Kafka が速い理由

主要ポイント

- ビデオ「System Design: Why is Kafka fast?」は、Apache Kafka の速さをそのアーキテクチャと設計選択によって説明していると考えられます。
- 研究によると、Kafka のパフォーマンスは、追加専用ログ、ゼロコピー原則、バッチ処理、そして効率的なパーティショニングによって向上しています。
- 証拠は、Kafka の速さが分散型イベントストリーミングプラットフォームの設計にあることを示唆しています。主要なコンポーネントには、プロダクサ、ブローカー、そしてコンシューマがあります。

はじめに

このブログ記事は、ByteByteGo の YouTube ビデオ「System Design: Why is Kafka fast?」の内容を基にしており、その洞察を書き込みやすく、参照しやすい形式に変換することを目指しています。Apache Kafka は、リアルタイムデータ処理において高いパフォーマンスを発揮することで知られており、この記事ではその速さの理由を探り、初心者にもアクセスしやすくしています。

Kafka の主要コンポーネント

Apache Kafka は、3 つの主要なコンポーネントを持つ分散型イベントストリーミングプラットフォームとして動作します：- **プロダクサ**：Kafka トピックにデータを送信するアプリケーション。- **ブローカー**：データを保存し管理し、複製と分散を確保するサーバー。- **コンシューマ**：トピックからデータを読み取り処理するアプリケーション。

この構造により、Kafka は大量のデータを効率的に処理し、その速さに寄与します。

アーキテクチャレイヤーとパフォーマンス最適化

Kafka のアーキテクチャは、2 つのレイヤーに分かれています：- **コンピュートレイヤー**：プロダクサ、コンシューマ、ストリーム処理のための API を含み、相互作用を促進します。- **ストレージレイヤー**：トピックとパーティションのデータ保存を管理するブローカーを含み、パフォーマンスが最適化されています。

主要な最適化には以下があります：- **追加専用ログ**：データをファイルの末尾に順次書き込むことで、ランダム書き込みよりも速くなります。- **ゼロコピー原則**：データをプロダクサからコンシューマに直接転送し、CPU オーバーヘッドを減少させます。- **バッチ処理**：データをバッチで処理することで、レコードごとのオーバーヘッドを低減します。- **非同期複製**：リーダーブローカーがリクエストを処理しながらレプリカが更新されることで、利用可能性を確保しつつパフォーマンスの低下を防ぎます。- **パーティショニング**：データを複数のパーティションに分散し、並列処理と高いスループットを実現します。

これらの設計選択は、ByteByteGo のサポートブログ記事（Why is Kafka so fast? How does it work?）で詳しく説明されており、Kafka が速さとスケーラビリティにおいて優れている理由を説明しています。

データフローとレコード構造

プロダクサがレコードをブローカーに送信すると、そのレコードは検証され、ディスクのコミットログに追加され、耐久性のために複製され、プロダクサにコミットが通知されます。このプロセスは、順次 I/O を最適化することでパフォーマンスが向上します。

各レコードには以下が含まれます：- タイムスタンプ：イベントが作成された時刻。- キー：パーティショニングと順序付けのため。- 値：実際のデータ。- ヘッダー：オプションのメタデータ。

この構造は、ブログ記事で説明されており、効率的なデータ処理を確保し、Kafka の速さに寄与しています。

調査ノート：Apache Kafka のパフォーマンスの詳細な分析

このセクションでは、ByteByteGo のビデオ「System Design: Why is Kafka fast?」を拡張し、追加のリソースを用いて、Kafka のパフォーマンスに関する包括的な探求を行います。分析は、Kafka のアーキテクチャ、コンポーネント、そして特定の最適化について、詳細な説明と例を提供して明確に構成されています。

背景とコンテキスト　Apache Kafka は、高スループット、低レイテンシのデータストリーミングを処理する分散型イベントストリーミングプラットフォームとして開発され、現代のデータアーキテクチャの柱となっています。このビデオは、2022 年 6 月 29 日に公開され、システム設計のプレイリストの一部として、Kafka が速い理由を明確にすることを目指しています。この分析は、ByteByteGo の詳細なブログ記事（Why is Kafka so fast? How does it work?）に基づいており、ビデオの内容を補完し、追加の洞察を提供しています。

Kafka の主要コンポーネントとアーキテクチャ　Kafka の速さは、その主要なコンポーネントから始まります：- **プロダクサ**：Kafka トピックにイベントを生成し送信するアプリケーションやシステム。例えば、ウェブアプリケーションがユーザーのインタラクションに関するイベントを生成する場合があります。- **ブローカー**：データを保存し、パーティションを管理し、複製を処理するクラスターを形成するサーバー。典型的なセットアップでは、故障許容性とスケーラビリティのために複数のブローカーが含まれます。- **コンシューマ**：トピックにサブスクライブしてイベントを読み取り処理するアプリケーション、例えばリアルタイムデータを処理する分析エンジン。

アーキテクチャは、Kafka をイベントストリーミングプラットフォームとして位置付け、メッセージキューとは異なり「イベント」を使用しています。これは、イベントがパーティション内でオフセットによって順序付けられ、不可変である点で明確です。ブログ記事で詳しく説明されています。

コンポーネント	役割
プロダクサ	トピックにイベントを送信し、データフローを開始します。
ブローカー	データを保存し管理し、複製とサービスを処理します。
コンシューマ	トピックからイベントを読み取り処理し、リアルタイム分析を可能にします。

ブログ記事には、このアーキテクチャを示す図が含まれており、クラスターモードでのプロダクサ、ブローカー、コンシューマの相互作用を示しています。

レイヤードアーキテクチャ：コンピュートとストレージ　Kafka のアーキテクチャは、以下の 2 つに分かれています：- **コンピュートレイヤー**：API を通じて相互作用を促進します：- **プロダクサ API**：アプリケーションがイベントを送信するために使用。- **コンシューマ API**：イベントを読み取るために使用。- **Kafka Connect API**：データベースなどの外部システムと統合。- **Kafka Streams API**：ストリーム処理をサポートし、例えば「orders」トピックに対して KStream を作成し、Serdes をシリアライズし、ksqlDB を使用してストリーム処理ジョブを実行し、REST API を使用して「ordersByProduct」に送信することができます。提供される例は、「orders」にサブスクライブし、製品ごとに集計し、「ordersByProduct」に送信して分析を行うことです。- **ストレージレイヤー**：クラスター内の Kafka ブローカーを含み、データはトピックとパーティションで組織されています。トピックはデータベースのテーブルに似ており、パーティションはノードに分散され、スケーラビリティを確保します。パーティション内のイベントはオフセットで順序付けられ、不可変であり、追加専用です。削除はイベントとして扱われ、書き込みパフォーマンスが向上します。

ブログ記事では、ブローカーがパーティション、読み取り、書き込み、複製を管理し、複製の図が含まれています。

レイヤー	説明
コンピュートレイヤー	相互作用のための API：プロダクサ、コンシューマ、Connect、Streams、ksqlDB。
ストレージレイヤー	クラスター内のブローカー、トピック/パーティションが分散、イベントがオフセットで順序付け。

コントロールとデータプレーン

- **コントロールプレーン**：クラスターのメタデータを管理し、歴史的には Zookeeper を使用していましたが、現在は KRaft モジュールで選択されたブローカーにコントローラーを置き、Zookeeper を排除し、設定を簡素化し、メタデータの伝播を特定のトピックを通じて効率化しています。
- **データプレーン**：データの複製を処理し、フォロワーが FetchRequest を発行し、リーダーがデータを送信し、特定のオフセットまでのレコードをコミットするプロセスです。提供される例は、オフセット 2、3、4 の Partition 0 です。

レコード構造とブローカー操作　各レコードは、イベントの抽象化であり、以下を含みます：- タイムスタンプ：作成時刻。- キー：順序付け、共存、保持のためのパーティショニング。- 値：データコンテンツ。- ヘッダー：オプションのメタデータ。

キーと値はバイト配列であり、Serdes でエンコード/デコードされ、柔軟性が確保されます。ブローカー操作には以下が含まれます：- プロダクサリクエストがソケット受信バッファに到着。- ネットワークスレッドが共有リクエストキューに移動。- I/O スレッドが CRC を検証し、コミットログ（データとインデックスを含む

ディスクセグメント）に追加。- リクエストが複製のためにパージトリーに保存。- レスポンスがキューに入れられ、ネットワークスレッドがソケット送信バッファを通じて送信。

このプロセスは、順次 I/O を最適化し、ブログ記事で説明されており、Kafka の速さに大きく寄与しています。

レコードコンポーネント	目的
タイムスタンプ	イベントが作成された時刻を記録します。
キー	順序付け、共存、保持のためのパーティショニングを確保します。
値	実際のデータコンテンツを含みます。
ヘッダー	追加情報のためのオプションのメタデータ。

パフォーマンス最適化　いくつかの設計選択により、Kafka の速さが向上します：- **追加専用ログ**：ファイルの末尾に順次書き込むことで、ディスクシーク時間を最小限に抑えます。- **ゼロコピー原則**：データをプロダクサからコンシューマに直接転送し、CPU オーバーヘッドを減少させます。- **バッチ処理**：データをバッチで処理することで、レコードごとのオーバーヘッドを低減し、効率を向上させます。- **非同期複製**：リーダーブローカーがリクエストを処理しながらレプリカが更新されることで、利用可能性を確保しつつパフォーマンスの低下を防ぎます。- **パーティショニング**：データをパーティションに分散し、並列処理を実現し、スループットを増加させます。

これらの最適化は、ブログ記事で詳しく説明されており、Kafka が高スループットと低レイテンシを実現し、リアルタイムアプリケーションに適している理由です。

結論と追加の洞察　Apache Kafka の速さは、追加専用ログ、ゼロコピー原則、バッチ処理、非同期複製、そして効率的なパーティショニングを活用した慎重に設計されたアーキテクチャとパフォーマンス最適化の結果です。この分析は、ビデオに基づいており、ブログ記事で補完されており、Kafka のパフォーマンスについて包括的な視点を提供しています。予想外の深さがあり、単純な概要を期待する人々にとっても、Kafka がデータストリーミングのリーダーである理由を明確にするための設計選択のバランスを示しています。

ブログ記事には、完全なアーカイブへの 7 日間の無料トライアルが含まれており、以下のサブスクリプションエンリンクでアクセスできます（このサブスクリプションエンリンク）。興味がある方は、さらにリソースを提供しています。

この詳細な探求により、ビデオの意図である Kafka のパフォーマンスに関する教育を確保し、さまざまなソースから収集された研究と洞察に基づいて、正確性と深さを確保しています。

主要な引用

- System Design: Why is Kafka fast? YouTube ビデオ
- Why is Kafka so fast? How does it work? ByteByteGo ブログ記事
- Kafka アーキテクチャ図 ByteByteGo
- Kafka 複製図 ByteByteGo

- Kafka ブローカー操作図 ByteByteGo
- ByteByteGo ニュースレターへの Kafka 記事のサブスクリプション