Engineer

スケーラビリティー

📈 負荷が増えたとき、マシンの追加だけで対応できる度合い

どのようなものか

スケーラビリティーとは、リソースを追加したときに、システムの処理能力がどのように伸びるかです。マシンを2倍にしたとき、ほぼ2倍の負荷を同じ応答時間で処理できれば、よくスケールします。追加したハードウェアが、自分では解消できない何かを待つことに時間の大半を費やすなら、うまくスケールしません。負荷と費用の関係を表す言葉であり、システムにあるかないかの機能ではありません。

拡張には二つの方向があります。スケールアップ、すなわち垂直スケーリングは、より多くのコア、より多くのメモリー、より速いディスクを持つ、大きなマシンにすることです。ソフトウェアを変える必要はありませんが、購入に見合う最大のマシンが上限になります。スケールアウト、すなわち水平スケーリングは、複数のマシンで仕事を分担することです。同じ上限はありませんが、分割できる仕事だけが恩恵を受けます。

HTTPは、スケールアウトを念頭に形作られました。RFC 9110は、多くの実装が、HTTPのステートレス性を利用して、プロキシー経由の接続を再利用したり、複数のサーバーへリクエストを動的に分散したりすると説明しています。リクエストの間に何も保持しないウェブ層なら、ロードバランサーの後ろに同じインスタンスを増やせば拡張でき、どのインスタンスでも、どのリクエストにも応答できます。

負荷には複数の側面があり、ある側面では伸びても、別の側面では止まることがあります。1秒あたりのリクエスト数、保存するデータ量、テナント数、同じレコードを奪い合う書き込み元の数です。最後が最も難しいのは、周囲に何台のサーバーがあっても、一つの行を一度に変更できるトランザクションは一つだからです。

その計算を表すのが、アムダールの法則です。処理の一部を順番に実行しなければならないとき、並列のハードウェアが速めるのは残りの部分だけなので、直列部分が上限を決めます。処理の5%が直列なら、マシンが千台あっても、全体が20倍を超えて速くなることはありません。データベースでは、直列部分はロックであることが多く、アプリケーションでは、一つの共有カウンター、キュー、セッションストアであることが多くあります。

概念を判断に生かす

「スケーラビリティーとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

スケーラビリティーは、成長にかかる費用を決めます。よくスケールするシステムでは、トラフィックが倍になっても、請求書に追加の費用が載るだけです。うまくスケールしなければ、負荷よりも請求額の方が速く増えたり、ハードウェアを増やしてもまったく役立たなくなったりし、トラフィックが流れている最中の再設計が対策になります。

ハードウェアは、プロセッサー、メモリー、ディスクのボトルネックを緩和できますが、非効率な処理や直列化は残り得ます。不要にテーブル全体を走査するクエリーは、大きなサーバーでも処理を浪費し得ます。大きなマシンにしても、一つの共有行に対する更新の直列化はなくなりません。また、アプリケーションサーバーを増やすと、待ち行列に書き込み元が増えることもあります。マシンを増やす必要がありそうに見えるシステムの多くで、まず必要なのは、何が起きているかを読み解くことです。

PostgreSQLで、最初に読み解く場所は `pg_stat_statements` です。正規化した各SQL文について、呼び出し回数と実行時間を記録する拡張です。平均ではなく合計時間で並べると、サーバーが実際に何に時間を使っているかが分かります。2ミリ秒のクエリーを1時間に100万回実行すると、その時間のうち2,000秒を使います。一方、2秒のレポートを2回実行しても、4秒です。

```
SELECT calls, round(total_exec_time) AS total_ms,
       left(query, 60) AS query
  FROM pg_stat_statements
 ORDER BY total_exec_time DESC
 LIMIT 10;
```

マシンを増やす方向へ一段進むごとに、固有の運用費用がかかるため、順序が重要です。SQL文の無駄を減らしたり、インデックスを改善したりすれば、サーバーをもう一台追加せずに性能を高められる場合があります。読み取りレプリカは、設定の変更で導入します。パーティショニングで変えるのは、一つのテーブルです。データを別々のデータベースサーバーに分けるシャーディングは、周辺の運用を変えます。マイグレーション、バックアップ、復元、レポートでは、個別のシャードを考慮しなければなりません。シャード間の結合をアプリケーションコードへ移す場合もあり、シャードキーが分からないクエリーは、すべてのシャードへ問い合わせなければならない場合もあります。

実際の活用例

中規模のアプリケーションを支えるPostgreSQLデータベースには、利用者のアカウント、テナント、地理データ、日々の電力消費量が入っていました。トラフィックは少ないのに、データ量が増えるにつれて遅くなりました。依頼は、アーキテクチャーを全面的に変えずに10倍速くすることであり、将来のデータ増加に対応する余地も目標に含んでいました。実施したのは、空間データのGiSTインデックス、テナントと日付範囲の複合インデックス、電力テーブルの時間によるパーティショニング、地理データのハッシュパーティショニング、`EXPLAIN ANALYZE`で検証したクエリーの書き換え、接続プールです。応答時間は元の10分の1となり、データベースのリソース使用量は40%減りました。システムは、ハードウェア更新の必要性を抑えつつ、データ量が増えても遅くならずに対応できました。

PostgreSQLとSQL Serverのデータベースサーバーを10台運用する中規模の企業では、インフラを利用者のトラフィックが40%増えても、サービスを劣化させずに処理できるよう拡張しました。併せて、設定の調整とインデックスの最適化により、クエリーの応答時間も30%改善しました。

作業の順序は、シャーディングの前に最適化するためのノートにまとめています。まず調べ、処理の無駄を減らし、読み取りをプライマリーから外し、負荷が集中する箇所に合わせてテーブルを分割し、一台のマシンが無駄のない処理で埋まって初めてシャーディングします。同ノートは、シャーディングが妥当な場合

を三つ挙げています。すでに無駄を省いたSQL文でプライマリーが飽和している場合、データ量が保守できる範囲を超えた場合、分離そのものが要件である場合です。

問題が起きるところ

一つのデータベースの前に、ウェブサーバーを増やす。 アプリケーションサーバーは、それぞれ独自の接続を開きます。アクティブなセッション数が、データベースのコア数の何倍にもなると、サーバーは、その切り替えに時間を費やします。ハードウェアを増やす前に、接続プールで解消する問題であり、ウェブサーバーをもう一台加えると悪化します。

一つの行に更新が集中する。 在庫カウンターや残高など、すべてのリクエストが同じレコードに書くと、何台のマシンがあっても、スループットの上限は、おおむね一つのロックが処理できる書き込み量を、ロックを保持する時間で割ったものになります。ロックの中に遅い処理があれば、その上限は下がります。トランザクションを短くし、外部への呼び出しは外で行い、一つの品目がマシンの能力を超えたときは、その在庫を、所有者がそれぞれちょうど一つとなる小さな単位に分けます。

一つのサーバーに置いた状態。 一つのプロセスのメモリーに保持したセッション、そのローカルディスクに保存したアップロード、同じジョブのコピーは他にないと想定する定期ジョブは、どれも、二つ目のインスタンスが同等に働くことを妨げます。どのインスタンスも、どのリクエストにも応答できるよう、その状態を共有ストアに置きます。

レプリカを、プライマリーと同じものとして扱う。 ストリーミングレプリカには少し遅れがあるため、書いた直後にレプリカから読むと、その内容が見えないことがあります。直近の書き込みは、プライマリーから読みます。また、スタンバイ上の長いレポートは、変更の再生を妨げると中断されることがあります。PostgreSQLは、これをクエリーの競合として、競合の回避とレプリカの遅延との間で折り合いをつける設定とともに説明しています。

クエリーが分割キーを指定しない。 パーティショニングは、月やテナントなどのキーで一つのテーブルを分けるため、キーを指定したクエリーは、必要なパーティションだけを読みます。キーを省いたクエリーは、すべてのパーティションを読み、分割の恩恵を受けません。クエリーがすでに絞り込みに使う項目で分け、大きなテーブルでのみ効果を見込みます。PostgreSQLのマニュアルの目安は、データベースサーバーの物理メモリーより大きいテーブルです。

参考資料

- RFC 9110：HTTPの意味論、第3.3節 — 接続、クライアントとサーバー、実装が複数のサーバーへリクエストを分散できるステートレスな設計です。
- PostgreSQL：テーブルのパーティショニング — パーティショニングの意味、効果がある場合、クエリーが必要としないパーティションを除外するパーティションプルーニングを説明します。
- PostgreSQL：pg_stat_statements — 正規化した各SQL文の呼び出し回数と所要時間を記録する拡張です。
- PostgreSQL：ホットスタンバイ — スタンバイ上の読み取り専用クエリーと、それを中断し得る競合を説明します。
- RFC 9111：HTTPキャッシュ — HTTPキャッシュによる応答の保存と再利用を説明します。同じリクエストを繰り返すたびに、オリジンサーバーへ到達する必要をなくせます。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「スケーラビリティーとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/scalability/