ホワイトペーパー

モノリシックからクラウドネイティブなオペレーショナル・データベースへの移行

YugabyteDBがデータレイヤーでマルチクラウドモビリティを提供する方法

yugabyteDB

目次

はじめに 1

オペレーショナル・データベースの進化 2

モノリシックからシャーディング型および分散型へ 2

強力な一貫性から結果整合性へ 3

ACIDからBASEへ 4

SQLからNewSQLおよびNoSQLへ 4

リレーショナルからマルチモデルへ 4

オンプレミスからクラウドホスト方式へ 5

サービスとしてのデータベース（DBaaS）の台頭 5

データベースにおける「クラウドネイティブ」の定義 5

NoSQLデータベースの3つの課題 6

運用の複雑さ 6

フラストレーションのたまるアプリケーション開発 6

一貫性のないカスタマーエクスペリエンス 7

分散SQLで非SQLの課題を解決する 7

分散型SQLを選ぶ理由 7

分散型SQLデータベースのアーキテクチャ 8

YugabyteDB：クラウドネイティブなデータベース向けのクラス最高の分散SQL 9

導入の柔軟性 9

ハイパフォーマンス 9

運用のシンプルさ 9

PostgreSQLとの互換性 9

セキュリティ 9

機能の比較 9

まとめ 10

はじめに

規模の大小を問わず、企業では、自社のミッションクリティカルなアプリケーションを、オンプレミスのデータセンター内で実行される古いモノリシックアーキテクチャからパブリック、プライベート、またはハイブリッドのクラウドインフラストラクチャで実行される最新のマイクロサービスアーキテクチャへ移行する動きが進んでいます。アプリケーション開発チームと運用チームは、DockerやKubernetesなどの多数のクラウドネイティブなインフラストラクチャテクノロジーを活用してこのような移行を実現しています。

しかし、マイクロサービスとクラウドインフラストラクチャへの移行をさらに加速させる上で1つ大きな障害となるのは、ステートフルなデータレイヤーとステートレスなアプリケーションレイヤーを、同じ速度で移動することができないことです。このステートフルなデータレイヤーは、永続データベースとインメモリ・キャッシュで構成されます。データベースのコンテキストでは、SQLベースのリレーショナルデータベースが動的なクラウドインフラストラクチャに適していないことがよく知られています。一方、NoSQLおよびNewSQLデータベースは一見クラウドに対応しているように見えますが、これらのデータベースを本番環境で大規模に実行すると重大な運用上の課題があることが分かります。

このホワイトペーパーでは、特にオペレーショナル・データベースの観点から、このような明確な業界トレンドの原因と影響に焦点を当てます。また、YugabyteDBもご紹介します。YugabyteDBは、現在のミッションクリティカルなアプリケーション向けにゼロから構築された、クラウドネイティブで強力な一貫性を備えた分散SQLデータベースであり、切望されていたクラウドを跨いだ運用のモビリティをデータレイヤーで提供すると同時に、開発者がアプリケーションを非常に容易に開発できるようにします。

オペレーショナル・データベースの進化

アプリケーションのアーキテクチャとインフラストラクチャは非常に大きな変化を遂げましたが、ミッションクリティカルなオンライントランザクション処理（OLTP）アプリケーションを支えるオペレーショナル・データベースもまた、アプリケーションサイドと比べるとゆっくりではあるものの、大きく変化しました。

モノリシックからシャーディング型および分散型へ

リレーショナルデータベースは、**Structured Query Language（SQL）**データベースとも呼ばれ、この40年間、事実上のOLTP標準であり続けています。リレーショナルデータベースは、本質的にモノリシックアーキテクチャであり、特殊な高コストのハードウェアを基盤とする単一ノードで実行されるため、いくつかの問題をはらんでいます。アプリケーションの読み書きの量が増加するに連れて、面倒な垂直方向のハードウェア拡張が必要になります。

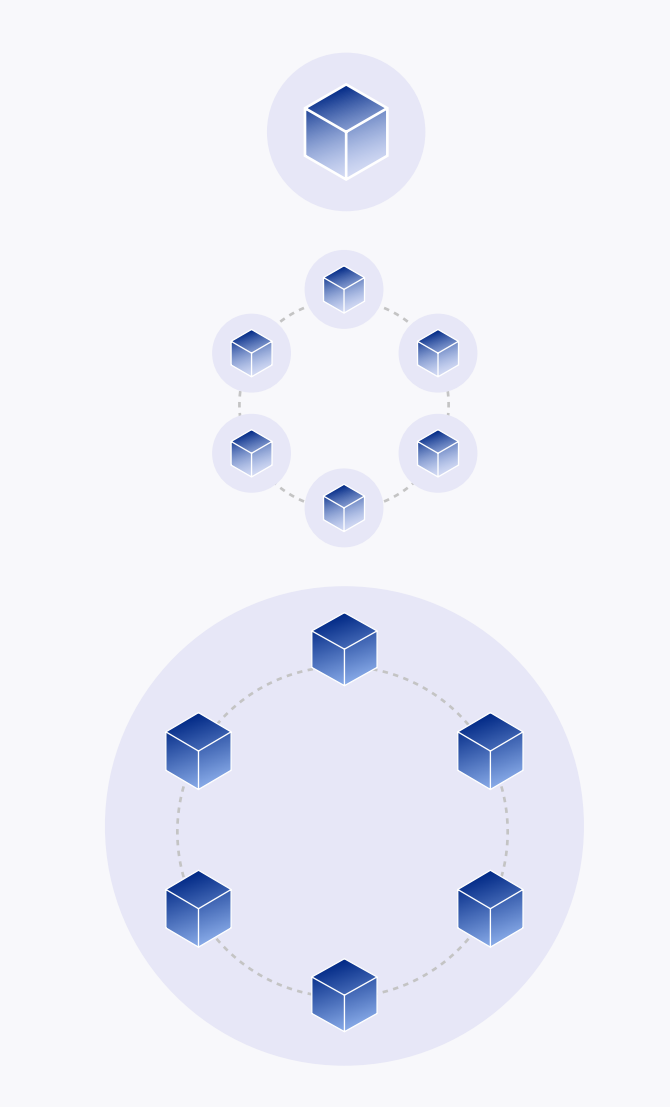
可用性を高めるには、新しい独立したデータベースインスタンスを追加し、非同期または同期レプリケーションを使用してデータをコピーする必要があります。マスター/スレーブとも呼ばれる非同期レプリケーションでは、コミットされたデータがマスターからスレーブに定期的にレプリケートされます。マスターで障害が発生した際、スレーブ側でまだ受け取っていない書き込みがあれば、データの損失が発生する可能性があります。

スレーブへのフェールオーバーの際にダウンタイムが生じるため、可用性も100%ではありません。同期レプリケーションは、2フェーズコミットなどのプロトコルによって実装されるアトミック書き込み操作を使用してデータ損失ゼロを保証します。しかし、このようなシステムは、2つのインスタンスのいずれかがダウンするか、2つのインスタンス間の接続の喪失につながるネットワークの問題が発生すると利用できなくなります。

データが独立したデータベースに分割されるシンプルなシャーディングは、線形拡張性を達成するために追加されるアーキテクチャの強化です。アプリケーションは、どのシャードにどのようなデータがあるかを突き止める責任を負わなければなりません。その結果として生じる複雑さは、特にビジネスニーズの変化に伴ってシャードの数が増加する場合に過小評価される傾向があります。

各シャードの高可用性も、非同期または同期レプリケーションよって保たれます。そのため、運用の複雑さがさらに増すことになります。このようなシャーディング型リレーショナルデータベースは一般に**NewSQL**と呼ばれます。

一方、Not Only SQL（SQLだけではない）を表す**NoSQL**とは、分散型で水平方向に拡張可能であり、シャーディングされ、デフォルトでマルチコピーによってレプリケートされるデータベースのことです。分散型、マルチコピーによるレプリケーションというアプローチは、SQLデータベースとNewSQLデータベースよりも高い読み書きの可用性を提供します。これは、シャードあたりのレプリカの数であるレプリケーション係数を増減させるだけで、許容される障害の数を調整できるためです。NewSQLデータベースは、古い世代のデータベースより高い可用性を提供することを目標に、NoSQLに似た分散アーキテクチャを採用しています。



強力な一貫性から結果整合性へ

有名な**CAP（一貫性、可用性、分断耐性）定理**は、分散システムではネットワーク分割が不可避であるということを説明するものです。そのため、このようなネットワーク分割が生じるたびに、100%の一貫性か、100%の可用性か、いずれか一方のみを選択しなければならなくなります（通常の運用中にこのような妥協を行う必要はありません）。モノリシックリレーショナルデータベースでは分割は問題にはならないため、一貫性と可用性のトレードオフは不要です。実用面に影響するトレードオフは、拡張性の欠如です。シャーディング型のNewSQLデータベースと分散型のNoSQデータベースでは、拡張性の問題は解決されますが、ネットワーク分割が発生したときには100%の一貫性と100%の可用性のいずれか一方を選択しなければなりません。実際には、この選択は、データベースが最適化されている主要なワークロードタイプに基づいて行われます。

NewSQLは、少数のエンティティが狭いコンテキストで処理される（オンラインチェックアウトなど）製品カタログなどの従来型の「System of Record」OLTPアプリケーションに使用されます。したがって、更新直後に表示されるデータがエンティティのすべてのオブザーバにとって一貫している**強力な一貫性**（即時一貫性とも呼ばれます）が求められます。

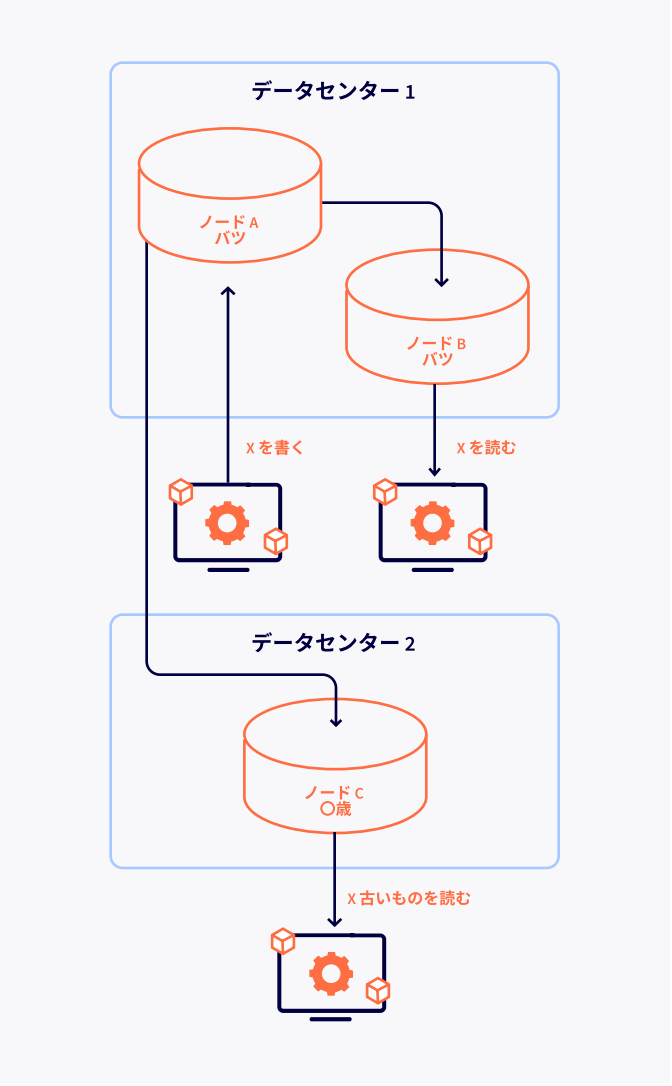
強力な一貫性の基本的な実装は、さまざまな種類の短期間のロックおよび同時実行制御に依存します。これにより、データの中間の状態や不正確な状態は完全に回避されるか、クライアントアプリケーションからは見えなくなります。予想どおり、このようなデータベースは、ネットワーク分割が発生しているときには可用性が低下し、レイテンシーが大きくなります。NewSQLデータベースは、強力な一貫性を維持しながら可用性を大幅に高めることを目指しています。

一方、従来型のNoSQLデータベースは一般に、ハイブリッドトランザクション/分析処理（HTAP）とオンライン分析処理（OLAP）の2種類のアプリケーションのうちの1つに使用されます。

これらのアプリケーションは、どちらも高い書き込み可用性を必要とする一方で、読み取りの古さはある程度許容できます。したがって、**結果整合性**が求められます。結果整合性とは、エンティティに対する新しい更新が行われないことを前提に、エンティティのすべての読み取りが最終的に最後に更新された値を返すという理論的な保証のことです。このような場合、エンティティの総数が多いため、一組のエンティティを含めるか含めないかはユーザーエクスペリエンスに影響しないと考えられます。

この結果整合性に付随する基本的な予測不能性を考慮すると、NoSQLデータベースは、サービスレベル契約（SLA）の保証が必須であるミッションクリティカルなOLTPおよびHTAPアプリケーションには推奨されません。批評家は、結果一貫性は新しい更新を完全に回避するという楽観的で非現実的とも言える要件を前提とする「楽観的」一貫性であると指摘しています。

結果整合性を中核とする従来型のNoSQLデータベースは、クォーラムベースのアプローチを採用し、強力な一貫性に向けた調整をさらに進めています。しかし、このような間に合わせのソリューションには、パフォーマンスの明確な低下と運用およびエンドユーザーの問題の大幅な増加が伴います。当初はNoSQLの早期導入者としての役割を果たした大手テクノロジー企業でさえ、現在は増え続けるエンジニアリング投資を見直さざるを得なくなっています。



ACIDからBASEへ

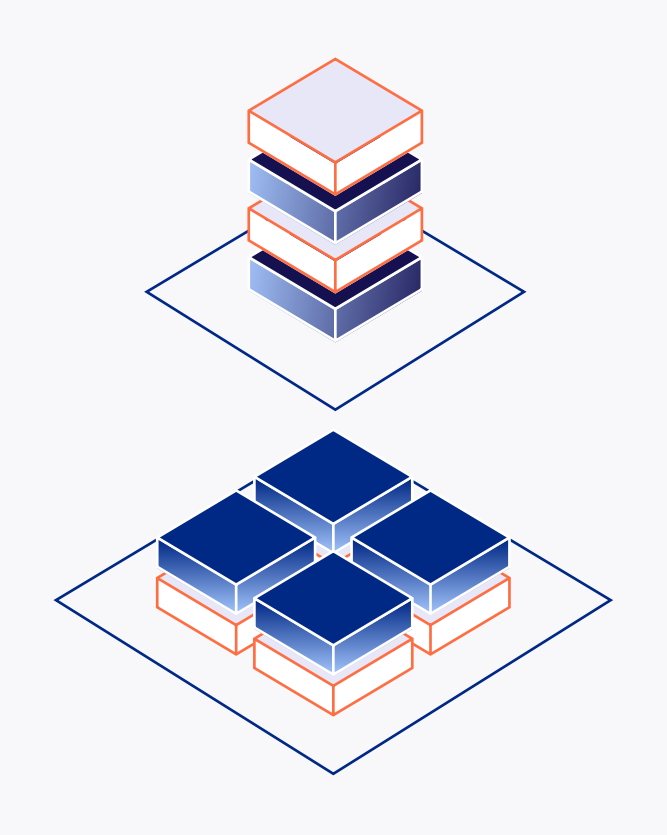
データベースのサーバーサイドの一貫性およびレプリケーションアーキテクチャは、クライアントトランザクションの処理方法に直接影響します。強力な一貫性を備えたリレーショナルデータベースの大規模な導入の背後にある最も重要な理由は、クライアントトランザクションに関する**原子性、一貫性、独立性、永続性（ACID）**保証の概念でしょう。このようなトランザクションでは、開発者がアプリケーションで予想される書き込み/読み取り動作を推測し、（分離レベルを調整することで）簡単に制御することができます。

ACIDトランザクションの背後にあるロックベースの実装アプローチを分散型NoSQLに適用すると、それらのデータベースが長期間利用できなくなります。高可用性が主要なニーズであることを踏まえると、これは受け入れられません。しかし、NoSQLデータベースでは代わりに、データが古いか、ダーティーであっても、読み書きの高可用性が重視される**基本的な可用性、ソフトステート、結果整合性（BASE）**のクライアントトランザクションがサポートされます。分離レベルがREAD UNCOMMITTEDより悪いBASEトランザクションは、特定の時点における記録の正確な値が分からないため、System of Recordでの使用には適していません。

SQLからNewSQLおよびNoSQLへ

上述のサーバーサイドアーキテクチャとクライアントサイドアーキテクチャの違いから、NoSQLデータベースがSQLをクライアントインタラクション言語として採用することが簡単ではないということが分かります。SQLを使用すると、（場合によってはJOINを使用して）複数の行を一度に更新および照会できます。残念ながら、NoSQLデータベースでは、これらの行が同じシャードにあることが保証されず、マルチシャードトランザクションで可用性を減らして原子性を調整するという選択肢はありません。そのため、NoSQLデータベースでは、JOINがサポートされておらず、独自のクライアント言語が用意されています。たとえば、Apache Cassandraには、Cassandra Query LanguageというSQLの亜種があり、MongoDBには独自のCRUD操作があります。

NewSQLデータベースの目的は、JOINを含む従来のSQL構文をできる限り多くサポートすることです。これらのデータベースは基本的に、複数の独立したモノリシックリレーショナルデータベース上で実行される抽象レイヤーです。ストリーミングデータを分析するための、ユースケース固有のデータベースもいくつか存在します。このような非OLTPユースケースにSQLを選択する主な理由は、開発者に人気の高いSQLを活用する必要があるという点です。



リレーショナルからマルチモデルへ

リレーショナルデータベースは、各列が強く型付けされた値を持つテーブルの行として整理される構造化データに適しています。このような厳格なスキーマのアプローチは、半構造化データまたは非構造化データを必要とするアプリケーションには適していません。NoSQLデータベースは、より柔軟なスキーマ（Apache CassandraやApache HBaseなどの列指向のデータベースなど）を提供するか、スキーマレス（たとえば、MongoDBなどのドキュメントデータベース）に向かうことで、この問題を解決します。メモリ内データ構造を持つキーバリューストア（Redisなど）もありますが、そのような永続ストアまたはスキーマの概念はありません。

オンプレミスからクラウドホスト方式へ

分散アーキテクチャと水平拡張を特徴とするNoSQLデータベースは、インスタンスを動的にプロビジョニングできるクラウドホスト方式の展開に適しています。一方、リレーショナルデータベースのモノリシックアーキテクチャと垂直拡張の要件は、そのインフラストラクチャニーズが相対的に見ると動的ではないということを意味しています。リレーショナルデータベースはクラウドでホストできますが、このようなケースには静的なキャパシティを持つオンプレミスのデータセンターが適しています。

Database-As-A-Service（DBaaS）の台頭

ここ数年、主要なクラウドプロバイダーと大手データベースベンダーの両方がDatabase-As-A-Service（DBaaS）ソリューションを導入しています。共通のテーマは、顧客がデータベースの管理や監視という運用の複雑さを懸念する必要がなくなるというものです。今では、運用はプロバイダーの仕事となり、コストはソリューションの価格に含まれています。多くの場合、ソリューションは、Amazon RDS/AuroraやMongoDB Atlasなどのオープンソースデータベースのホスト版です。いくつかの限られたケースでは、クラウドプロバイダーはAmazon DynamoDBなどの独自のサービスとしてのデータベースも提供しています。

データベースにおける「クラウドネイティブ」の定義

クラウドネイティブは、アプリケーションレイヤーにおいては、急速に現実のものとなりつつあります。しかし、アプリケーションを支えるデータレイヤーに関しては、同じことは言えません。クラウドホスト方式のNoSQLおよびNewSQLデータベースは、クラウドインフラストラクチャ上に水平方向に展開できますが、データベースのコア機能を統一的にかつ信頼性を保ちながら、さまざまな異なるクラウドプロバイダーを抽象化することはできません。クラウドネイティブなデータベースの5つの明確な特徴を以下に示します。

- **非常に高い弾力性：**クラスタを迅速かつ確実にスケールアップおよびスケールダウンできます。
- **地理的に冗長でAlways-onの可用性：**マルチAZ/マルチリージョンのクラスターを簡単に作成し、その後いつでも、計画外の障害と計画されたアップグレードへの復元力を維持したまま、AZまたはリージョンを拡張または縮小することができます。
- **ハードウェアの柔軟性：**コストとパフォーマンスの理由からコンピューティングとストレージのタイプを切り替える必要がある場合には、それをシームレスに行うことができます。
- **マルチクラウドモビリティ：**別のクラウドプロバイダーに移行するか、複数のクラウドプロバイダー上に共存することで、特定のクラウドプロバイダーからのロックインを回避できます。
- **データ配置ポリシー：**アプリケーションに影響を与えることなく、地域固有のデータレジデンシーコントロールを定義して適用できます。

DBaaSソリューションは、基盤となるクラウドインフラストラクチャをぼかして、「クラウドネイティブ」と称して宣伝されていることに注意してください。このようなサービスには、特定のクラウドベンダーのロックインや保有するデータおよびサーバーの制御不能が内在しているため、企業はこれらのサービスを真の意味でクラウドネイティブとは考えていません。また、従来のNoSQLやNewSQLデータベースをステートフルコンテナ（KubernetのStatefulSetsなど）に展開することは、特定のクラウドプロバイダーのロックインの問題は解決されますが、それはアジャイルな開発と運用という最終目標の達成には役立ちません。これは、これらの従来型データベースがクラウドネイティブな基盤の上で構築されたものではないからです。これらのデータベースのデータストレージおよびレプリケーションには、オーケストレーション対応のステートフルコンテナの能力を最大限に引き出すためになくてはならない可用性とモビリティが欠如しています。

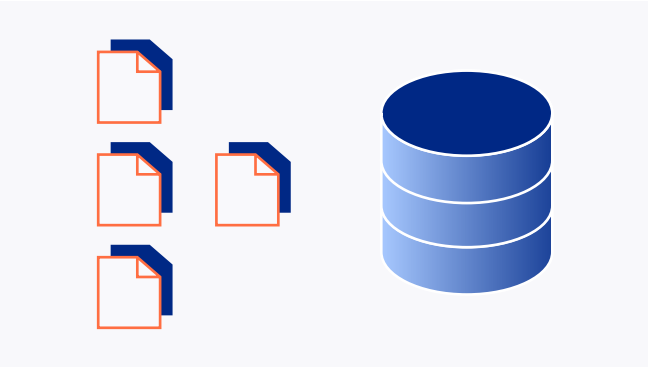
NoSQLデータベースの3つの課題

このセクションでは、現世代のNoSQLデータベースの3つの課題である運用の複雑さ、フラストレーションのたまるアプリケーション開発、一貫性のないカスタマーエクスペリエンスについて説明します。

運用の複雑さ

前のセクションで述べたように、データベースはクラウドでホストされるように進化しましたが、真のクラウドネイティブからはまだ程遠い状態にあります。現在の運用の複雑さは、最新のクラウドインフラストラクチャの復元力と地理的冗長性を最大限に活用したいと考えている組織にとって厄介な問題です。さらに、NoSQLの中核的要素である結果整合性を追求すると、パフォーマンスを低下させるバックグラウンドでの修復機能や、予測不能でメモリにかかる負荷が高いコンパクションの大量発生といった隠れたコストが増加することになります。事実、ほとんどの企業が永続データベースと一緒に独立したキャッシュ層（RedisやMemcacheなど）も実行していることから、すべての運用上の課題も倍増する結果となります。

データレイヤーをアプリケーションレイヤーと同じフェーズを経て同じ速度で移行するという当初のニーズに照らして考えると、上記の運用上の課題によってこのような移行はほぼ不可能となってしまいます。運用チームがこのような移行を押し付けられた場合、予測不能なダウンタイムや、運用上の人為的ミスが発生しやすい状況など、ビジネス上の損失が生じることになります。



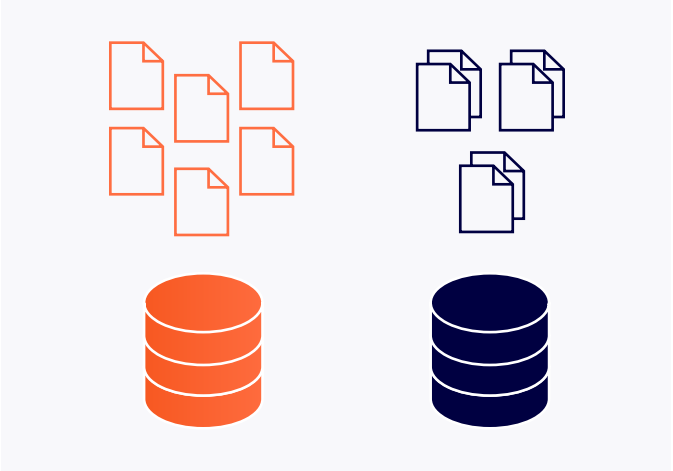
フラストレーションのたまるアプリケーション開発

アプリケーション開発者は、データベースクライアントコードの読み取り/書き込み動作を簡単に推測できるように、ACIDトランザクションのシンプルさを求めています。リレーショナルデータベースでは、関連する複数の行を全か無かの一貫した方法で更新または読み取ることができる、複数行のACIDがサポートされます。しかし、ほとんどのNoSQLデータベースでは、単一行のACIDトランザクションすらサポートされていません。結果整合性によって「C」がリモートレプリカで妥協されるからです。

その結果、開発およびテストサイクルが非常に長くなります。望ましい読み取り/書き込み動作を達成するため、めったに発生しない厄介なケースにも漏れなく対処する必要があるためです。たとえば、「失敗した書き込みの後にどのような値が読み取られるか」という質問に対する答えはNoSQLデータベースによって大きく異なります。さらに、ほとんどのNoSQLデータベースでは、セカンダリインデックスがサポートされないか、（すべてのシャードにアクセスすることの直接的な結果として）セカンダリインデックスのパフォーマンスが極端に低くなります。この重要な機能の欠如も開発の遅延につながります。複雑な次善策を設計、実装、テストする必要があるからです。

多くのNoSQLデータベースは、強力な一貫性のメリットを実現し始めており、結果整合性を備えたシステムをクォーラムベースの強力な一貫性設定に調整できるようにしています。しかし、この形態での調整可能な一貫性は、失敗した書き込みの後のダーティーリードや最終書き込み者が優先された後の予測不能な読み取りなどの多くの状況において、本当の意味では解決策にはなり得ないことは明白です。このアプローチを採る開発者は、動作を保証するため、アプリケーションのテストにさらに多くの時間を費やさざるを得なくなります。

上記すべての課題に加えて、独立したインメモリ・キャッシュ層と基盤となる永続データベース層との一貫性を保つという課題もあります。キャッシュの無効化とキャッシュの生成については、パフォーマンスの低下を回避するためにアプリケーションで慎重に処理する必要があります。最後に、ほとんどのデータベースは、柔軟なスキーマ、キーバリュー、時系列、グラフなどの特定のアプリケーションのニーズにしか適していません。そのため、継続的に新しいデータベースを評価し、それらのデータベースをデータ層に組み込むという面倒な作業は、開発者が担うことになります。



一貫性のないカスタマーエクスペリエンス

開発者は、結果整合性を備えたNoSQLデータベース上に強力な一貫性を備えたOLTP/HTAPアプリケーションを構築する目的で、あらゆるチューニング作業を行います。このような開発者の尽力にもかかわらず、エラー発生は避けられません。そのエラー状態にある間は、エンドユーザーも一貫性の欠如の影響を受けることになります。

たとえば、いくつかの品目が在庫切れのために小売業者の商品カタログから削除されたとしましょう。しかし、データが顧客に提供されたとき、それらの削除はデータに反映されていませんでした。そのデータの提供元となったノードで削除がまだ適用されていなかったためです。 別の例として、時系列モニタリングおよびモノのインターネット（IoT）ユースケース向けのアラートにおいて、集計値の計算に使用する時系列メトリックデータを無視してしまうケースも挙げられます。不正確なデータに基づいてチームメンバーを早朝に叩き起こすという事態は避けなければなりません。同様に、ユーザーのプライバシー設定が即座に反映されない場合、そのユーザーの行動が同じアカウントの他のユーザーに表示される可能性があります。

分散SQLでNoSQLの課題を解決する

分散SQLデータベースは、サーバーのクラスターに展開される単一の論理リレーショナルデータベースです。このデータベースは、データを複数のサーバーに自動的にレプリケートおよび分散します。これらのデータベースは、強力な一貫性を備えており、クラウドのアベイラビリティ・ゾーンと地理的ゾーンにわたって一貫性をサポートします。

分散型SQLデータベースは少なくとも以下の特性を備えています。

- データとオブジェクトにアクセスして操作するためのSQL API
- クラスターの複数のノードへのデータの自動分散
- 強く一貫した方法でのデータの自動レプリケーション
- クライアントが基盤となるデータ分散について知らなくても済むようにするための分散クエリ実行のサポート
- 分散ACIDトランザクションのサポート

分散SQLを選ぶ理由

従来のSystem of Recordはビジネスイノベーションの圧力を受けています。これにより、企業はITコストを削減し、コンプライアンスによってリスクを軽減しながら、高価値のアプリケーションとサービスを提供する必要に迫られています。

しかし、マイクロサービス、クラウドネイティブアプリケーション、エッジおよびIoTワークロードという形のこれらのアプリケーションは、以下のような特性を備えた新しいクラスのデータベースを必要とします。

耐障害性に優れ、継続的に利用可能：高速フェールオーバーにより、ノード、ゾーン、リージョン、データセンターで障害が発生したときやシステムメンテナンスを実行しているときでも重要なサービスを継続的に利用できる

水平方向に拡張可能：運用チームが、重い負荷がかかっているときでもクラスターにノードを追加するだけで、ダウンタイムなしで簡単にスケールアウトでき、負荷が軽くなったときにスケールインできる

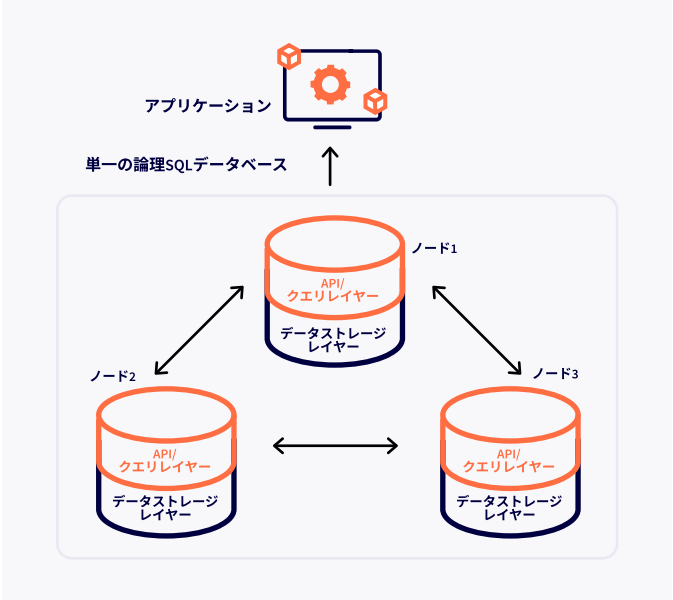
地理的に分散：オペレーターが同期および非同期のデータレプリケーションと地理的分割を活用し、地理的に分散された構成でデータベースを展開できる

SQLとRDBMS機能の互換性：開発者がクラウドネイティブシステムの水平方向の拡張性と従来のRDBMSのACID保証および強力な一貫性のいずれか一方のみを選ぶ必要がなくなる

ハイブリッドおよびマルチクラウド対応：組織が場所を選ばずデータインフラストラクチャを展開して実行し、特定のクラウドプロバイダーにロックインされるのを回避できる

分散SQLデータベースのアーキテクチャ

分散SQLデータベースは、従来のRDBMSの長所とクラウドネイティブなデータベース機能を兼ね備えています。単一の論理SQLデータベースの一部として2つのアーキテクチャレイヤーを備えています。



SQLクエリレイヤー

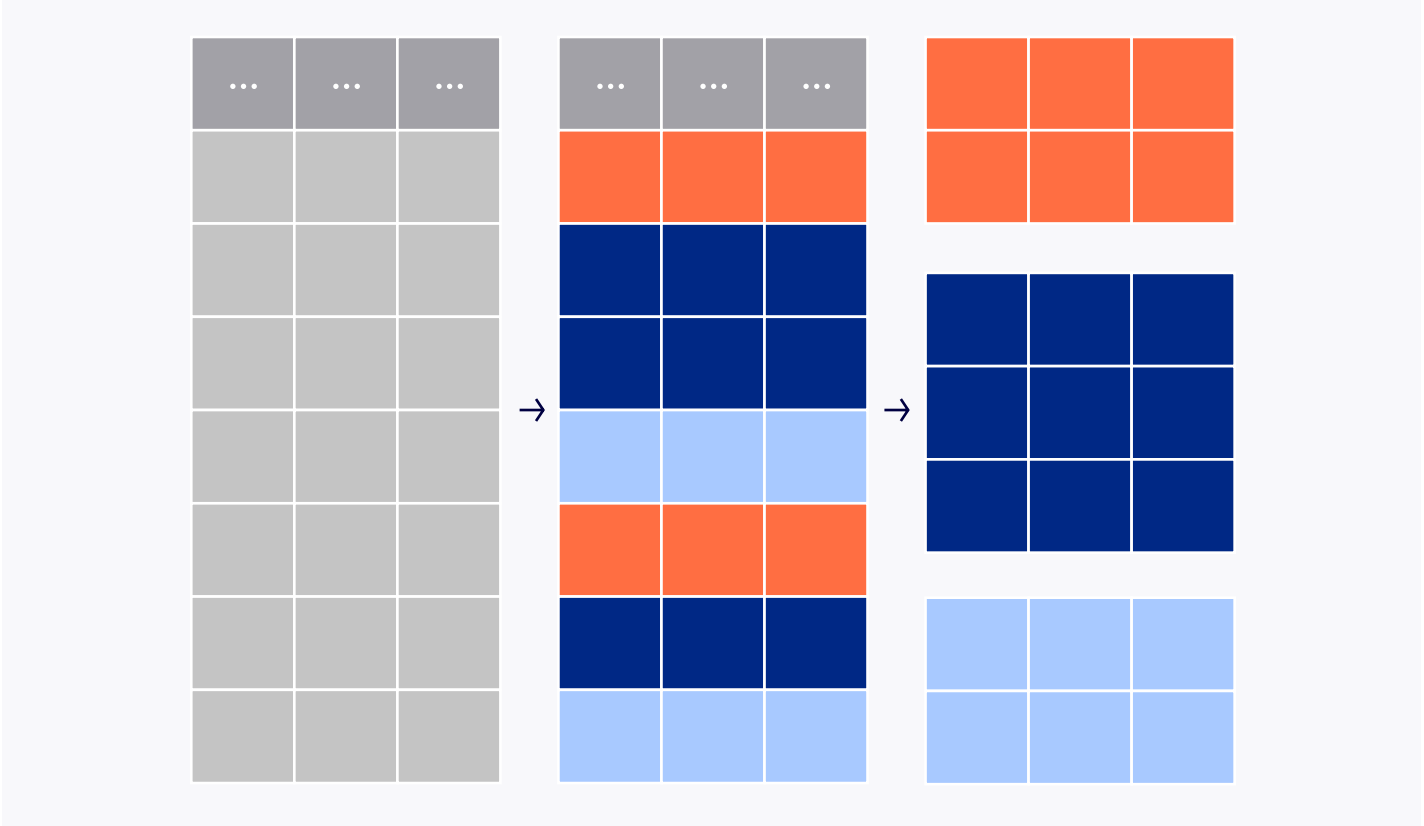
分散SQLデータベースは、アプリケーションがリレーショナルデータをモデリングし、それらのリレーションに関連するクエリを実行するためのSQL APIを備えています。クエリは、データベースクラスターの複数のノードに自動的に分散されます。

分散データストレージレイヤー

分散SQLデータベースのデータ（インデックスを含む）は、クラスターの複数のノードに自動的に分散（またはシャーディング）されます。そのため、1つのノードがハイパフォーマンスと高可用性のボトルネックになることはありません。

強力なSQL APIレイヤーをサポートするには、基盤となるストレージレイヤーが、クラスターのすべてのノードにわたる**強く一貫したレプリケーション**に基づいて構築されている必要があります。これは、障害発生時の可用性を保証するためにデータベースへの書き込みが複数のノードで同期的にコミットされることを意味します。

そして最後に、データベースストレージレイヤーは、複数のノードにある複数の行にわたってトランザクションの調整が必要になる**分散ACIDトランザクション**をサポートします。



YugabyteDB：クラウドネイティブなデータベース向けのクラス最高の分散SQL

YugabyteDBは、トランザクションアプリケーション向けのクラウドネイティブな分散SQLデータベースです。このデータベースは100%オープンソースで、Kubernetesアプリケーションワークロードを実行するときの可用性と復元力の課題を解決できるように設計されています。運用のシンプルさ、開発者の生産性、顧客満足という3つの基本原則を念頭に置いて設計されています。

このデータベースは、高可用性を確保するためにレプリカを使用し、**Raftプロトコル**を使用することで同期をサポートします。その他の機能には、拡張性を確保するための分割（タブレットと呼ばれます）があり、タブレット間トランザクションの場合は、2フェーズ・コミット・プロトコルも実装されます。

YugabyteDBは、人手を介すことなく、SQLテーブルをタブレットに分割します。また、データ損失ができる限り発生しないように、タブレットレプリカを構成済みの障害ドメインに自動的に分散します。この**動作は、ユーザーの障害ドメイン、レプリケーション係数、障害ドメインに対するデータベースアフィニティ**の設定によって変化する場合があります。

導入の柔軟性

YugabyteDBは、パブリック、プライベート、およびハイブリッドクラウド環境、VM、コンテナー、またはベアメタルで動作します。組織は任意のインフラストラクチャ環境にこのデータベースを導入することができます。このデータベースは、スムーズなエクスペリエンスを実現するマルチクラウドフルマネージド型のサービスとしてのデータベース（DBaaS）として利用することもできます。YugabyteDBは、分散SQLデータベースの中で最も広範なレプリケーションおよび地理的分散オプションを提供します。

ハイパフォーマンス

YugabyteDBは、高スループット・低レイテンシートランザクションに対処できます。1秒あたり100万トランザクションと数千の同時接続に拡張できることが本番環境で実証されています。

運用のシンプルさ

組織は、YugabyteDBのセルフマネージド型またはフルマネージド型のDBaaSサービスを使用して、エッジおよびクラウドでの運用をシンプル化できます。このデータベースはまた、その他のデータソースと統合またはシンクすることができます。これにより、データエンジニアは機械学習、分析、長期保存のストレージ、災害復旧のためのデータ・パイプラインを構築できるようになります。

マルチ・クエリ・モデル

PostgreSQLとCassandraの両方のAPIをサポートするプラグイン可能なクエリレイヤーで、データベースの無秩序な肥大化を排除します。

本質的な地理分散

ACID一貫性を保ちながら、ゾーン、リージョン、クラウドにまたがるデータ分散を実現。最も完全なグローバルレプリケーションを実現します。

PostgreSQLとの互換性

YugabyteDBはPostgreSQLと接続の互換性があるだけでなく、コードの互換性もあります。このデータベースはまた、トリガー、関数、ストアドプロシージャ、強力なセカンダリインデックスなどの一連の包括的で高度なRDBMS機能を備えています。このため、開発者は使い慣れたインターフェイスとPostgreSQL互換のフレームワーク、アプリケーション、ドライバー、ツールからなる充実したエコシステムを活用して即座に生産性を高められます。

セキュリティ

YugabyteDBは、セキュリティを念頭に置いてゼロから構築されており、組織がより分散化された環境でも堅牢なセキュリティ体制を維持することを可能にします。YugabyteDBは、保存中および移動中のデータの暗号化、データベースレイヤーでのマルチテナンシーのサポートとテナントごとの暗号化、コンプライアンスを確保するための地域局所性、地理的なアクセス制御の管理などの機能を備えています。

機能の比較

最後に、YugabyteDBとこのホワイトペーパーで先に説明したその他の運用データベースを比較します。

機能	従来型のSQL	従来型のNewSQL	従来型のNoSQL	YugabyteDB
スキーマ	リレーショナル	リレーショナル	マルチモデル	マルチモデル
言語	SQL	SQL	カスタム	SQLおよびCQL
拡張性	垂直	水平	水平	水平
一貫性	強力	強力	結果的（調整可能）	強力
レプリケーション	構成可能	構成可能	自動	自動
可用性	低	低	高	高
トランザクション	ACID	ACID	BASE	ACID
キャッシング	独立	独立	独立	組み込み
クラウドインフラ	クラウドホスト方式	クラウドホスト方式	クラウドホスト方式	クラウドネイティブ
ターゲット アプリケーション	OLTP	OLTP	HTAP, OLAP	OLTP, HTAP

まとめ

企業は、マイクロサービスおよびコンテナーから得られるものと同等のクラウドネイティブな機敏性を提供する、マルチモデルでエンタープライズグレードのOLTP/HTAPデータレイヤーを必要としています。しかし、現世代のソリューションはこのニーズに対応しきれていません。SQLおよびNewSQLデータベースは、強力な一貫性によって開発者の生産性を高めますが、クラウドネイティブな特長とマルチモデル機能がないため、運用の複雑さが大幅に増えることになります。一方、クラウドでホストされ、結果整合性を備えた、OLTP/HTAPユースケース向けのNoSQLデータベースを使用すると、開発者の生産性が低下するだけでなく、運用の複雑さも増加します。

YugabyteDBは、開発者と技術的運用チームをここで挙げたような長年の辛い妥協から解放します。このデータベースは、スケールアウト、クラウドネイティブ、強力な一貫性を中核に構築されており、開発者が一般的なNoSQL言語を顧客向けOLTP/HTAPアプリケーションのコンテキストで活用できるようにすると同時に、簡単かつ柔軟な運用管理が可能なデータレイヤーを提供します。

連絡する
yugabyte.com | contact@yugabyte.com