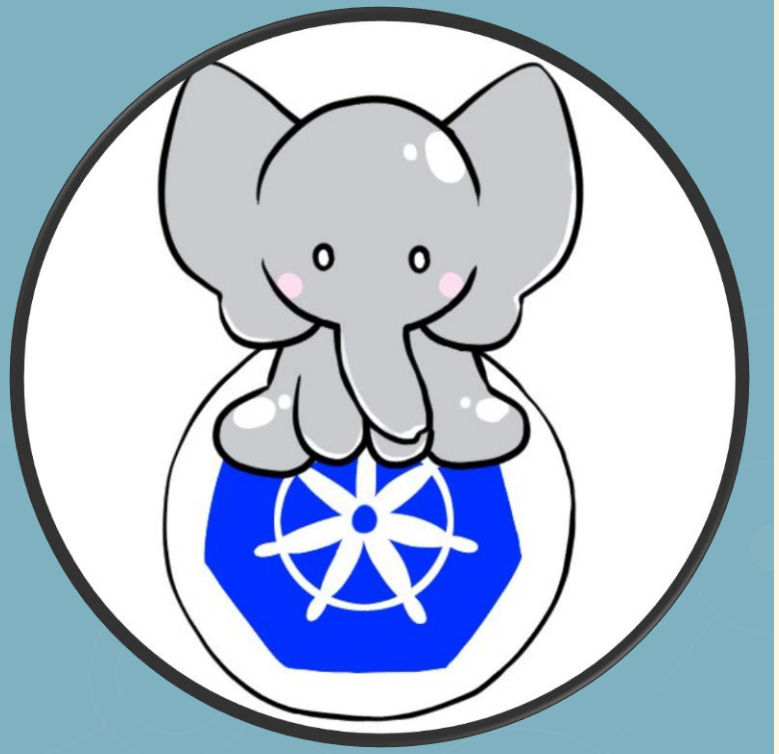
NewSQLへの誘い

Cloud Native Database Meetup #1 , 2021/7/16

こば(@tzkb)



最近やっていること

- 色んなDBをつまみぐい、@ITで連載中

アジェンダ

1. そもそもNewSQLとは？
2. ストレージエンジンの観点から
3. 分散トランザクションの観点から
4. 更に詳しく知りたい方へ

1　そもそもNewSQLとは？

NewSQLへの流れ

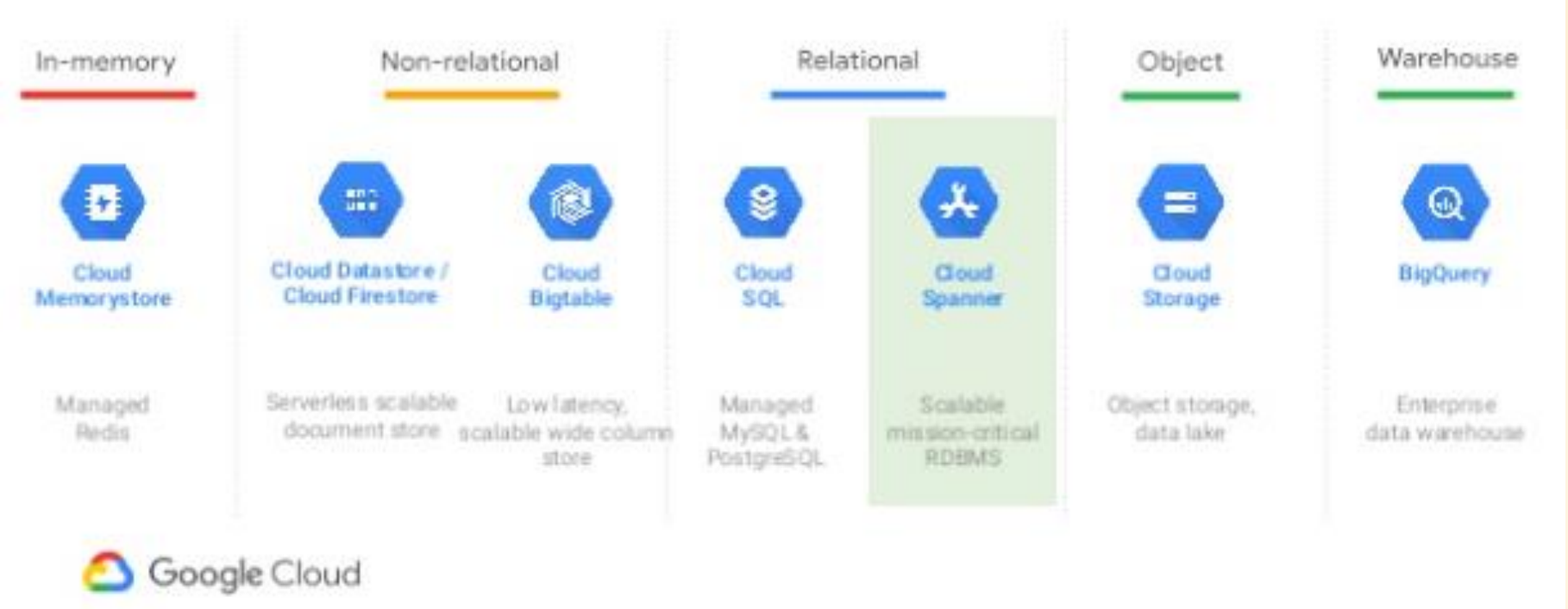
- 2000年代初め、DBは**プロプライエタリのみ**でOSSは無かった。
  単一ノードで稼働、垂直スケールのみ。

- 2000年代後半、**NoSQL**が勃興。高い可用性と拡張性を併せ持つ。
  リレーショナルではなく、ACID txも保証されないものが多かった。


- 2011年のAslett Reportで**NewSQL**の用語が登場、NoSQLの
  ような拡張性と柔軟性を持ち、SQLとACID txをサポート。

Spannerの登場

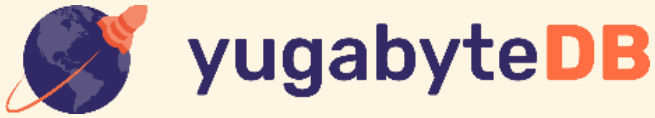
- **Spanner: Google’s Globally Distributed Database(2012)** を発表。
- SQL準拠、**ACIDトランザクションをサポート**。
- **地理分散**（例：東京ー大阪ーソウル）が可能な、分散SQLデータベース。
- 従来のRDBが弱いとされた、**Writeもスケールアウト**が可能。



NewSQL市場の転換と拡大

- 2011年頃のNewSQLは、Spannerとは異なるアーキテクチャ。
- VoltDBやNuoDBなど、NoSQLが解決できなかった課題（マルチマスタ・スケールアウト型のRDB）の解決を目指していた。

- そして、Google Cloud Spanner以降、類似実装のOSSが誕生する。



- その特徴は、
  - **強い整合性**を持ち、
  - **ACIDトランザクションをサポート**する、
  - 地球規模に展開可能な、**分散SQL**データベース

2 ストレージエンジンの観点から

一般的なRDBMSのコンポーネント

← SQL文を構文解析して、

← 実行計画(プランツリー)をつくり、

← プランツリーに沿ってクエリを実行する

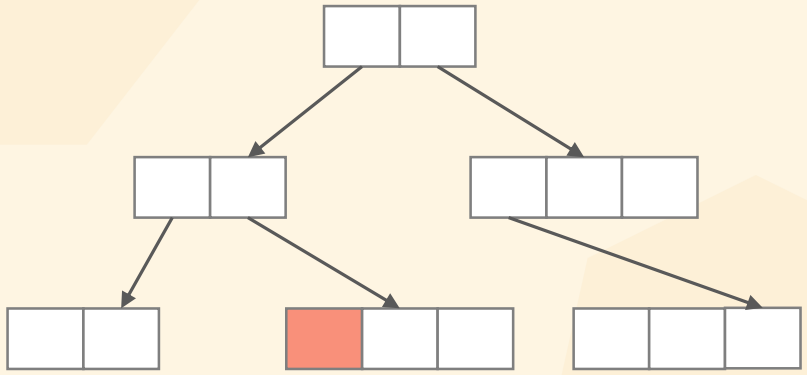
← この辺を**ストレージエンジン**と言ったりする。
← データ管理の中心的役割を果たす。

ストレージエンジンも時代とともに変わる

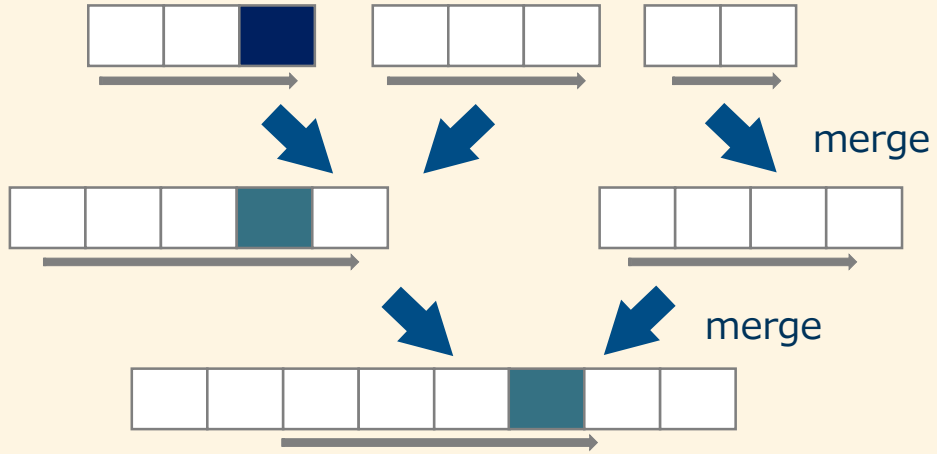
- **B+Tree**：RDBMSで長年使われてきたストレージ構造
  上書き型のデータ構造で、Read Amplificationを抑制できたが、WriteやSpaceの効率が良いとは言えない。

- **LSM-Tree**：Immutableでシーケンシャルな書き込みを行う構造
  Write/Space Amplificationを抑えるストレージ構造として、RDBMSでも採用されつつある。例：MyRocks、Cloud Spanner、TiDB等

【B+Tree】



【LSM-Tree】



ストレージエンジンの特性を理解する

- ストレージエンジンを比較する際に、Read/Write/Spaceの観点が重要。
- 3つ全てを達成することは難しいため、どこかを諦める必要が生じる。

【特徴】

	Read Efficiency	Wrire Efficiency	Space Efficiency
B+Tree	**Best**		
LSM Tree(Leveled)			**Best**
LSM Tree(Tiered)		**Best**	

- TiDB（TiKV）、CockroachDBでは RocksDB を利用している。

- YugaByteDBは独自開発のDocDBを用いる。
- なお、CockroachDBは独自ストレージエンジン（Pebble）への転換を進めており、このレイヤでの競争が激化している。

NoSQLでの変化はNewSQLにも

- どんな高度な分散データベースも、データはノードローカルなストレージに貯められるケースが多い。

- 従来のRDBMSではReadが優先され、**B+Tree**のストレージ構造を使ってきた。

- CassandraなどのNoSQLを中心に、**LSM-Tree**の構造が使われることが増えてきた。これはWriteに強く、今主流のSSDとの相性も良い。
- そうした背景から、大規模にスケールアウトさせるNewSQLも、LSM-Treeを採用しつつある。圧縮が効くため、Space効率でも有利。

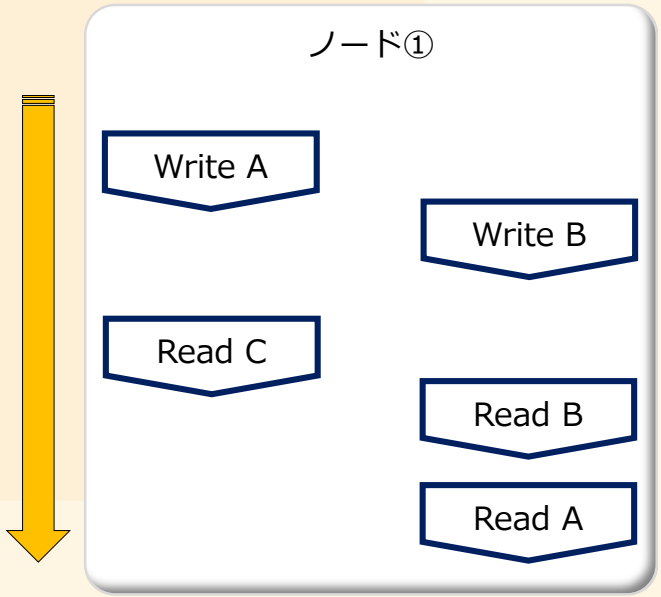
- CockroachDBやTiDBでもLSM-Treeの実装（RocksDBやPebble）を使っている。

3　分散トランザクションの観点から

分散データベースの課題

- 可用性または拡張性を求めて、データベースを分散すると始まる
トランザクションとの戦い。

【シングルノードの場合】



トランザクションを順序通りに
並べることは簡単。

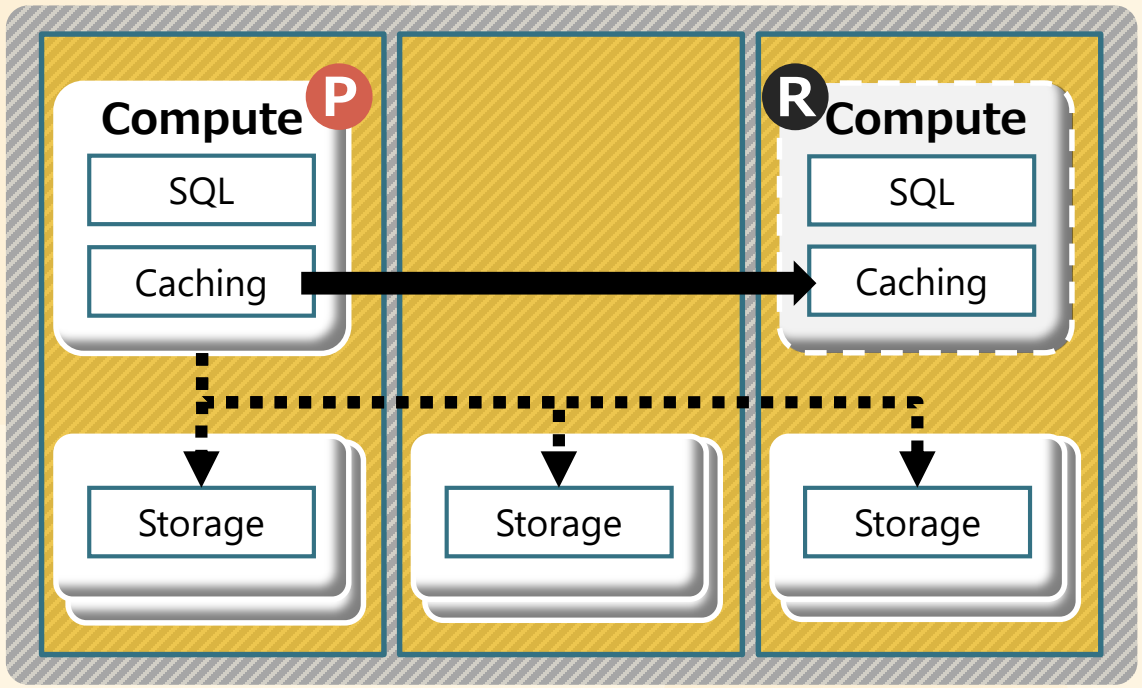
【マルチノードの場合】

ノード間の時刻は厳密には一致しない。
トランザクションを時系列で並べることが難しい。
他ノードをブロックすれば直列化できるが、スケールしない。

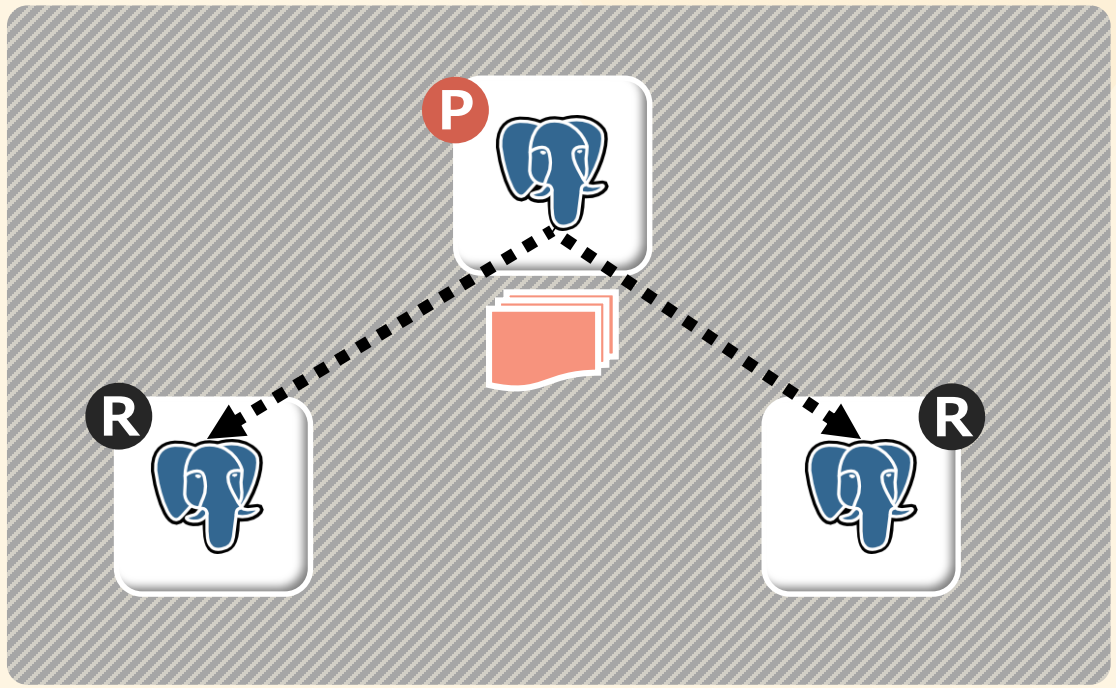
（参考）シングルリーダーなら簡単

- マルチノードであっても、ReadもWriteも単一のリーダーが行う構成ならば、トランザクションの問題は解決可能。
- しかし、リーダーがボトルネックとなるため、拡張性が十分でない。

【Amazon Aurora】

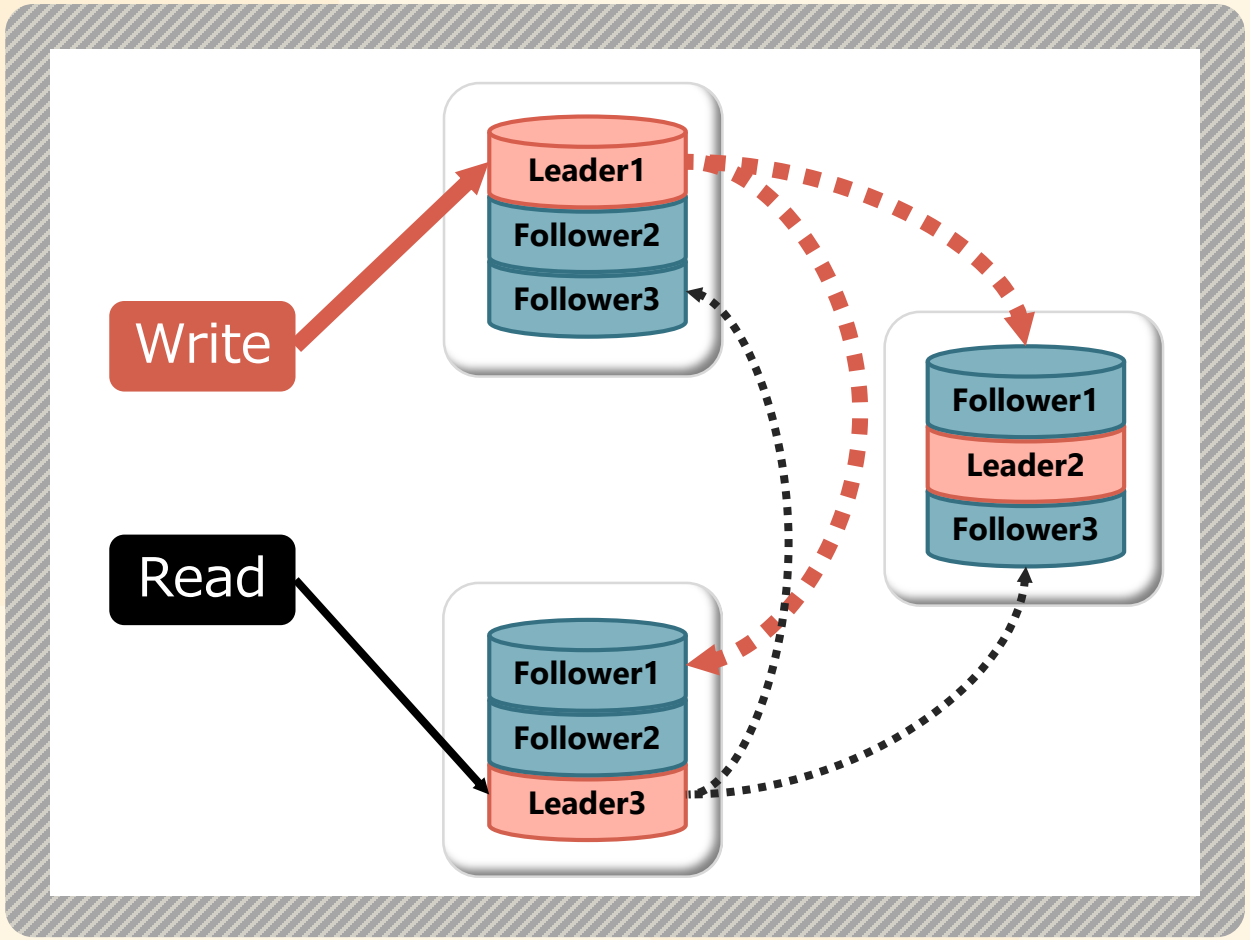


【PostgreSQLのReplication】



Raftと単一パーティションの更新

- データをパーティション化して**マルチリーダー**に、それぞれのレプリカを**Raft**(合意プロトコル)を用いて同期する。



【Write Path 】 ⟶ ⇢

①WriteはLeaderに送られ、Leaderのlogに更新内容が記録される。
②全てのFollowerにlogを複製。
③Followerの**過半数**から複製済の応答が返る。
④Leaderは更新をコミット。

【Read Path 】 ⟶ ⇢

①Readも原則はLeaderへ送られる。
②LeaderはFollowerへハートビートを送信し、**過半数**からの応答を待つ。
③自身がLeaderあることが確認できたので、Read結果を返す。

４　更に詳しく知りたい方へ

（宣伝です）NewSQLを学ぶなら！

- 先ほどの図は単一パーティションのWrite/Readだったが、現実では複数パーティションを1TxでWriteすることも多い。
- じゃ、2フェーズコミット？色々むずかしいよね、、、
- そんなあなたに！

- 第Ⅰ部 **ストレージエンジン**
- 第Ⅱ部 **分散システム**

※電子版はオライリーのサイトから

Questions?

 @tzkb

 @tzkoba