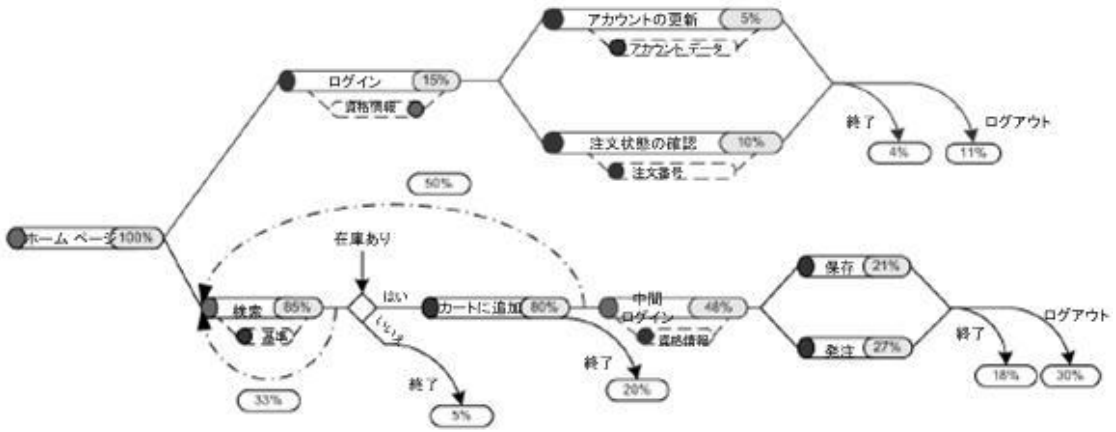


図 16.10 ワークロード モデル

テスト環境図のサンプル

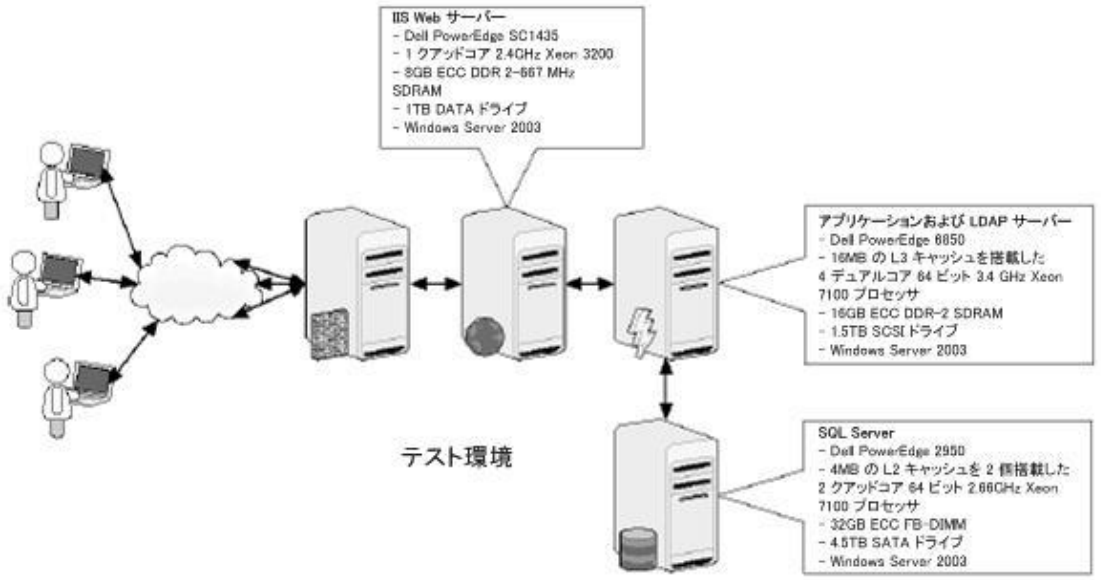


図 16.11 テスト環境

概要説明のサンプル

「結果のグラフは､応答時間とリソース使用率の両方を一緒に示しています｡詳しい調査により､アプリケーション サーバーの CPU 使用率とキューの長さが、応答時間の大幅低下に関係していることを示しています。アプリケーション サーバーの CPU 使用率が応答時間低下の要因になっているようですが、まだ確認は取れていません｡簡単に参照できるように、残りの図とグラフを補足情報として付属します｡」

「参考資料」セクションのサンプル

「未加工のデータと追加のサポート情報は、ビルドと PerfTest-{日付}-{発行番号} というタグ付きで、バージョン管理システムにチェックインされています｡」

関係者へのレポートの作成

関係者へのレポートの 8 つの主要コンポーネントを以下に示しますが、レポートによっては当てはまらないコンポーネントもあります｡同様に、レポートで伝えようとしているメッセージによっては、追加情報を含める必要がある場合もあります。ほとんどの場合、以下の 8 つのコンポーネントを含めると関係者への優れたレポートになりますが､メッセージを明瞭かつ直感的に理解できるものにするには創造性が必要になることもあります。

関係者へのレポートを準備する際は、次の主要コンポーネントを含めることを検討します。

- 結果が関係する基準
- 結果グラフ
- 1 つのインスタンスの測定値を示す表 (例 : 達成した最大スループット)
- 基準に関連した図やグラフの口頭での簡単なまとめ
- ワークロード モデル (図)
- テスト環境 (注釈付きの図)
- 監視内容、懸案事項、および推奨事項のまとめ
- 「参考資料」セクション

基準に関する説明のサンプル

「このレポートは、最も一般的と想定される使用シナリオで、予想されるピーク時の半分の負荷をかけた状態でのエンド ユーザーの応答時間に関するコンプライアンスに関係します。関係する要件は、要件管理システムに記録された要件 Perf### から Perf??? までです。」

結果グラフのサンプル

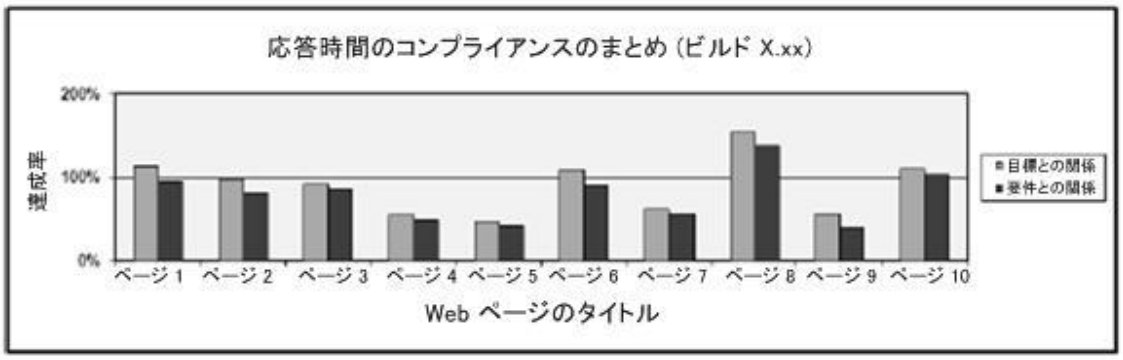


図 16.12 応答時間のコンプライアンスのまとめ

1 つのインスタンスの測定値を示す表のサンプル

サポートしている指標	最大仮想ユーザー数		125
	ピーク	平均	目標
CPU 使用率 : Web サーバー 1	42%	18%	<75%
CPU 使用率 : Web サーバー 2	53%	21%	<75%
CPU 使用率 : アプリケーション サーバー	89%	48%	<75%
キュー : アプリケーション サーバー	23	2	1

図 16.13 1 つのインスタンスの測定値

基準に基づく結果のまとめのサンプル

「ページ 8 とページ 10 の応答時間を除き、収集した全指標が目標値を実現しました。

- ページ 10 は目標値に 2% 足りませんでした。
- ページ 8 は目標値に 38% 足りませんでした。」

ワークロード モデル図のサンプル

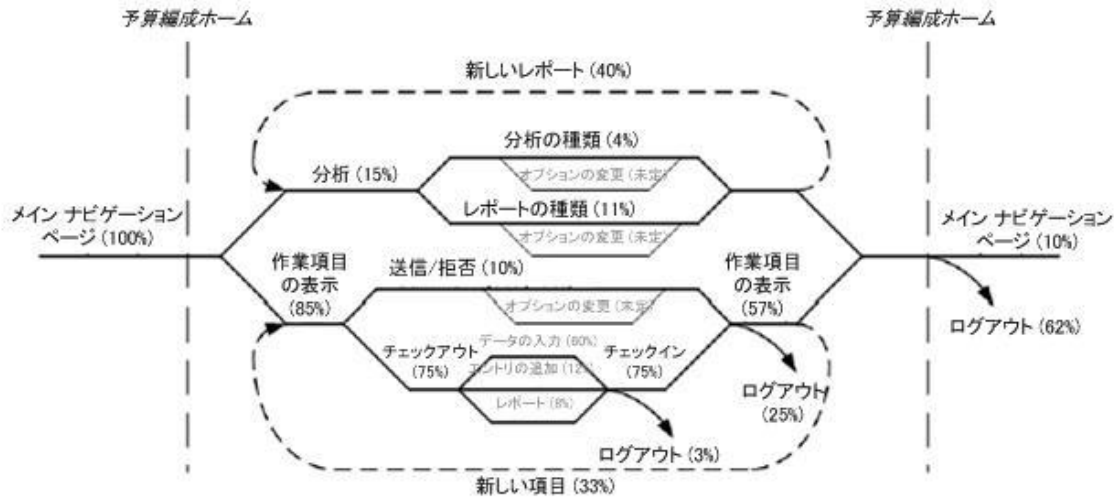


図 16.14 ワークロード モデル

テスト環境図のサンプル

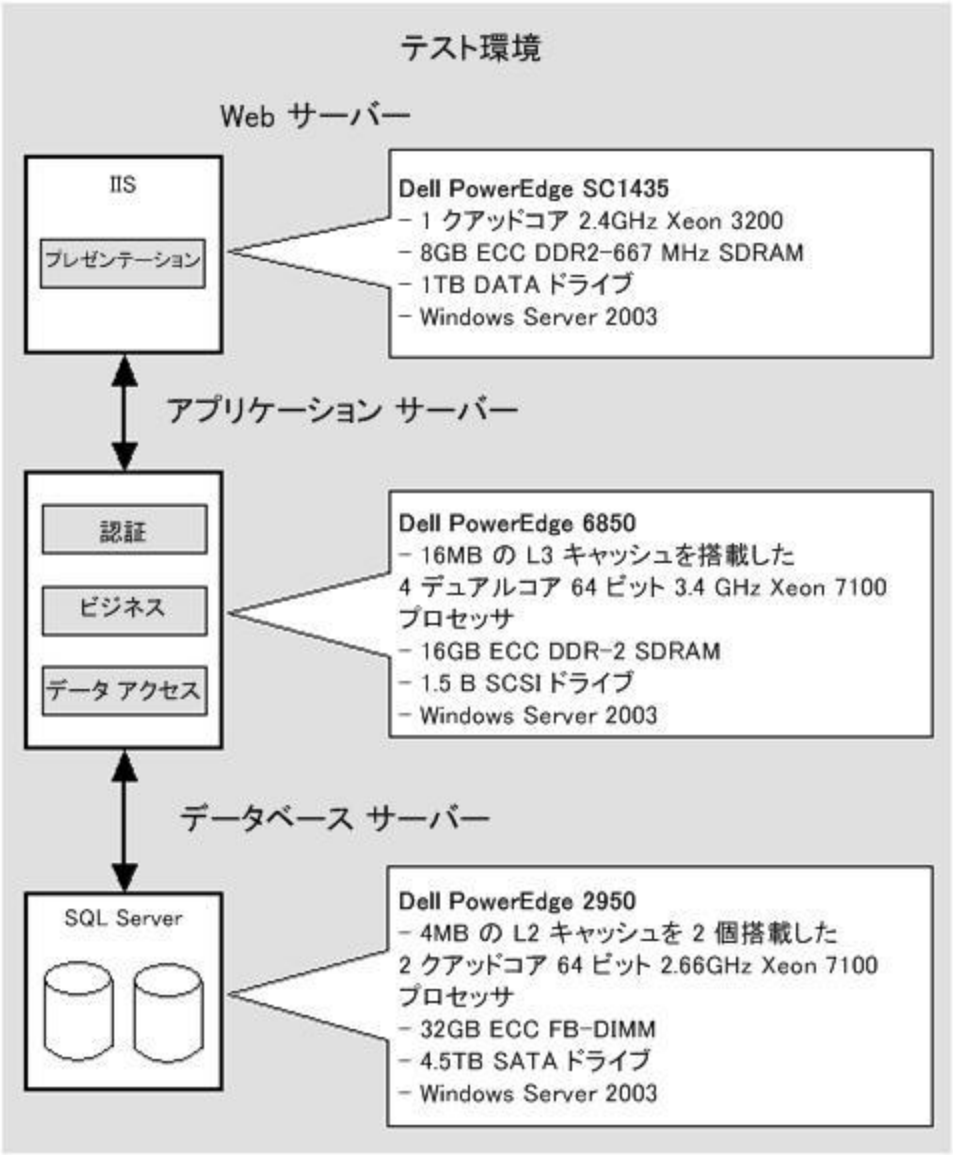


図 16.15 テスト環境

監視内容および推奨事項に関する説明のサンプル

「パフォーマンス テストとチューニングのチームは、テストの条件と結果を基に、以下の操作を実行することをお勧めします。

1) より難しいシナリオで、より高い負荷をかけて、パフォーマンス テストを続行します。
2) ページ 8 とページ 10 が受け入れ基準に達していない根本原因の確認を最優先し、その後、チューニングする必要があります。
3) 新たなページで受け入れ基準に達しないことが示される場合は、根本原因の特定とチューニングのサイクルに取り掛かります。」

「参考資料」セクションのサンプル

「このレポートの作成に使用したすべてのデータ、およびそのデータを生成したテストの実行に使用したすべてのデータは、リリース候補と PerfTest-{日付}-{RC 番号}-Validation というタグ付きで、読み取り専用としてバージョン管理システムにチェックインされています。

バージョン管理システムにアクセスしなくても、同じデータを個人の {¥¥共有リソース¥場所} に一時的にコピーしています。」

まとめ

パフォーマンス テスト レポートは、技術上およびビジネス上の重要な意思決定をサポートする結果データを提供するプロセスです。効果的なレポートを作成するには、データを利用するユーザーを考慮して、そのデータの最適な提示方法を決めることが重要です。最も効果的に行われたパフォーマンス テストの結果は、その結果を得た方法の背後にある分析、比較、および詳細を表し、ビジネス上の重要な意思決定に影響します。

第 VIII 部

パフォーマンス テストの技法

内容 :

- **Web アプリケーションの負荷テスト**
- **Web アプリケーションのストレス テスト**

第 17 章 Web アプリケーションの負荷テスト

目的

- 負荷テストの主な概念を理解する。
- Web アプリケーションの負荷テストを行う方法について学習する。

概要

ここでは、Web アプリケーションの負荷テストを行う方法について説明します。負荷テストは、アプリケーションの最大処理能力、および処理能力を低下させる可能性があるすべてのボトルネックを特定するのに役立ちます。Web アプリケーションの負荷テストを実行する基本的なアプローチは、次のとおりです。

1. パフォーマンスが重要なシナリオを特定する。
2. 主なシナリオ間で全負荷を分散するためのワークロード プロファイルを特定する。
3. パフォーマンス対象に照らして検証するために収集する指標を特定する。
4. 負荷のシミュレーションを行うためのテストを設計する。
5. 設計したテストに応じて負荷を実装するツールを使用し、指標を取得する。
6. テスト中に取得した指標データを分析する。

上記の手順は、反復テスト処理を使用することにより、パフォーマンス目標の実現に役立ちます。

Web アプリケーションの負荷テストを行う理由は数多くあります。最も基本的な種類の負荷テストは、通常時と予想ピーク時両方の負荷状態で Web アプリケーションの動作を確認するために使用します。負荷テストを始めるときは、少数の仮想ユーザーから開始し、通常時からピーク時に向かって徐々に負荷を増やすことをお勧めします。その結果、このように徐々に負荷を増やしたときにアプリケーションの動作がどのように変わるかを確認できます。最終的には、パフォーマンス対象のしきい値の上限まで増加します。たとえば、サーバー プロセッサの使用率が 75% に達するか、エンド ユーザーの応答時間が 8 秒を超えるまで負荷を増やし続けます。

本章の使い方

ここでは、負荷テストの主な概念と、Web アプリケーションの負荷テスト関連の手順について理解します。次に示す各セクションの説明を参考に、この章を最大限に活用してください。

- 「入力」および「出力」では、Web アプリケーションの負荷テストに関する主な入力と、負荷テストの主な結果について理解します。
- 「負荷テストのアプローチ」では、Web アプリケーションの負荷テストのアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- 手順に関するさまざまなセクションで、Web アプリケーションの負荷テストに関連する各手順の詳細を理解します。

入力

Web アプリケーションの負荷テストに有効な入力を以下に示します。

- パフォーマンスが重要な使用シナリオ
- ワークロード モデル
- パフォーマンス受け入れ基準
- 受け入れ基準に関連付けられるパフォーマンス指標
- Web アプリケーションの設計者や開発者との面談で返されたフィードバック
- アプリケーションのエンド ユーザーとの面談で返されたフィードバック
- アプリケーションを保守および管理する運用スタッフとの面談で返されたフィードバック

出力

負荷テストから得られる主な結果を以下に示します。

- 負荷およびパフォーマンス テスト向けに更新されたテスト計画とテスト設計
- スループット、応答時間、リソース使用率など、さまざまなパフォーマンス測定値
- ホワイトボックス テストの段階で分析する必要がある潜在的ボトルネック
- さまざまな負荷レベルでのアプリケーションの動作

負荷テストのアプローチ

Web アプリケーションの負荷テストに関連する手順を以下に示します。

1. 手順 1 - パフォーマンス受け入れ基準の特定
2. 手順 2 - 主なシナリオの特定
3. 手順 3 - ワークロード モデルの作成
4. 手順 4 - 対象負荷レベルの特定
5. 手順 5 - 指標の特定
6. 手順 6 - 具体的なテストの設計
7. 手順 7 - テストの実行
8. 手順 8 - 結果の分析

以下に、これらの手順を図示します。図の後、各手順について詳しく説明します。

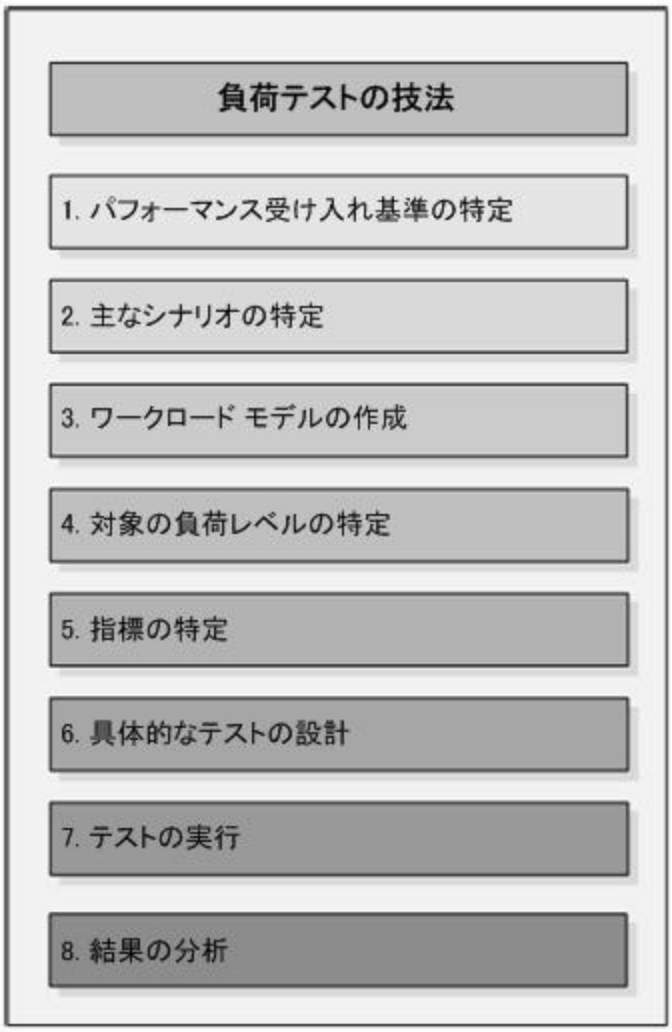


図 17.1 負荷テストの手順

手順 1 - パフォーマンス受け入れ基準の特定

アプリケーションの開発ライフ サイクルの初期段階でパフォーマンス受け入れ基準を特定しておくことが最も重要です。多くの場合、アプリケーションの受け入れ基準を記録し、チーム メンバ全員がアクセスできる場所に、レビューおよびコメントできる形式で格納すると非常に役立ちます。通常、基準は、ビジネス、業界、テクノロジ、競合他社、ユーザーからの要件などを均衡させて特定します。

多くの場合、テストの対象には次のようなものが含まれます。

- **応答時間 :** 製品カタログは 3 秒以内に表示される必要があるなどです。
- **スループット :** システムでは 1 秒間に 100 個のトランザクションをサポートする必要が

あるなどです。

- **リソース使用率 :** プロセッサ、メモリ、ディスク入出力 (I/O)、ネットワーク I/O など、アプリケーションで使用されているリソースの量は見落とされがちです。
- **最大ユーザー負荷 :** このテストの対象で、特定のハードウェア構成で実行できるユーザー数が決まります。
- **ビジネス関連の指標 :** この対象は、通常値とピーク値の業務量に対応します。たとえば、特定の時間に処理される注文数またはヘルプ デスクへの問い合わせ数などがあります。

手順 2 - 主なシナリオの特定

シナリオとは、一般に複数のアプリケーション アクティビティを含む、予想されるユーザー パスです。主なシナリオには、具体的なパフォーマンス目標のあるシナリオ、リスクが高くなると考えられるシナリオ、最もよく使用されるシナリオ、パフォーマンスに大きな影響を与えるシナリオなどがあります。主なシナリオを特定する基本手順を以下に示します。

1. Web アプリケーションのすべてのシナリオを特定します。たとえば、最も基本的な E コマース アプリケーションでは、次のユーザー シナリオをサポートする必要があります。
    - カタログの参照
    - 製品の検索
    - 発注
2. 各シナリオに関連するアクティビティを特定します。たとえば、"発注" シナリオには、次のアクティビティが含まれます。
    - アプリケーションにログオンする。
    - 製品カタログを参照する。
    - 特定の製品を検索する。
    - ショッピング カートに商品を追加する。
    - クレジット カードの詳細を検証してから、発注する。
3. 最もよく実行されるシナリオ、または最もリソースを集中的に使用するシナリオを特定します。これらのシナリオは、負荷テストで使用される主なシナリオになります。たとえば、E コマース アプリケーションでは、おそらくカタログの参照が最もよく実行されるシナリオです。これに対して、発注シナリオは、データベースにアクセスするため、最もリソースを集中的に使用するシナリオになります。
    - 既存の Web アプリケーションで最もよく実行されたシナリオは、ログ ファイルを確認することで判断できます。

    - 新しい Web アプリケーションで最もよく実行されるシナリオは、市場調査、履歴データ、市場動向などから判断できます。
    - リソースを集中的に使用するシナリオは、設計ドキュメントや実際のコード実装を使用して特定できます。主なリソースは次のとおりです。
        - プロセッサ
        - メモリ
        - ディスク I/O
        - ネットワーク I/O

これらの主なシナリオを特定したら、それらのシナリオを使用して、ワークロード プロファイルの作成と負荷テストの設計を行います。

手順 3 - ワークロード モデルの作成

ワークロードの分散を定義する際は、ユーザー シナリオの特性を判断するため次の点を考慮します。

- ユーザー シナリオは、ユーザーがタスクを完了するために使用するナビゲーション パス (中間手順や中間アクティビティなど) と定義されます。また、ユーザー セッションとも考えられます。
- 通常、ユーザーはセッション中、ページを表示するたびに一時停止します。これは、ユーザーによる遅延時間または思考時間と呼ばれます。
- 複数のユーザーにまたがって 1 つのセッションを表示する場合、セッションの持続時間は平均値になります。同時使用数、重複するユーザー数、または単位時間あたりのユーザー セッション数に変換される負荷レベルを定義する際は、この点を考慮することが重要です。
- 新しいユーザーとリピート ユーザー、またはそのいずれかがすべてのシナリオを実行できるわけではありません。主なユーザーを予測し、それに応じてテストを実行します。

手順 4 - 対象の負荷レベルの特定

前の手順で特定したワークロードの分散に適用する負荷レベルを特定します。対象の負荷レベルの特定は、テストを使用してさまざまな運用環境の負荷状況を予測または比較できるようになることを目的とします。 対象の負荷レベルの特定に共通して使用される入力を以下に示します。

- パフォーマンス対象に応じた業務量 (現在と予定の両方)
- 主要シナリオ

- 作業分布
- セッションの特性 (ナビゲーション パス、持続時間、新しいユーザーの割合)

上記の項目を組み合わせると、特定の対象負荷がかかったワークロード モデルを実装するために必要な残りの詳細を判断できます。

手順 5 - 指標の特定

パフォーマンス テストの実行中に収集できる指標は、事実上、無限に存在します。ただし、指標を収集しすぎると分析しにくくなり、アプリケーションの実際のパフォーマンスが低下する可能性があります。このため、パフォーマンスの目標に最も関連する指標、およびボトルネックを特定する場合に役立つと予想される指標を特定することが重要です。価値ある情報をもたらすのは、正確かつ状況に基づいて分析され、厳選された指標だけです。

以下に、プロジェクトに最も貴重な情報をもたらす指標を特定するための提案事項をいくつか示します。

- **容易にテストできるアプリケーションのパフォーマンスに関連する質問を定義する :** たとえば、発注する際のチェックアウト応答時間と、1 分間の発注数をたずねる質問には、確実な回答が得られます。
- **これらの質問に対する回答から、外部の期待値と比較する品質目標を決定する :** たとえば、チェックアウト応答時間は 30 秒で、1 分間に最大 10 個の発注を行う必要があるなどです。この回答は、市場調査、履歴データ、市場動向などに基づきます。
- **指標を特定する :** パフォーマンス関連の質問と回答の一覧を使用して、これらの質問と回答に関連する情報を提供する指標を特定します。
- **サポートしている指標を特定する :** 同じアプローチを使用して、パフォーマンスの測定とシステムのボトルネックの特定に重点を置く、より低レベルの指標を特定できます。ほとんどのチームでは、低レベルの指標を特定する際に、1 人のユーザーによる負荷状態や通常の負荷状態でこれらの指標のベースラインを判断することが有効であることがわかります。このことは、アプリケーションの受け入れ可能な負荷レベルを判断するのに役立ちます。ベースラインの値は、さまざまな負荷レベルのアプリケーションのパフォーマンスを分析するのに役立ちます。また、ベースライン値を使用して、ビルドまたはリリース間の傾向分析を開始できます。
- **収集する指標を定期的に再評価する :** 目標、優先順位、リスク、および現在の問題点によって、プロジェクトの方向が変わることは避けられません。このような変化により、さま

ざまな指標によって、以前に確認されていた価値よりも多くの価値がもたらされることがあります。

また、アプリケーションのパフォーマンスをより詳しく評価したり、潜在的ボトルネックを特定したりするには、次のカテゴリの指標を監視すると役立つことがよくあります。

- **ネットワーク固有の指標 :** この一連の指標によって、ルーター、スイッチ、ゲートウェイなど、ネットワークの全体的状態と効率に関する情報が提供されます。
- **システム関連の指標 :** この一連の指標は、サーバーのリソース使用率を特定するのに役立ちます。利用するリソースには、プロセッサ、メモリ、ディスク I/O、およびネットワーク I/O があります。
- **プラットフォーム固有の指標 :** プラットフォーム固有の指標は、Microsoft .NET Framework 共通言語ランタイム (CLR) や ASP.NET 関連の指標など、アプリケーションをホストするために使用するソフトウェアに関連します。
- **アプリケーション固有の指標 :** これらの指標には、アプリケーションの状態を監視し、パフォーマンスの問題点を特定するためにアプリケーション コードに挿入されるカスタム パフォーマンス カウンタなどがあります。カスタム カウンタを使用して、特定のロックの取得を待機している同時実行スレッド数や、Web サービスへの外部呼出しを行うためにキューに入れられる要求数を判断できます。
- **サービス レベルの指標 :** これらの指標は、アプリケーションの全体的なスループットと待機時間を測定するのに役立ちます。つまり、特定のビジネス シナリオに結び付けられている可能性があります。
- **ビジネスの指標 :** これらの指標は、特定の時間内に行われる発注数など、ビジネスに関連する情報のインジケータです。

手順 6 - 具体的なテストの設計

これで、シナリオ、主な指標、およびワークロードの分析を使用して、実施する具体的なテストを設計できるようになりました。一般に、目的、収集するデータ、含めるさまざまなシナリオ、対象の負荷レベルは、テストごとに異なります。重要なのは、チームがアプリケーションを理解、評価、またはチューニングするために必要な情報を収集できるテストを設計することです。

テストを設計する際は、次の点を考慮します。

- テストの設計は、ツールでの実装が難しいため、変更しません。
- 設計どおりにテストを実装できない場合は、実装するテストに関連する詳細を記録するよ

うにします。

- 実際のテストを作成するために必要なすべての補足データを、モデルに含めるようにします。
- パフォーマンス テストに無効なデータを含めることを検討します。たとえば、最初にパスワードを誤入力し、2 回目に正しく入力するユーザーを含めます。
- 通常、初めて使用するユーザーは、使用経験のあるユーザーよりも各ページや操作に費やす時間が長くなります。
- 考えられる最適なテスト データは、運用データベースまたはログ ファイルから収集したテスト データです。
- 人間のエンド ユーザーだけでなく、システム ユーザーやバッチ プロセスについても考えてください。たとえば、ユーザーがサイトで操作を行っている間に、注文状態を更新するために実行されるバッチ プロセスがあるかもしれません。この場合、こうしたプロセスがリソースを消費している可能性があるので、こうしたプロセスを考慮に入れる必要があります。
- 完璧を求めすぎないようにしてください。また、単純化しすぎるという落とし穴にも陥らないようにしてください。一般に、妥当なテストが設計できたらテストの実行を開始し、結果を収集しながら段階的に設計を拡張することをお勧めします。

手順 7 - テストの実行

正確さに欠ける負荷シミュレーションは、それ以前のアクティビティのすべての作業を無意味にする可能性があります。テストの実行によって収集されたデータを把握するには、負荷シミュレーションがテストの設計を反映している必要があります。シミュレーションがテストの設計を反映しないと、結果が誤解されやすくなります。負荷シミュレーションを準備する際は、次の手順を考慮します。

1. テスト環境ができる限り厳密に運用環境を反映するようにテスト環境を構成します。その際、テスト環境と運用環境のすべての違いに注意します。
2. 特定した指標やリソース使用率を対象とするパフォーマンス カウンタが測定中で、シミュレーションの正確性を損なわないことを確認します。
3. 適切な負荷生成ツールを使用し、テスト設計で指定された特性のある負荷を作成します。
4. シミュレーションの正確性を検証するために、負荷生成ツールを使用して、テスト設計で指定されている対象負荷レベルまで構築してからテストを実行します。テストを実行する際は、次の点を考慮します。
    - ユーザー プロファイルに照らして分散させた少数のユーザーで負荷テストを

開始し、徐々に負荷を増やします。シミュレーションの正確性を評価するときに、負荷を増やすたびにシステムが安定する時間がかかることを考慮に入れることが重要です。

- 負荷がテストの設計で指定されている対象負荷レベルを上回った場合でも、継続的に負荷を増やし、パフォーマンス対象に対して特定されたリソースのしきい値に達するまで動作を記録することを考慮します。システムが、特定されたしきい値を上回るタイミングに関する情報は、テストの対象負荷レベルの指標と同じくらい重要な価値があります。
- 同様に、サービス レベルの限界値に達するまでユーザー数を増やし続けることが役立つことがよくあります。限界値を超えると、スループット、応答時間、およびリソース使用率の SLA に違反することになります。

注： 負荷の生成に使用するクライアント コンピュータ (エージェント) にストレスをかけすぎないようにします。正確なテスト結果を得るには、プロセッサやメモリなどのリソース使用率が使用率のしきい値よりもはるかに下回っているようにする必要があります。

手順 8 - 結果の分析

テスト結果を分析して、各テストの実行後または全テストの完了後のパフォーマンスのボトルネックを確認できます。結果を正しく分析するには、相関関係のある応答時間とシステム データのグラフ作成のトレーニングを行い、経験を積む必要があります。

データ分析の手順を以下に示します。

1. 取得したデータを分析し、結果を指標の受け入れ可能レベルと比較して、テスト対象のアプリケーションのパフォーマンスの傾向が目標に近づいているのか目標から離れていくのかを判断します。
2. 測定した指標を分析して、潜在的ボトルネックを診断します。必要な場合は、分析に基づいて、その後のテスト サイクルで新たな指標を取得します。たとえば、負荷テストの最初の反復処理中と仮定し、プロセスではメモリの使用が著しく増加しているとします。これは、メモリ リークの可能性を示しています。その後の反復処理では、生成に関連する新たなメモリ カウンタを取得して、アプリケーションのメモリ割り当てパターンについて調査できます。

まとめ

負荷テストは、アプリケーションの最大処理能力、およびパフォーマンスを低下させる可能性があるすべてのボトルネックを特定するのに役立ちます。

Web アプリケーションの負荷テストを実行する基本的な方法は、パフォーマンスが重要な主要シナリオの特定、主なシナリオ間で全負荷を分散するためのワークロード プロファイルの特定、パフォーマンス目標を満たしているかどうか検証するために収集する指標の特定、負荷テストのシミュレーションのために使用するテスト ケースの作成、テスト ケースに応じて負荷をシミュレーションするためのツールの使用と指標の取得、そして最後に、テスト中に取得された指標データの分析を行うことです。

第 18 章 Web アプリケーションのストレス テスト

目的

- ストレス テストの主な概念を理解する。
- Web アプリケーションのストレス テストを行う方法について学習する。

概要

ストレス テストは、非常に高い負荷がかかった状態でアプリケーションの堅牢性、可用性、および信頼性を判断することに重点を置く、パフォーマンス テストの 1 種です。ストレス テストの目標は、非常に高い負荷がかかった場合のみに発生または表面化するアプリケーションの問題点を特定することです。このような状態には、高い負荷、多くの同時実行、限られたコンピュータ リソースなどがあります。ストレス テストを正しく行うと、同期やタイミングに関するバグ、内部ロックの問題、優先順位の問題、リソースの損失に関するバグなどを検出するのに役立ちます。限界点に達するまでシステムに負荷をかけ、その結果問題を生じる可能性があるバグを検出することがストレス テストの考え方です。システムは、不十分なリソースで高い負荷の作業を処理することを想定するのではなく、受け入れ可能な範囲内の方法 (データが破損したり失われたりしない) で動作 (失敗など) をすることを想定します。

一般に、ストレス テストではさまざまな負荷がかかった状態で複数の主要運用シナリオのシミュレーションを行います。たとえば、プロセッサを集中的に使用するアプリケーションを既に実行しているサーバー上に、アプリケーションを展開するとします。その結果、アプリケーションはすぐにプロセッサのリソース不足に陥り、プロセッサ サイクルを求めてその他のアプリケーションと競合せざるを得なくなります。また、1 つの Web ページにも、ストアド プロシージャやクラスなどの 1 つのアイテムにもストレス テストを行うことができます。

ここでは、Web アプリケーションのストレス テストについて大まかに説明します。ストレス テストは、非常に高い負荷がかかった状態でのみ表面化するアプリケーションの問題を特定するのに役立ちます。

ストレスがかかった状態の例

ストレスがかかった状態の例を次に示します。

- ユーザーまたはデータに関する過剰な量の負荷。この例にはサービス拒否 (DoS) 攻撃、幅広く閲覧されるニュース アイテムで多くのユーザーに対して Web サイトを閲覧するように 3 分間促す場合などがあります。
- ディスク ドライブ障害などによるリソースの減少。
- 予期しないシーケンス。
- 予期しない停止、または停止からの復旧。

ストレス関連の兆候の例

ストレス関連の兆候の例を次に示します。

- データが失われる、または破損する。
- ストレスが取り除かれた後でもリソース使用率が非常に高い状態が続く。
- アプリケーション コンポーネントが応答しない。
- エンド ユーザーに処理されない例外が表示される。

本章の使い方

ここでは、ストレス テストの主な概念と、Web アプリケーションのストレス テスト関連の手順について理解します。

- 「入力」および「出力」では、Web アプリケーションのストレス テストに関する主な入力と、ストレス テストの主な結果について理解します。
- 「ストレス テストのアプローチ」では、Web アプリケーションのストレス テストのアプローチの概要を示します。また、チーム メンバ用の簡単なリファレンス ガイドを提供します。
- 手順に関するさまざまなセクションで、Web アプリケーションのストレス テストに関連する各手順の詳細を理解します。
- 「ストレス テストの使用シナリオ」では、ストレス テストが使用される、実際の環境でのさまざまなシナリオを理解します。

入力

ストレス テストを行う場合、おそらく次に示す項目のうち 1 つ以上を参照します。

- 前回のストレス テストの結果
- アプリケーション使用法の特性 (シナリオ)
- 非常に高い負荷がかかった状態のシナリオに関する懸念事項

- ワークロード プロファイルの特性
- ピーク時の負荷の現在の処理能力 (負荷テストで判断できます)
- ハードウェア、ネットワーク アーキテクチャ、およびデータ
- 災害に関するリスクの査定 (ブラックアウト、地震の可能性など)

出力

ストレス テストから得られる出力には、次のようなものがあります。

- ストレスがかかった状態でのアプリケーションの測定値
- ストレスがかかった状態でのアプリケーションの兆候
- 堅牢性、可用性、および信頼性の問題を解決するためにチームが使用できる情報

ストレス テストのアプローチ

Web アプリケーションのストレス テストに関連する手順を以下に示します。

1. **手順 1 - テスト対象の特定 :** テストのアクティビティから得ることを期待する結果を考慮して、ストレス テストの対象を特定します。
2. **手順 2 - 主なシナリオの特定 :** 考えられる問題を特定するためにストレス テストを行う必要がある、アプリケーションのシナリオまたは状況を特定します。
3. **手順 3 - ワークロードの特定 :** 「主なシナリオの特定」手順で特定したシナリオに適用するワークロードを特定します。これはワークロードと、ピーク時の負荷状態の処理能力の入力値に基づきます。
4. **手順 4 - 指標の特定 :** アプリケーションのパフォーマンスに関して収集する指標を特定します。これらの指標は、「主なシナリオの特定」で特定したシナリオで考えられる問題に基づきます。
5. **手順 5 - テスト ケースの作成 :** 1 つのテストを実行する手順を定義するテスト ケースと、想定する結果を作成します。
6. **手順 6 - 負荷のシミュレーション :** テスト ツールを使用して、テスト ケースごとに必要な負荷のシミュレーションを行い、結果として得られる指標データを取得します。
7. **手順 7 - 結果の分析 :** テスト中に取得した指標データを分析します。

以下に、これらの手順を図示します。図の後に、各手順を詳しく説明します。

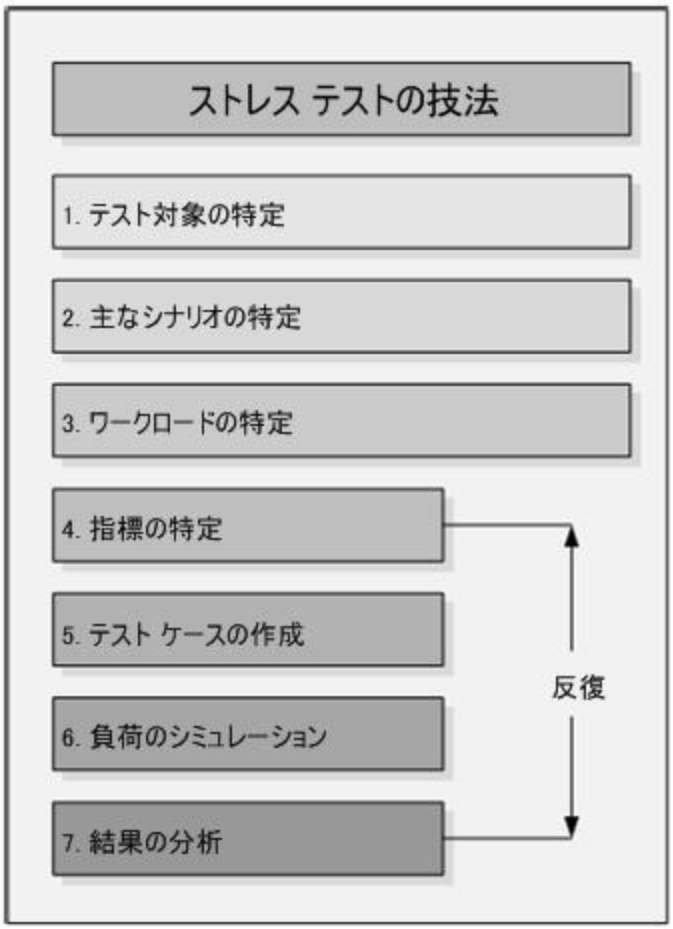


図 18.1 ストレス テストの手順

手順 1 - テスト対象の特定

ここでは、次の質問について自問したり他人に問いかけたりすると、ストレス テストから得ることを期待する結果を特定するのに役立ちます。

1. テストの目的は、運用時にシステムが壊滅的なまでに失敗する可能性がある状態を特定することか。
2. 壊滅的な障害に対する防衛策を構築するために、情報をチームに提供することが目的か。
3. メモリ、ディスク領域、ネットワーク帯域幅、プロセッサ サイクルなどのシステム リソースが少なくなった場合の、アプリケーションの動作を特定することが目的か。
4. ストレスがかかった状態で、機能が停止しないことを確かめることが目的か。たとえば、運用パフォーマンスの指標は目的を満たしていても、アプリケーションの機能が目的を満たしていない場合があります。データベースに注文が挿入されない、検索でアプリケーシ

ョンから一部の製品情報が返されない、フォーム コントロールが適切に設定されない、ストレス テスト中にカスタム エラー ページへのリダイレクトが発生するなどといった現象です。

手順 2 - 主なシナリオの特定

ストレス テストから大きな価値を得るためには、テストでアプリケーション全体の成功が最も重要である使用シナリオに重点を置く必要があります。こういったシナリオを特定するには、通常、まずストレス テストを行う必要があるシナリオを 1 つ定義して、考えられるパフォーマンスの問題を見極めます。適切なシナリオを選択する際に、次のガイドラインを検討します。

- アプリケーション全体のパフォーマンスにとってどれだけ重要であるかを基にシナリオを選択します。
- パフォーマンスに影響を与える可能性が最も高い操作のテストを試みます。このような操作には、ロックや同期の集中的な実行、長時間実行されるトランザクション、ディスクを集中的に使用する入出力 (I/O) 操作などがあります。
- 負荷テストのデータによってボトルネックになる可能性があると特定された、アプリケーションの特定の領域を基にシナリオを選択します。負荷テスト後の調整によりボトルネックを解消する必要がありますが、さらにその領域のストレス テストを行って、変更によって極めて高いストレス レベルをどの程度処理できるかを確認する必要があります。

一般的な E コマース アプリケーションに関して、他の使用シナリオから取り出して個別にストレス テストを行う必要があるシナリオの例を次に示します。

- 特定の製品の在庫を更新する注文処理シナリオ。この機能では、ロックや同期の問題が明らかになる可能性があります。
- ユーザー クエリに基づいて検索結果のページを切り替えるシナリオ。ユーザーが特に結果が広範にわたるクエリを指定する場合には、メモリ使用率に大きな影響を及ぼすことがあります。たとえば、クエリがデータ テーブル全体を返すと、メモリ使用率が影響を受けることになります。

手順 3 - ワークロードの特定

特定のシナリオに適用する負荷をしきい値の上限を超えるまでかけることでシステムにストレスをかけ、ストレスがかかった状態の結果を確認できます。アプリケーションがストレスの兆候を示し始める時点の負荷状態を判断する方法の 1 つは、負荷を段階的に増やして、さまざまな

負荷がかかった状態でアプリケーションの動作を確認することです。重大な問題が発生するまで、さまざまなワークロードで体系的にテストを行うことが重要です。ユーザーの追加、遅延時間の減少、提示されるユーザー アクティビティの数と種類の追加または削減、テスト データの調整などを行うことによって、さまざまなワークロードでテストを行うことができます。

たとえば、アプリケーションの全登録ユーザーそれぞれが 30 秒に 1 回ログオンを試みるシミュレーションを行うように、ストレス テストを指定することもできます。ダウンタイムの後、突然アプリケーションが再度使用可能になり、アプリケーションが再度オンラインになるのを待ち望んでいたすべてのユーザーがブラウザを更新するという状況をシミュレーションすることができます。実環境ではこのようなことが起こることはめったにありませんが、そのような場合にアプリケーションがどう反応するかを見ておくことに実質的価値がある程度の頻度では起こります。

正確で現実に即したテスト データ (種類と量、さまざまなユーザー ログイン、製品 ID、製品カテゴリなど) でワークロードを表現し、デッドロックやリソース使用量などの重大な問題をシミュレーションできるようにすることを忘れないでください。ストレス テストの適切なワークロードを特定する際に通常役立つアクティビティを以下に示します。

- **作業分布を特定する :** 主なシナリオごとに、シミュレーションを行う作業分布を特定します。分布は、ストレス テスト中にシナリオを実行しているユーザーの人数と種類に基づきます。
- **ピーク時のユーザー負荷を見積もる :** アプリケーションの負荷がピーク時のユーザーの予想最大数を特定します。シナリオごとに特定した作業分布を使用して、主要シナリオごとのユーザー負荷の比率を算出します。
- **アンチプロファイルを特定する :** 別の方法として、最初にアンチプロファイルを正常なワークロードに適用することもできます。アンチプロファイルでは、検討中のシナリオでワークロード分布が反転します。たとえば、注文処理シナリオの通常負荷がワークロード全体の 10% の場合、アンチプロファイルはワークロード全体の 90% になります。残りの負荷は他のシナリオに分散されます。アンチプロファイルを使用すると、重要なシナリオに通常負荷状態を上回る負荷がかけられるため、ストレス テストの出発点として役に立ちます。

手順 4 - 指標の特定

指標を適切に特定および捕捉することで、パフォーマンスの目標と比較してアプリケーションが

どの程度うまく機能したかについての情報が提供されます。また、指標により、アプリケーション内で問題のある領域やボトルネックを特定できます。

「テスト対象の特定」手順で特定した期待するパフォーマンス特性を使用して、シナリオごとに考えられる危険性に重点を置いて、取得する指標を特定します。指標は、パフォーマンスとスループットの両方の目標と関連するだけでなく、潜在的問題 (アプリケーションに組み込まれているカスタムのパフォーマンス カウンタなど) についての情報も提供します。

指標を特定する際は、直接の対象、またはこうした対象に直接的、間接的に関連するインジケータのいずれかを使用します。関連するパフォーマンス対象に関するパフォーマンス指標を次の表に示します。

パフォーマンス指標	**カテゴリ**
指標の基本セット	
プロセッサ	• プロセッサ使用率
プロセス	• メモリ使用量 • プロセッサ使用率 • プロセスのリサイクル
メモリ	• 使用可能なメモリ • メモリ使用率
ディスク	• ディスク使用率
ネットワーク	• ネットワーク使用率
取引と業務の指標	• 1 秒あたりの取引数 • 成功した取引数 • 失敗した取引数 • 成功した注文数 • 失敗した注文数
スレッド	• 1 秒あたりの競合数 • デッドロック数 • スレッド割り当て
応答時間	• 取引時間

手順 5 - テスト ケースの作成

通常、ワークロードのプロファイルと主なシナリオを特定するだけでは、テスト ケースを実装および実行するために必要なすべての情報は提供されません。ストレス テストを完全に設計するためのその他の入力には、パフォーマンスの対象、ワークロードの特性、テスト データ、テスト環境、特定した指標などがあります。各テスト設計では、期待する結果と収集する主要データの両方またはどちらかについて記述します。その結果、各テスト ケースの実行後に "合格"、"不合格"、または "結果不確定" としてその記述に印を付けることができます。

以下に、発注シナリオを基にしたテスト ケースの例を示します。

テスト 1 - 発注シナリオ

ワークロード : 1,000 人の同時実行ユーザー。
思考時間 : テスト スクリプトの各操作の後に、1 ～ 10 秒のランダムな思考時間を使用する。
テスト期間 : テストを 2 日間実行する。

想定する結果 :

- デッドロックまたはメモリの使用により、アプリケーションがホストするプロセスがリサイクルされない。
- スループットは 1 秒あたりの要求数が 35 を下回らない。
- 完了した全トランザクションの 95% の応答時間が 7 秒を超えない。
- 競合関連の問題による "サーバー ビジー" エラーが全応答の 10% を超えない。
- テスト実行中は、注文トランザクションが失敗しない。データベースのエントリ数と "成功したトランザクション" の総数が一致する。

手順 6 - 負荷のシミュレーション

ここまでの手順を適切に完了したら、ストレス テストを実行する際の負荷のシミュレーションを行う準備が整います。一般に、以下の手順でテストを実行します。

1. テスト環境が、期待する構成またはテスト用に設計した構成に一致していることを確認します。
2. 指標の収集用に、テストとテスト環境の両方が正しく構成されていることを確認します。
3. テストを実行する前に、簡単な "スモーク テスト" を実行し、テスト スクリプトやリモート パフォーマンス カウンタが正しく機能していることを確認します。
4. シナリオで他に作業を行う場合を除いて、システムをリセットしてから、正式なテストの実行を開始します。

注 : 負荷の生成に使用するクライアント コンピュータ (負荷生成コンピュータ) にストレスをかけすぎないようにします。プロセッサやメモリなどのリソースの使用率を十分低くしておき、負荷生成環境自体がボトルネックにならないようにします。

手順 7 - 結果の分析

取得したデータを分析し、結果を指標の受け入れ可能レベルと比較します。結果から要求パフォーマンス レベルに達していないことが示された場合は、ボトルネックの原因を分析し、解消します。確認された問題に対処するには、次に示す項目を 1 つ以上実行する必要があります。

- 設計の見直しを行う。
- コードの見直しを行う。
- テストの実行中に障害の原因をデバッグできる環境でストレス テストを実行する。

パフォーマンスの問題が確認されても、現時点でチューニングを行っても改善される見込みが少ない状況では、追加のテストを実施して、問題を早めに識別できるインジケータを特定し、好ましくない事態を避ける必要があります。

ストレス テストの使用シナリオ

以下に、ストレス テストを実際に適用する方法の例を示します。

- **アプリケーションのストレス テスト :** この種のテストでは、通常、コンポーネントを個別に切り分けないで、システム全体に負荷がかかった状態で複数のトランザクションに注目します。アプリケーションのストレス テストでは、データのロックやブロック、ネットワークの混雑、アプリケーション全体にわたるさまざまなコンポーネントまたはメソッドにおけるパフォーマンスのボトルネックなどに関連する欠陥を明らかにできます。テストの範囲は 1 つのアプリケーションであるため、通常、この種のストレス テストはしっかりとしたアプリケーション負荷テスト作業を行った後、または処理能力計画の最終テスト段階として使用されます。また、共有コードや共有コンポーネントから発生する競合状態と一般的なメモリ リークに関連する欠陥が見つかることもよくあります。
- **トランザクションのストレス テスト :** トランザクションのストレス テストは、運用環境で予想される負荷の量を上回る状態でトランザクション レベルのテストを行うことを目的とします。これらのテストでは、アプリケーション全体をテストするときに、リソース

の制約が同じで負荷だけが高いといった、ストレスの高い状態での動作の検証に重点を置きます。トランザクションを個別またはグループに分離するため、アプリケーション レベルでのテストの際に生じるコンポーネント間の相互作用による複雑さが追加されず、個々のコンポーネントのスループットの処理能力や特性を非常に明確に把握できます。これらのテストは、特定のコンポーネント レベルでのチューニング、最適化、およびエラー状態の検出に役立ちます。

- **体系的なストレス テスト：** この種のテストでは、同じシステム上で実行している複数のアプリケーション全体にストレスまたは非常に高い負荷状態が生成されます。それによって、アプリケーションの期待する機能の限界近くまで負荷を高めます。体系的なストレス テストの目的は、さまざまなアプリケーションがお互いをブロックし、メモリ、プロセッサ サイクル、ディスク領域、ネットワーク帯域幅などのシステム リソースを求めて競合する状態での欠陥を明らかにすることです。この種のテストは統合ストレス テストとしても呼ばれます。大規模な体系的ストレス テストでは、同一の統合環境ですべてのアプリケーションに同時にストレスをかけます。組織の中には、大規模なテスト ラボでこの種のテストを実行したり、ハードウェアやソフトウェアのベンダと共同でテストを行う組織もあります。

試験的ストレス テスト

試験的ストレス テストは、システム、アプリケーション、またはコンポーネントを、実環境で起こる可能性は低くても、起こる可能性がある一連の例外的なパラメータまたは状態に置くためのアプローチです。一般に、試験的テストは、同時学習、テスト設計、およびテスト実行の対話型プロセスと見なすことができます。通常、試験的ストレス テストは、既存のテストを変更するか、アプリケーションやシステムの管理者と共同作業して、システム内で起こる可能性は低くくても起こり得る状態を作成することによって設計します。この種のストレス テストは、通常、特定の障害モードと関連する、さらに体系的なストレス テストが必要かどうかを判断するために実施されるため、この種のストレス テストを個別に実行することはめったにありません。「ある状況でシステムがどのように反応するか」という質問の答を明らかにするための試験的ストレス テストの例をいくつか示します。

- 全ユーザーが同時にログオンする。
- 負荷分散機能を突然停止する。
- 負荷のピーク時に全サーバーが定時ウイルス スキャンを同時に開始する。
- 使用がピークのときにデータベースをオフラインにする。

まとめ

ストレス テストは、非常に高い負荷がかかった状態でのみ表面化するアプリケーションの問題を特定するのに役立ちます。このような状態は、メモリ、プロセッサ サイクル、ネットワーク帯域幅、ディスク領域などのシステム リソースが不足することから、Web アプリケーションでよく見られる、予測できない使用パターンによって発生する過剰な負荷にまで及びます。

ストレス テストで中心となるのは、アプリケーションの堅牢性、信頼性、および安定性に重点を置いた対象と主要ユーザー シナリオです。正しい手法を適用し、テスト結果を効果的に分析することによってストレス テストの効果が高まります。適切な手法を適用できるかどうかは、ユーザー負荷とデータ量の両方に関するワークロード状態の再現、主要シナリオの再現、および主要パフォーマンス指標の解釈を的確に行えるかどうかに左右されます。