月間動画再生数約 5 億回を誇る TVer の、広告配信基盤における Memorystore & Bigtable 併用戦略と実践的チューニング

Google Cloud Next Tokyo

髙品 純大

株式会社 TVer
広告プロダクト本部

contents

01. TVer と TVer 広告

民放公式テレビ配信サービス

広告型の完全無料でお楽しみいただける民放公式テレビ配信サービス

月間再生数 4.96 億回

月間動画再生数は最高 4.96 億回
MUB は最高 4120 万

src:
https://tver.co.jp/news/20250109.html
https://tver.co.jp/news/20250210.html

月 800 番組以上

ドラマ, バラエティ, アニメ, スポーツ,報道, ドキュメンタリーなど幅広いジャンルで常時、月 800 番組以上配信

TVer 広告

受容性が高い動画広告

テレビ CM と同様に自然な
タイミングで動画広告が挿入
されるため、広告の違和感や
嫌悪感が低い。

1st party data による ターゲティング

ユーザーから直接取得する
1st Party Dataと、TVer 独□の
ターゲティングデータを保有
しており、データドリブンな
広告配信が可能。

ブランドセーフティ

権利処理済かつ JIAA(□般社団法□ □
本インタラクティブ広告協会)の
ブランドセーフティ基準を満たした
安□・安全なコンテンツにのみ配信。

02. 広告配信と NoSQL DB

02. 広告配信と NoSQL DB

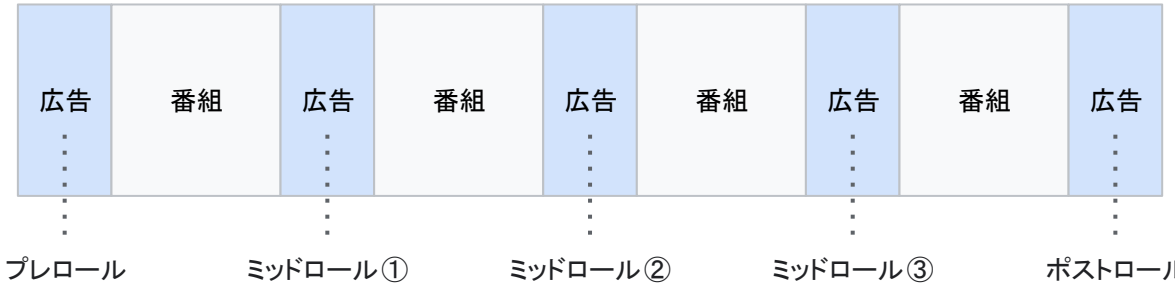
Key-Value データベース を選んだ理由

動画広告配信・視聴計測

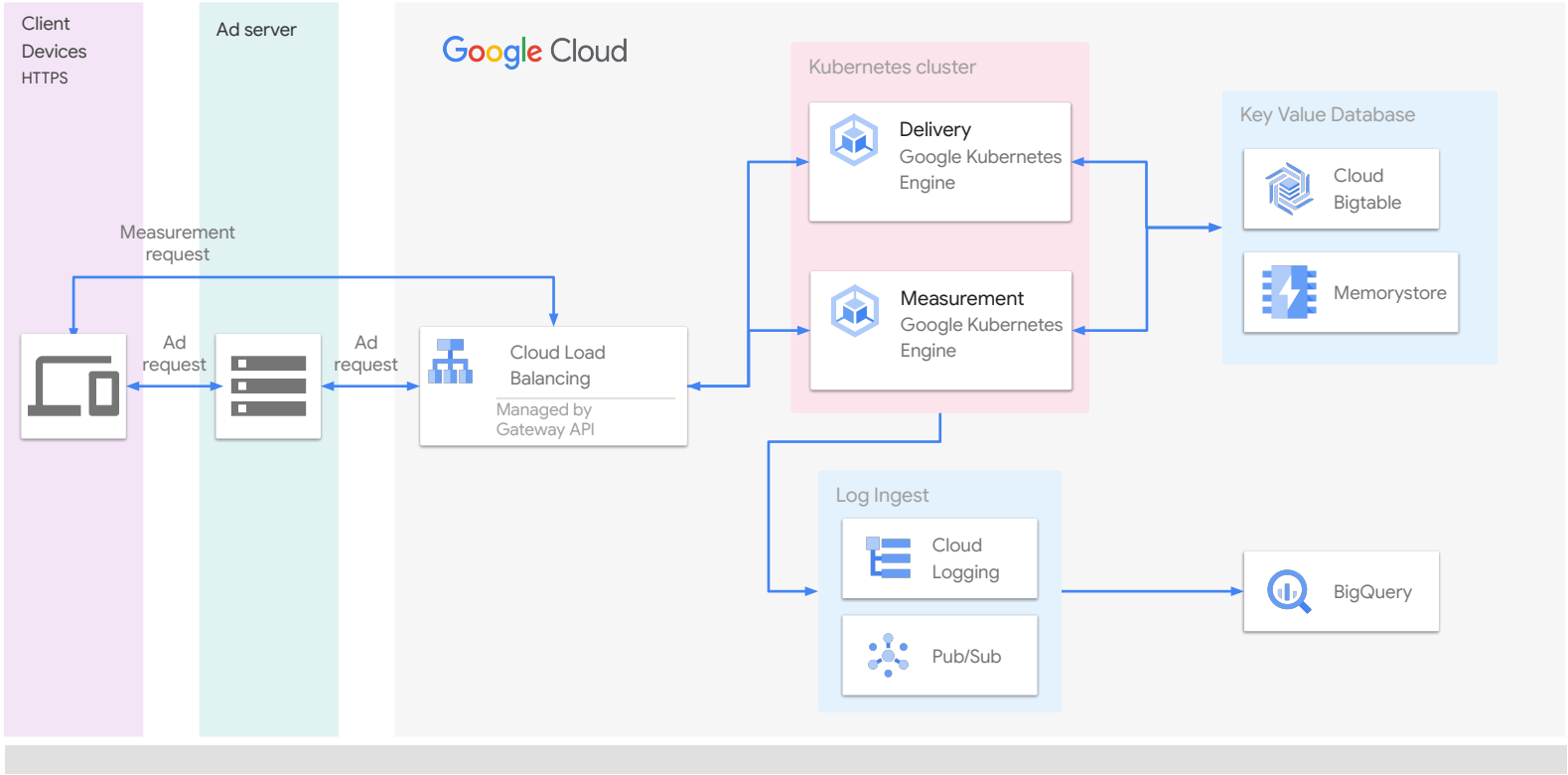
TVer における広告配信の流れ

1. アドサーバーに広告リクエストを送信する
2. アドサーバーが、VAST 形式の広告レスポンスを返す
3. VAST レスポンスに基づいて、広告クリエイティブが再生される
4. 広告再生中に、各種計測用のビーコンが計測サーバーに送信される

VOD の番組再生における典型的な広告挿入タイミング
src: https://biz.tver.co.jp/



Architecture: TVer Ads Delivery & Measurement System



Key-Value データベース を選んだ理由

広告配信基盤に求められる以下の非機能要件を満たすため

- レイテンシ
- 拡張性

広告配信サーバーのレイテンシ

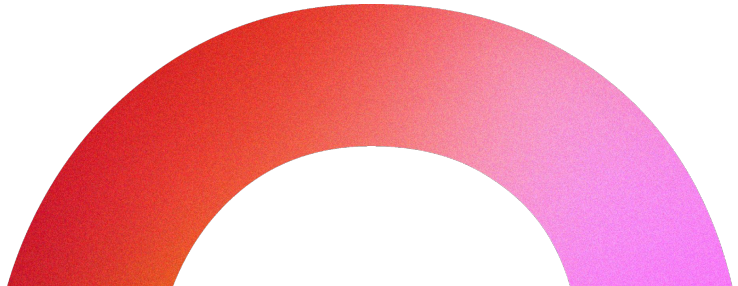
- 前段アドサーバーの数百ミリ秒のタイムアウト以内に広告を返さなければならない
- このとき、最適な広告を選び出すために RDB に複雑なクエリを発行していると間に合わない
- したがって、広告選択にとって最適なアルゴリズム・データ構造を使ってアプリケーション側で計算する必要がある
- データベースに求めるのは、広告選択に必要なデータにミリ秒のレイテンシでアクセスできること

事業の伸長を支える拡張性

- MUB, 動画再生数, 広告出稿数などの事業指標が伸長することでトラフィックが増加しても、レイテンシの要求は緩まない
- 合理的なコストでスケールするアーキテクチャを考えなければならない
- 水平方向へ拡張する分散データベースが望ましい

02. 広告配信と NoSQL DB

広告配信サーバーのレイテンシとシステムの拡張性の要件を満たすには、Key-Value データベースが最適



03. MRC & Bigtable 併用戦略

03. MRC & Bigtable 併用戦略

- Key-Value データベース 選定プロセス
- Memorystore for Redis Cluster (MRC) と Bigtable をどのように使い分けているか

Key-Value データベース選定プロセス

1. Data Design

広告配信・広告視聴計測に
必要なデータを設計する。

- データ構造
- 概算データ量
- 格納場所
- キー設計

2. Implementation

広告配信・視聴計測サーバーのプロトタイプを実装する。

3. Benchmark

Google Cloud にデプロイしたプロトタイプに対して、
K6 で性能評価を実施する。

Data Design

Description	Key Space * Size of Value	QPS (Read & Write)	Note
配信設定	小	数万 QPS	• 配信可能な案件とその設定 • キー空間が狭く、特定のキーにアクセスが集中するため、ホットキー対策が必須
予算消化	小	同上	• 各案件の予算消化状況 • 配信設定と同じ特性に加えて、リアルタイム性を要求
広告視聴セッション	大	同上	• 広告視聴ごとのユニーク ID • キー空間が広く、全体サイズが大きい、リアルタイム性を要求
フリークエンシー (FQ)	大	同上	• ユーザー ID * 広告 • キー空間が広く、全体サイズが大きい、リアルタイム性を要求
ユーザーセグメント	極大	同上	• ユーザーのセグメント（デモグラフィックデータ） • キー空間が広く、全体サイズが特に大きい

Google Cloud のKey-Value データベース

Name	Note
Memorystore for Memcached	● シンプルなデータ構造（Value は 事前にシリアライズされたデータ） ● 最大 20 個のノードを起動可能、CPU を 1 ~ 32, メモリを 1 ~ 256 GB の範囲でサイジング可能 ● **データの耐久性は低い**
Memorystore for Redis	● 高度なデータ構造 (lists、sets、sorted sets、hashes、bit arrays、hyperloglogs) ● 最大 5 つのリードレプリカを追加可能、プライマリは最大 300 GB まで拡張可能 ● レプリケーションによる可用性・耐久性の向上
Memorystore for Redis Cluster (MRC)	● シャーディングにより **書き込みスループットがスケールする** ● highmem-medium, highmem-xlarge は**ゼロダウンタイムでテラバイト単位のスケール** が可能 ● 2025 年 4 月に Memorystore for Valkey が GA. 新しいワークロードには Valkey 推奨 ● シャードにレプリカノードを追加することで **データの高い耐久性と高可用性を実現できる**
Bigtable	● Key-Value かつワイドカラム型ストア ● ノードを追加することで **読み取り・書き込みの両方がスケールする** （オートスケール可能） ● コンピューティングとストレージを分離するアーキテクチャにより **データの耐久性が高い** ● マルチクラスタでデータをレプリケーションすることにより **高可用性を実現可能**

データストア選定 v1

Bigtable

- 広告視聴セッション
- フリークエンシー
- ユーザーセグメント

キー空間が広く、GB ~ TB の大きいデータで、サーバー間でデータ更新をリアルタイムに共有したいデータを格納する。

ユーザー数や動画再生数に比例して増加するデータの格納先に適していると思われた。

MRC

- 予算消化

キー空間が狭く、数 GB 未満の小さいデータで、サーバー間でデータ更新をリアルタイムに共有したいデータを格納する。

サーバーでキャッシュして、アクセス頻度を抑えることでホットキー対策とする。

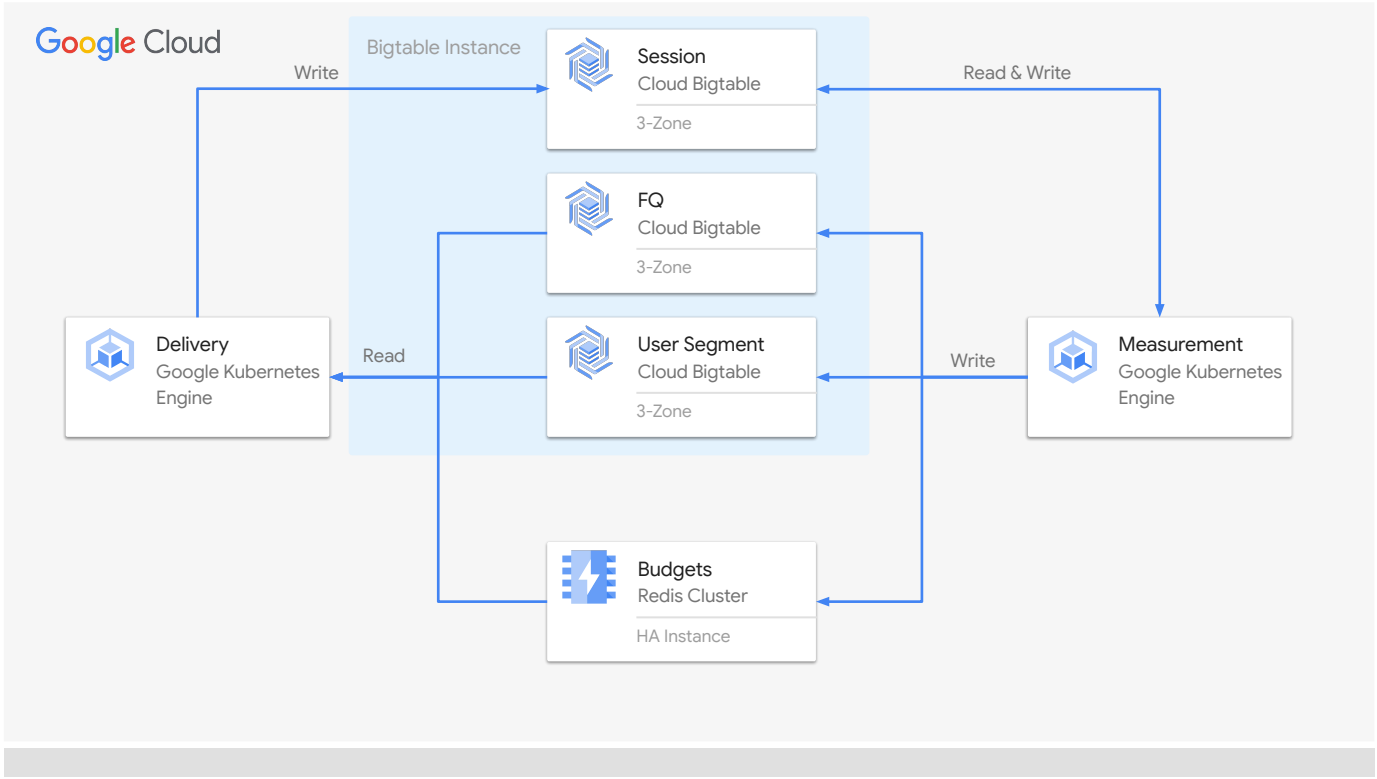
広告配信サーバーのオンメモリ

- 配信設定

キー空間が狭く、数 GB 未満の小さいデータで、更新頻度が低いデータはサーバーのメモリに格納する。

ネットワーク遅延がない分、オンメモリの方が高速。

Database Architecture v1

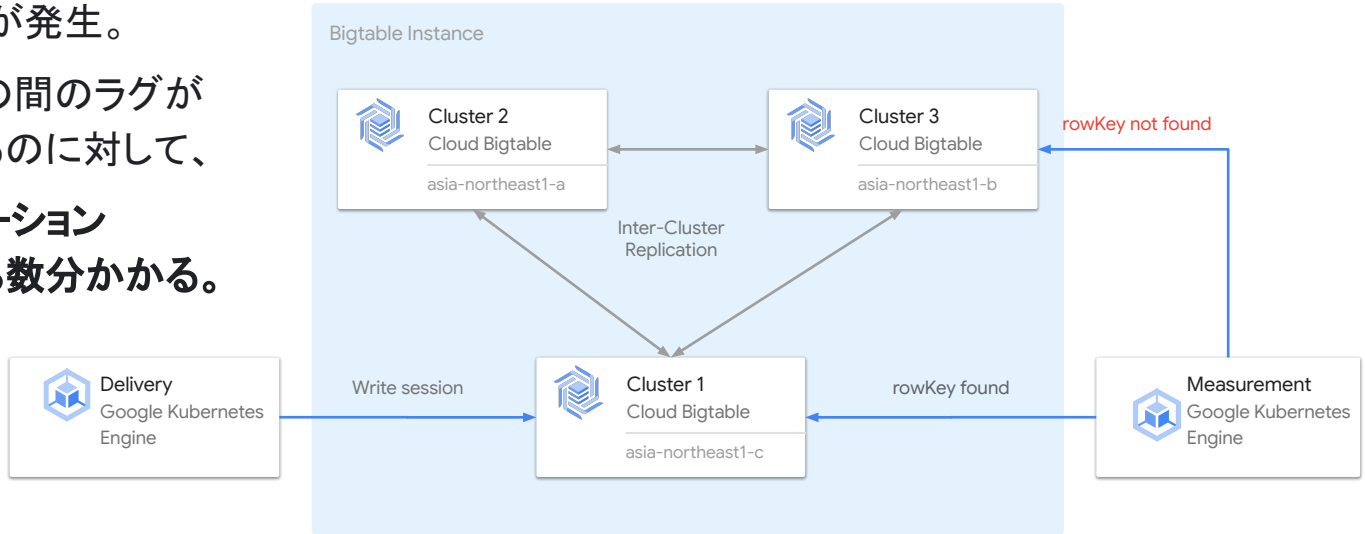


Issues in Benchmark

広告視聴セッション書き込み先と
読み取り先のクラスタが異なると rowKey
が見つからない課題が発生。

書き込みと読み取りの間のラグが
最短数十ミリ秒であるのに対して、

Bigtable の**レプリケーション**
レイテンシが数秒から数分かかる。



Bigtable の一貫性モデル

- 要件
  - Key-Value データベース に Write した最新の
    更新を数十ミリ秒後に読み取ることができる
- Bigtable の一貫性モデル
  - 単一クラスタルーティング → read-your-writes consistency
  - マルチクラスタルーティング → eventual consistency
  - **Bigtable で要件を満たすには**
    単一クラスタルーティングへの変更が必要
    - 万が一クラスタ障害が発生したら、
      手動でルーティング先のクラスタを変更
      しなければならない

MRC へマイグレーション

- Bigtable に拘ると、要件を満たすために一貫性と可用性のトレードオフを判断しなければならない
- 「書き込み後、数十ミリ秒後に読み取り可能である」要件を持つデータを MRC へマイグレーションすることで課題を解決した
- **MRC は数十ミリ秒という非常に短時間で収束する整合性とゾーン障害への耐性を備えている**

データストア選定 v2

Bigtable

- ユーザーセグメント
- ~~広告視聴セッション~~
- ~~フリークエンシー~~

キー空間が広く、GB ~ TB の大きいデータで、更新頻度が低く Read Heavy で **数秒から数分のレプリケーションレイテンシを許容する** データを格納する。

MRC

- 予算消化
- 広告視聴セッション
- フリークエンシー

数十ミリ秒で収束する整合性 を求めるデータを格納する。

予算消化は、数 GB 未満なのでサーバーでキャッシュして、アクセス頻度を抑えることでホットキー対策とする。

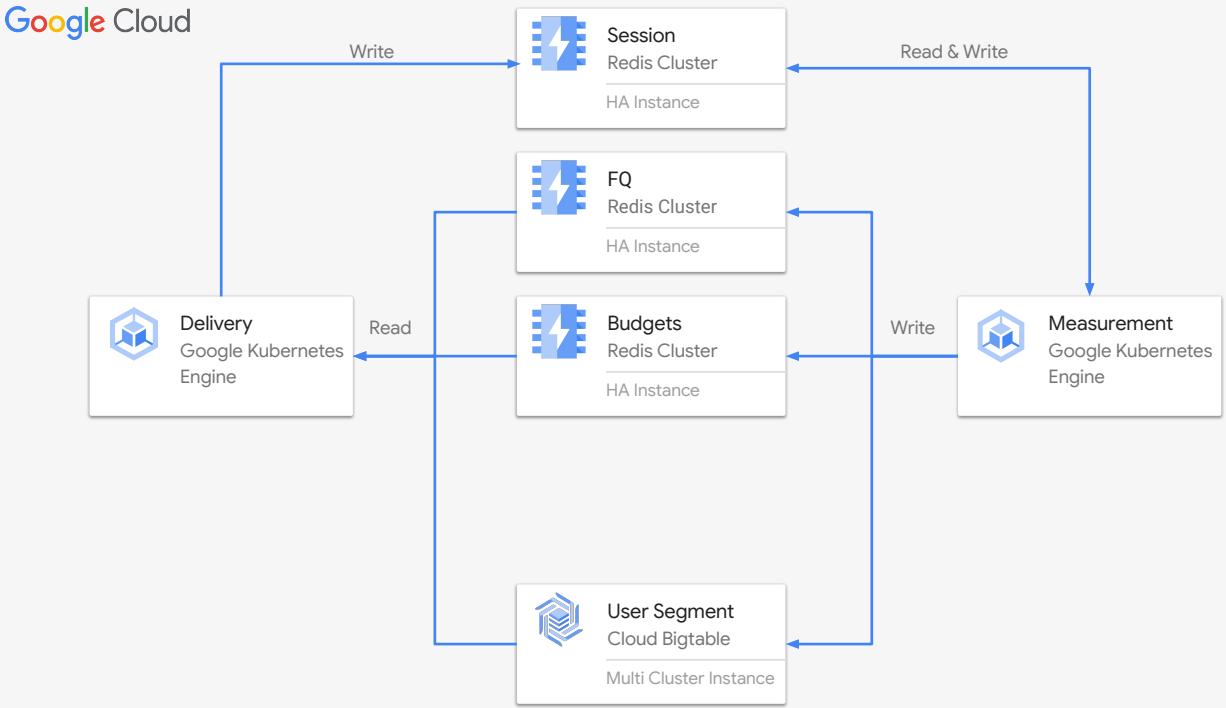
広告配信サーバーのオンメモリ

- 配信設定

キー空間が狭く、数 GB 未満の小さいデータで、更新頻度が低いデータはサーバーのメモリに格納する。

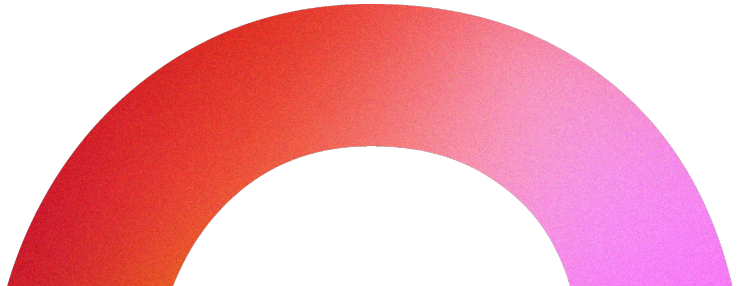
ネットワーク遅延がない分、オンメモリの方が高速。

Database Architecture v2



03. MRC & Bigtable 併用戦略

- 数十ミリ秒で収束する整合性を求めるデータは　MRC
- 数秒から数分のレプリケーションレイテンシを許容可能で、比較的安価で膨大なストレージが必要なデータは　Bigtable



Appendix: Bigtable row-affinity routing

- 2024 年 12 月に GA された row-affinity routing を利用すると
  リクエストの rowKey に基づいて**単一行の読み取りと書き込み** を
  特定のクラスタにルーティング可能。
  - ただし、**複数行には対応していない** 。
- row-affinity routing を利用すれば、読み取り後整合性の精度が向上し、
  要件を満たすことができるかもしれないが、
  MRC から再び Bigtable へ移行するモチベーションが高くない

src:

https://cloud.google.com/bigtable/docs/release-notes#December_11_2024

https://cloud.google.com/bigtable/docs/replication-overview?hl=ja#row-affinity-routing

https://cloud.google.com/bigtable/docs/routing?hl=ja#row-affinity

Appendix: Memorystore for Valkey

- Valkey は 2024 年に Redis 7.2 から Fork した OSS プロジェクト
- Memorystore for Valkey は Valkey 7.2 と 8.0 に対応したフルマネージド Key-Value データベースサービス
  - 2025 年 4 月に GA
- Valkey は、パフォーマンス, メモリ効率, 信頼性の面で強化されている
- 全ての新しいワークロードに推奨される
  - “What’s new with Memorystore for Valkey” at Google Cloud Next 25

src:

https://cloud.google.com/blog/ja/products/databases/announcing-memorystore-for-valkey

https://cloud.google.com/memorystore/docs/valkey/release-notes#April_02_2025

04. Bigtable 実践的チューニング

04. Bigtable 実践的チューニング

- Bigtable 負荷試験の勘所
  - Warm-up
  - gRPC Connection Pool Size

広告配信基盤の負荷試験

Professional Service Organization (PSO) チームと
徹底的に負荷試験を実施。

GKE, Bigtable の構成から、アプリケーションのデータ構造、設定のチューニングに至るまで、数ヶ月にわたって最適化。

PSO チームと実施した改善

GKE・Network

- GKE の設定最適化による性能改善
  - Pod, Node Sizing
  - podAntiAffinity
- スケール戦略
- Cloud NAT port 枯渇対策

Bigtable

- **読み取りレイテンシの改善**
  - Warm-up
  - gRPC Connection Pool Size の調整

Application

- 転置インデックスによる計算量の改善
- GOMAXPROCS の調整による性能改善
- net/http のパラメータ調整による TCP 接続の利用効率改善

Bigtable Warm-up

Bigtable の性能を引き出すために、負荷試験において
気をつけたこと

- ノード数を固定する
- Warm-up として、負荷試験と同じアクセスパターンで 10 分以上負荷をかける

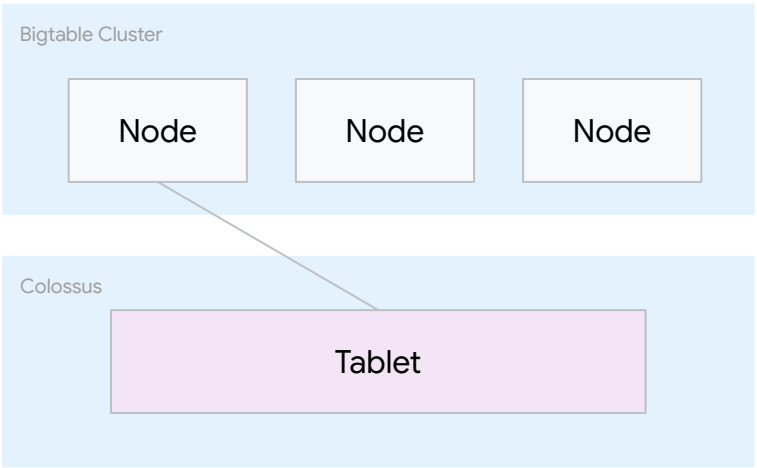
src: https://cloud.google.com/bigtable/docs/performance?hl=ja#testing

タブレットの分割と再配置

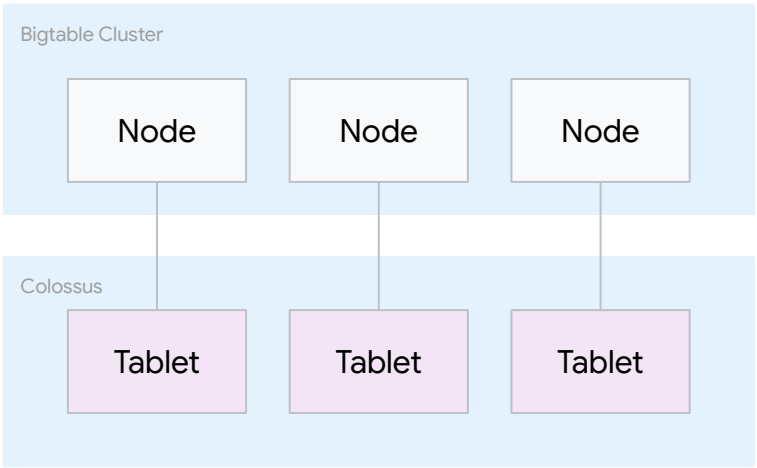
Bigtable はアクセスパターンに従って、タブレットの分割と再配置を行い、
10 分程度かけてデータを最適化する。

Bigtable の性能を引き出すために、負荷試験前の Warm-up で最適化させる。

負荷試験実施前にノード数を 1 → 3 台に変更した直後の状態

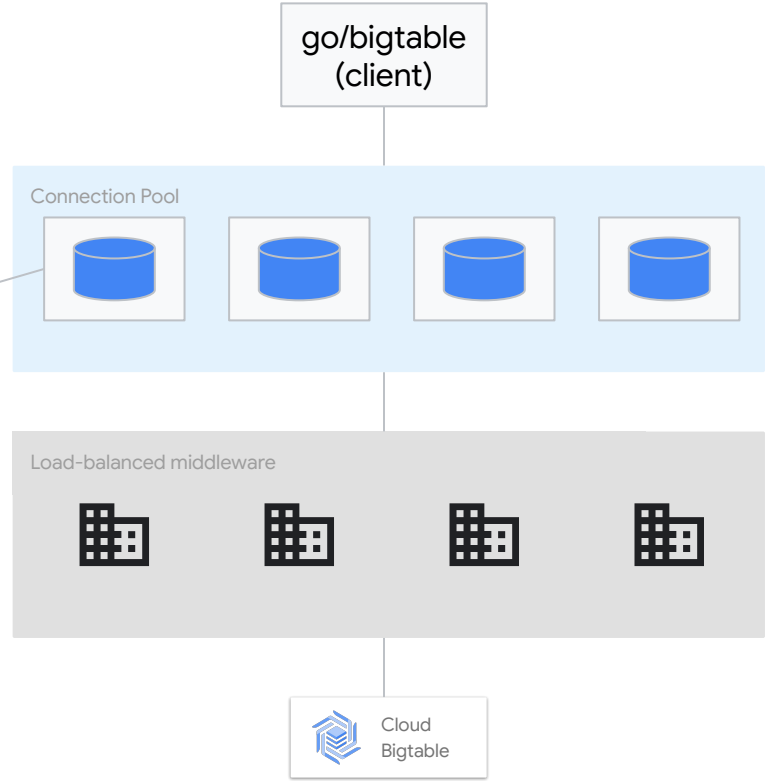


Warm-up によってデータが最適化された状態



コネクションプールサイズの調整

Go, C++, Java 用 Bigtable client は、
初期化時にコネクションプールサイズを
調整可能。

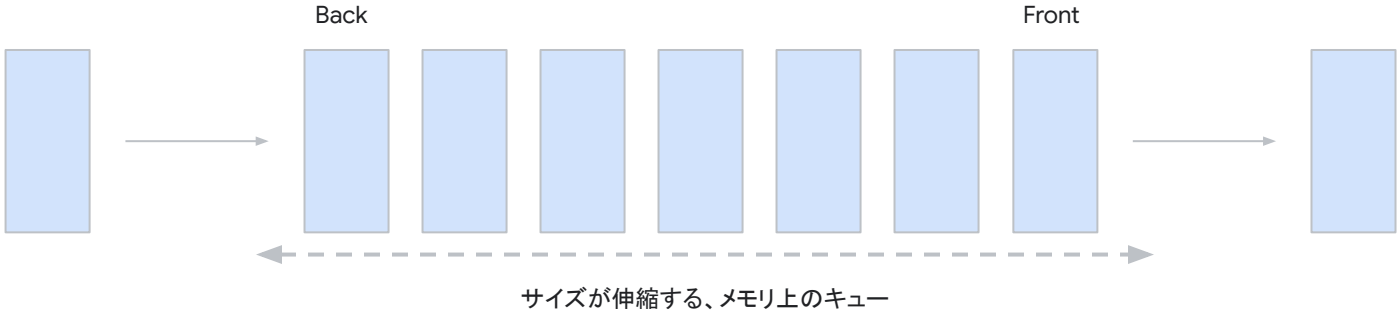


The number of connections in each pool is configurable in your code only when you use the Go, C++, or Java client libraries for Cloud Bigtable.

src:
https://cloud.google.com/bigtable/docs/connection-pools#when-to-change

コネクションプールサイズとレイテンシの関係

- コネクションプールサイズによって API リクエストの同時実行数が決まる
- 同時実行数に対してコネクションプールサイズが不足すると、
  API リクエストが**メモリ上のキューにバッファリングされる**
- **バッファリングによって API リクエストが待たされることになるので**
  レイテンシが増加する



API リクエストの同時実行数

- 以下の式で API リクエストの同時実行数が決まる
  - **Concurrency =**
    gRPC Connection Size * Streams per gRPC Connection
- gRPC Connection Size はアプリケーションコードで調整可能
  - デフォルト値は 4
- Streams per gRPC Connection は 100 で固定されており、調整不可
- つまり、API リクエストの同時実行数は 400 がデフォルト

コネクションプールサイズ不足に起因する問題

- API リクエストの**クライアントサイド指標のレイテンシ** が遅くなる
  - サーバーサイド指標のレイテンシは遅くならない
- アプリケーションの**メモリ使用量が増加** する
  - キューサイズが伸縮するため、メモリ不足で OOM Killed されることもある
  - コネクションプールサイズとメモリ使用量の関係性は **Cloud Profiler** で気づけた

クライアントサイドレイテンシの改善

- コネクションプールサイズの調整により、クライアントサイドで計測する ReadRows のレイテンシが 7ms まで改善した

Metrics	Before (Node * 1)	After (Node * 1)	After (Node *3)	Improvement
Cloud Bigtable Operations Latencies 95 percentile, ReadRows	49.39 s	112 ms	**7 ms**	**7055 times faster.**

メモリ使用量の減少

- コネクションプールサイズを増やしたことにより、広告配信サーバー (GKE Pod) のメモリ使用量が減少し、OOM Killed を回避

Metrics	Before	After	Improvement
Kubernetes Container Memory usage	37 GiB	**523 MiB**	**Reduced by approx. 98.6%.**

04. Bigtable 実践的チューニング

- Bigtable がデータを最適化する仕組みを理解して、負荷試験を実施する
- レイテンシを最適化するために、必要に応じて　Bigtable のコネクションプールサイズを調整する