Google Cloud

リアルタイム分析を支える Google Cloud のテクノロジー

西村 哲徳

Google Cloud, Data Analytics Specialist

アジェンダ

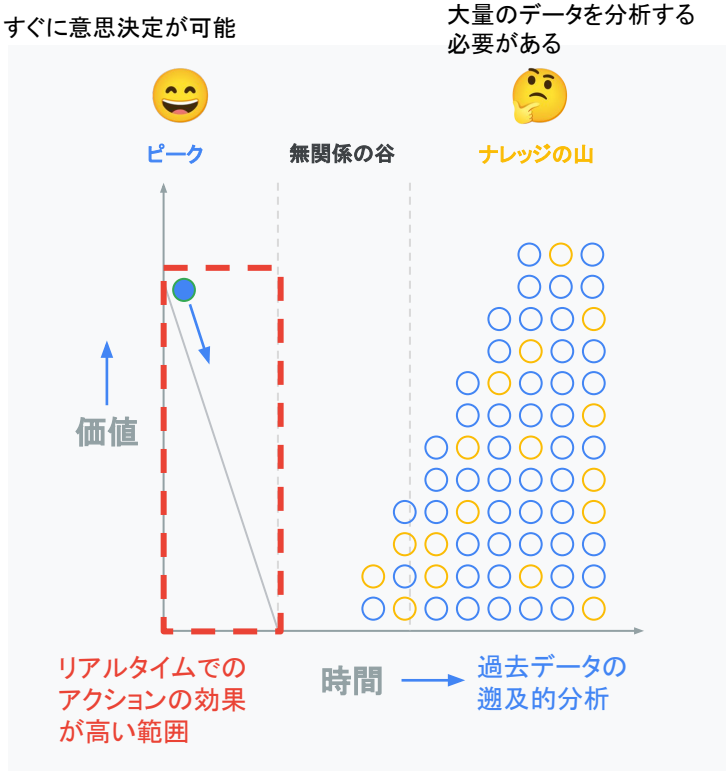
- リアルタイム分析とは
- Google Cloud で実現するリアルタイム分析アーキテクチャ

01

リアルタイム分析とは

リアルタイム分析の必要性

- 機会損失を防ぐ
  - 競争優位を獲得
  - 変化の早い状況への適応
  - より適切なデータポイントを収集
- 誤った判断を防ぐ
  - 鮮度の高いデータでの意思決定
- タイムラグによる選択肢の減少を防ぐ
  - 低コストの打ち手が取れる
  - 例）輸送コスト： 飛行機 vs トラック
  - 余裕をもった変更への適用
  - 意思決定の精度を高める



リアルタイム分析が可能にするユースケース

ヘルスケア

- リアルタイムの患者モニタリング
- 医療機器 IoT でのパーソナライズケア
- 保険金詐欺、廃棄物、薬の乱用の警告
- リアルタイムの医薬品供給と在庫の最適化

金融サービス

- 不正検出と防止
- 将来のコンプライアンスを計画
- リスクの予測と分析
- 顧客へのパーソナライズされたオファーを促進

小売

- リアルタイムのパーソナライズされたマーケティング
- ダイナミックプライシングとレコメンド
- リアルタイムの可視化で購入未完のアラート
- オムニチャネルでの顧客を即座に可視化

通信

- ネットワークの信頼性の改善
- 解約の最小化
- 加入者の不正を未然防止
- ネットワーク容量の分析と最適化

製造

- リアルタイムの在庫最適化
- 予知保全とアラート
- スマートサプライチェーン
- フリート管理と分析

リアルタイム分析に求められるシステム

増加するデータソースの取り込み

- データソース増加への対応
- 既存への影響を抑える
- フルマネージド

データ処理や分析の拡張性や柔軟性

- 季節性のあるトラフィック量への対応
- キャンペーン、イベントによるトラフィック増加への対応
- フルマネージド

無制限なデータに対する分析

- ウィンドウ集計と種類
- 遅延データの扱い
- 集計のトリガー
- イベント発生時間と処理時間の違い

ツール間の連携

- データ収集から分析までシームレスな連携
- Data Lake / DWH への効率的なデータ連携
- Data Lake / DWH に対する高度な分析

シンプルで運用負荷が少ないアーキテクチャ

02

Google Cloud で実現する リアルタイム分析アーキテクチャ

Google Cloud のリアルタイム分析ソリューションの特徴

堅牢で拡張性の高いデータ収集サービス

- データの冗長性を抑えながら複数の
  サブスクライバにデータをパブリッシュする信頼性
  の高いデータ/ イベントの取り込み
- サーバレスで自動スケーリング
- CDC サービス

統合されたストリームとバッチ処理

- ストリーム処理でデータとイベントから
  アクション可能な洞察をも得る
- バッチとストリームの統合により
  ラムダ アーキテクチャの負荷の軽減
- 様々なウィンドウ集計とウォーターマークによる遅
  延データのハンドリング

サーバレス アーキテクチャ

- リアルタイム ソリューションのキーである大量
  データに対応する拡張性
- スパイクの調整やプロビジョニングからの解放

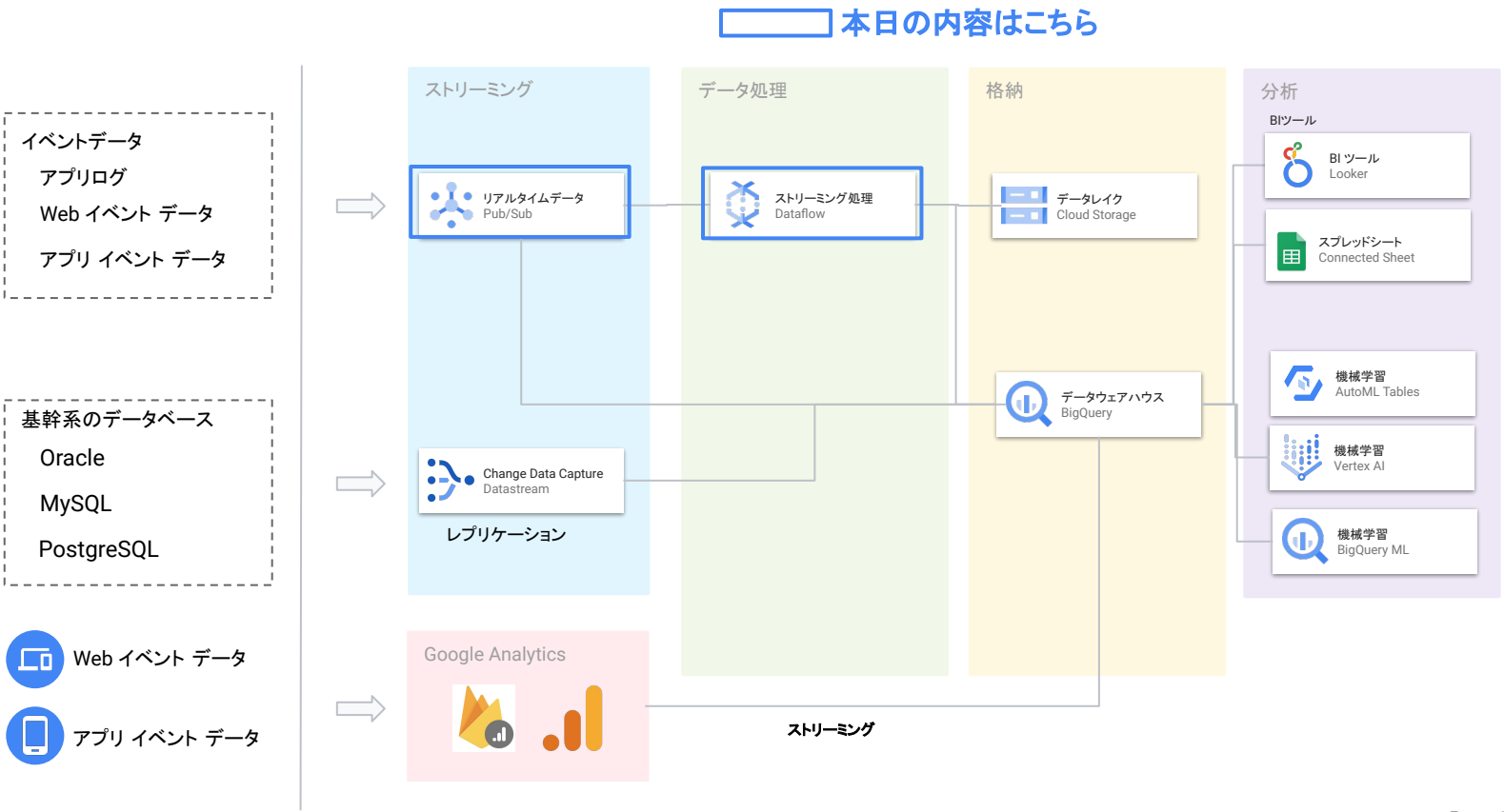
ストリーミングに対応したデータウェアハウス

- ストリーミング用のAPI
- ペタバイト規模のデータを格納、分析
- 機械学習等の高度な分析
- 分析ツールとの連携性

包括的な分析ツール

- DWH、機械学習、オンライン
  アプリケーションとの親和性

リアルタイム分析のアーキテクチャ



Architecture

Pub/Sub

アプリケーションとマシンがあらゆるものからインサイトを得るための ハイパー スケール、サブスクリプション メッセージング

Pub/Sub

リアルタイム分析のための メッセージングとイベントの取り込み

拡張性、耐久性のあるイベントの取込みと配信
サーバーレス、自動スケーリング、自動プロビジョニング。一貫したパフォーマンス。データレプリケーション。最大 31 日間、99.95% 以上の SLA

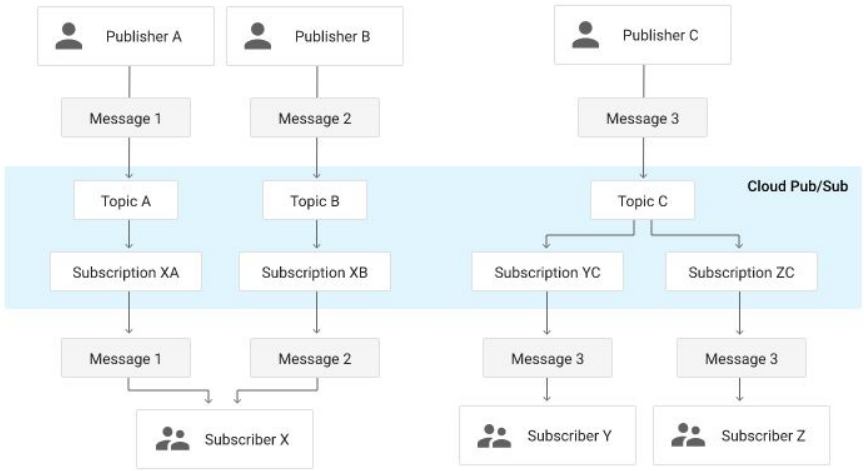
拡張性のある パブリッシュ / サブスクライブ パターン
トピックごとに最大10,000 のサブスクライバー; Push と Pull の配信モデル

グローバル ルーティング
地理に関係なく、イベントをパブリッシュおよびサブスクライブ

深いインテグレーション
Dataflow を使用したスケーラブルな分析、Functions を使用したサーバーレスアクション、組み込みの監視、監査ログ、コンプライアンス

Google サービスが 10 年以上にわたって基盤としてきた中核的な Google インフラストラクチャ コンポーネントをベースにしています。Google 広告や Google 検索、Gmail などの Google サービスは、このインフラストラクチャを使用して、**1 秒あたり 5 億件以上のメッセージ、総計 1 TB/秒以上のデータを送信**しています。

Pub/Sub の基本的なコンセプト



登場人物

- Publisher: メッセージを送信するアプリケーション
- Subscriber: メッセージを受信するアプリケーション
- Topic: Publisher のメッセージの送信先
- Subscription: Subscriber にメッセージを配信する リソース

Publisher と Subscriber の関係は、1 対多、多対 1、多対多など柔軟な構成が可能

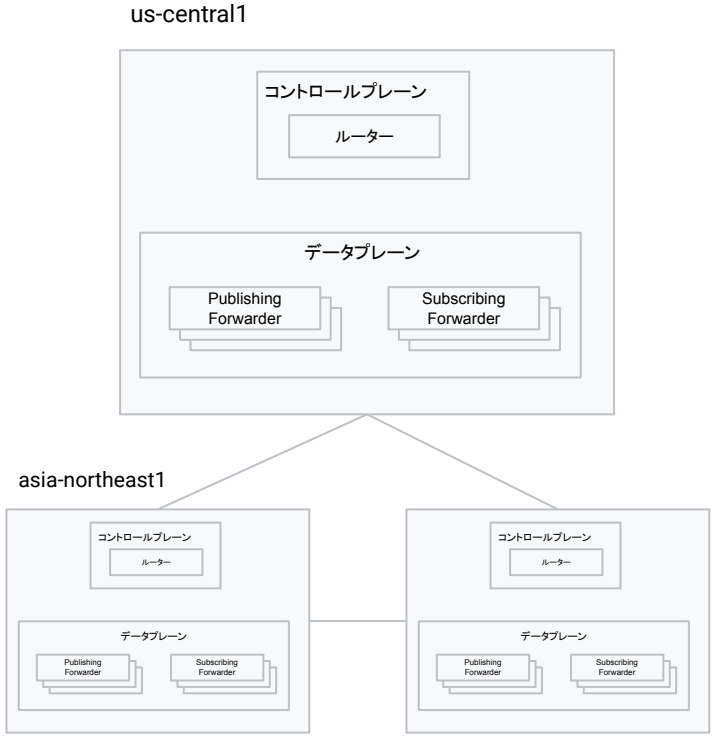
- 複数のアプリのデータを 1 つのアプリでまとめて処理
- 1 つのアプリのデータを用途に応じて複数のアプリで利用

Pub/Sub の様々な機能

- Push / Pull Subscription
  - Pull では **gRPC での Streaming Pull** も可能
- フィルタリング
- メッセージ保存期間
  - Topic: 最大 31 日間
  - Subscription: 最大 7 日間
- スナップショットとメッセージのシーク、再生
  - 特定の時間やスナップショットに戻りメッセージを再生
- デッドレタートピック
  - 配信不能メッセージをデッドレタートピックに送信
- メッセージの順序指定
- Exactly Once Subscription(preview)
- BigQuery Subscription(preview)
- モニタリング

Pub/Sub のアーキテクチャ

Pub/Sub がいかにして可用性を維持し、スケーラビリティや低レイテンシを実現しているのか



Pub/Sub はメッセージごとの並列処理（パーティション ベースのメッセージングではない）

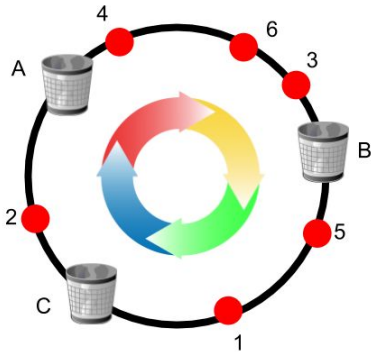
Pub/Sub は世界中のすべての Google Cloud リージョンで稼働

- 複数のリージョンの Publisher からの送信は、最も近い Google Cloud データセンターにパブリッシュされる
- メッセージはそれぞれ単一リージョンに保存

Pub/Sub を構成する主要なコンポーネント

- コントロールプレーン
  - データプレーン上のサーバに対する Publisher と Subscriber の割り当て
- データプレーン
  - Publisher と Subscriber の間で移動する　メッセージの処理

コントロール プレーン



ルーター : クライアントに対してフォワーダーを分配

- ネットワーク距離を最短になるクライアントの接続先データセンターを決定
- データセンター内で利用可能なフォワーダーセットに全体の負荷を分散
  - 負荷の均一性
  - 割り当ての安定性

Google Research 開発のコンシステント ハッシュ法

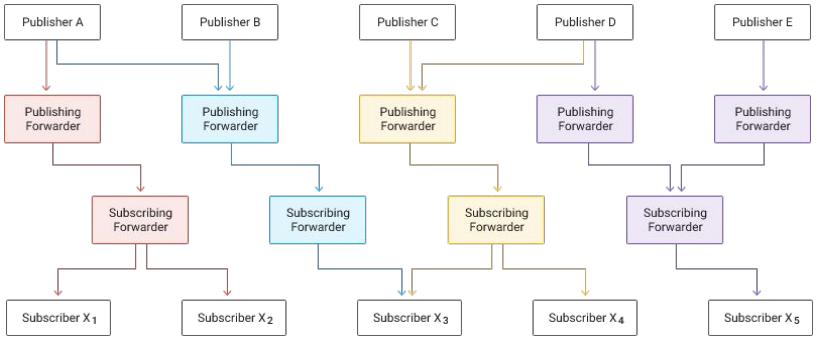
候補となるフォワーダーの順序付きリスト

クライアント:

- 順序に基づいてフォワーダーへ接続
- 障害が発生し接続が何度か失敗した場合、別のデータセンター内のフォワーダーへ接続試行

データ プレーン

クライアントからメッセージを受け取り、クライアントにメッセージを送信



Publisher は、複数の Publishing fowarder に同時に送信でき、Subscriber も複数の subscribing fowarder から受信可能

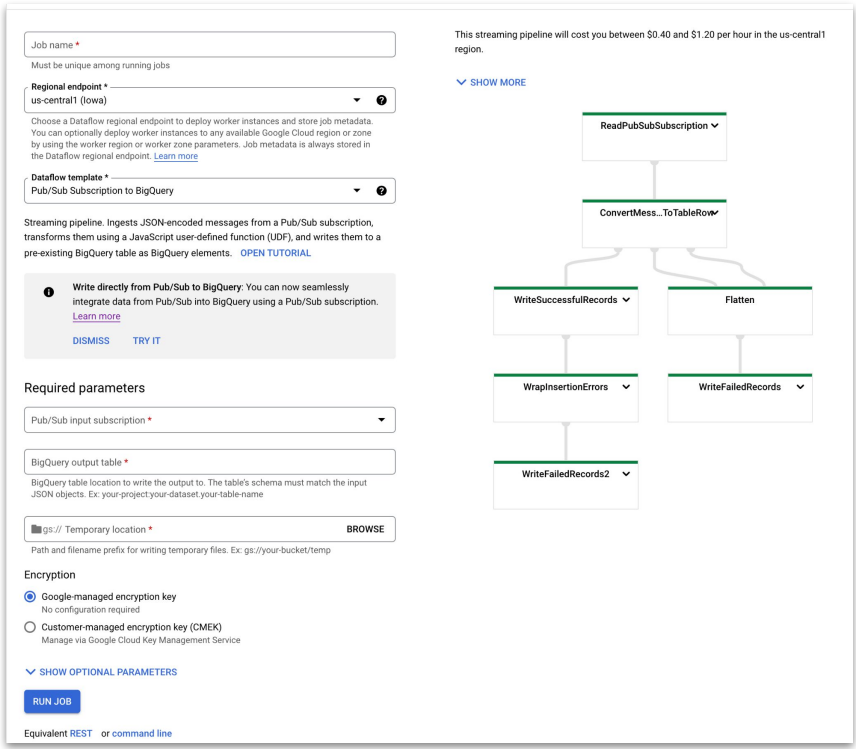
メッセージの流れ

1. パブリッシャーがメッセージを送信
   - プロキシレイヤで暗号化されPublishing forwarder に送信
   - **スループットに応じて動的に調整**
2. メッセージがストレージに書き込まれる
   - 「N÷2」個のクラスタに書き込まれると永続化(N は奇数)
   - クラスタ内の「M÷2」 個 のディスクに書き込まれると永続化(M は奇数)
3. 受信の確認応答をPublisher に返信
4. メッセージをストレージに書き込むと同時に、Subscriber にメッセージを配信
   - **スループットに応じて動的に調整**
5. Subscriber はメッセージを処理したことを示す確認応答を返信
   - Subscribing fowarder から確認応答をPublishing fowarder へ送信
6. 少なくとも1 つの Subscriber から確認応答を受信するとストレージからメッセージを削除

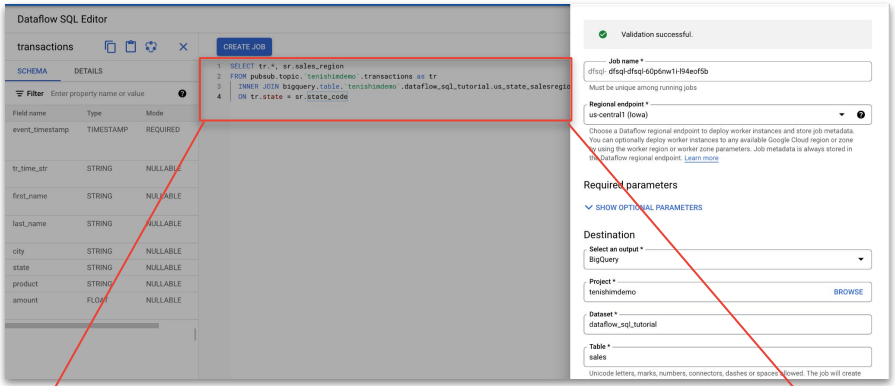
Google Cloud のサービスとの連携

Dataflow との連携

- Dataflow テンプレートを利用したデータ統合
  - Pub/Sub to BigQuery
  - Pub/Sub (avro / proto) to BigQuery
  - Pub/Sub to Pub/Sub
  - Pub/Sub to Splunk
  - Pub/Sub to GCS (avro / text)
  - Pub/Sub to MongoDB
  - Pub/Sub to ElasticSearch
  - Pub/Sub to JDBC

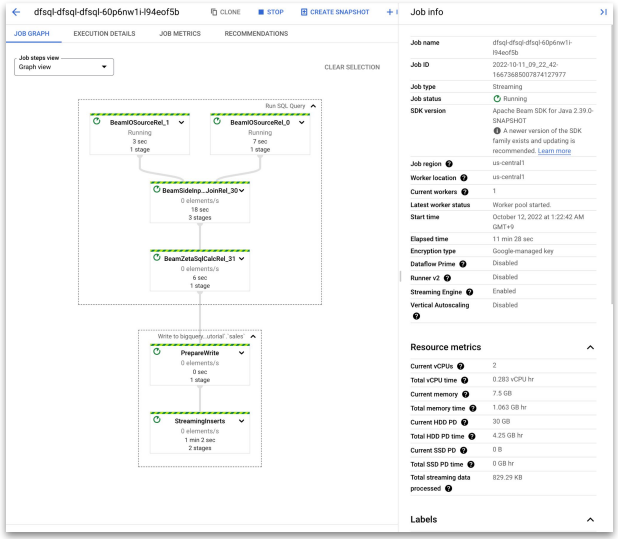


Dataflow SQL でストリーミングデータの結合




```
1  SELECT tr.*, sr.sales_region
2  FROM pubsub.topic.`tenishimdemo`.transactions as tr
3    INNER JOIN bigquery.table.`tenishimdemo`.dataflow_sql_tutorial.us_state_salesregio
4    ON tr.state = sr.state_code
```

Dataflow の Streaming
ジョブとして実行される



Pub/Sub のトピックと BigQuery 内のテーブルを結合して
BigQuery や Pub/Sub に ストリーミングデータを連携可能

BigQuery Subscription Preview

Dataflow でのデータ分析、変換、クレンジング等が不要な場合は直接　BigQuery へデータ連携

- 簡単にデプロイ
- サーバレス
- Streamin Pipeline のコストを抑える
- 低レイテンシでの連携
- BigQuery Storage Write API でストリーミング挿入 (後述)

Architecture

Dataflow

データ パイプラインの最適化と自動化、
ストリーミング インテリジェンスの複雑さを取り除く

Dataflow

ストリーミングおよびバッチデータ処理のための高度に自動化された自己修復データ パイプライン実行

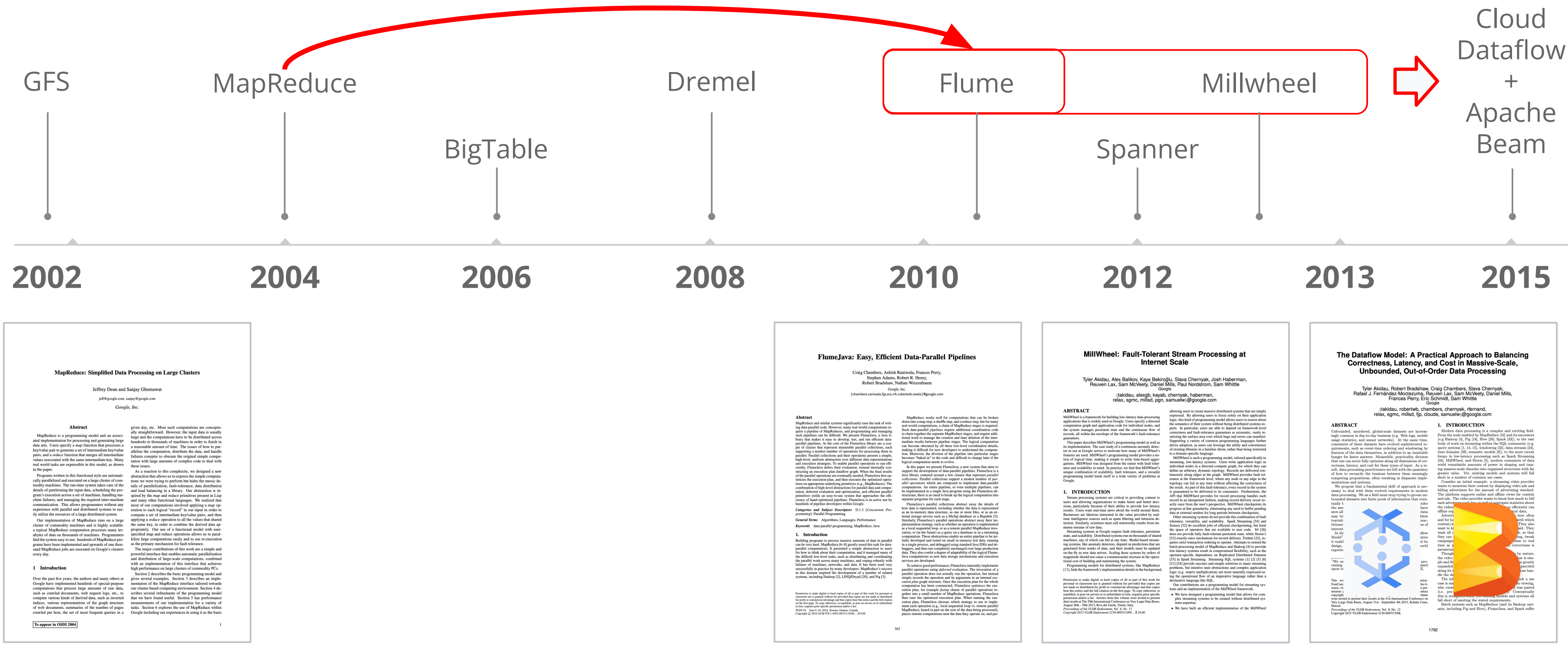
非常にシンプル
サーバーレス、自動プロビジョニング、および自己修復。自動化されたパフォーマンス、ワークバランシング、統合されたバッチとストリーミング

ML へのアクセス性
すぐに利用可能な MLOps、ML 推論、GPU と大規模データ処理

ベストな OSS と最適化されたプラットフォーム
ApacheBeamSDK によるオープン イノベーション。複数の言語 (Java, Python, Go) が利用可能。組込みの監視と観測でワークロードを最適化

規模に合わせて構築
水平スケーリングと垂直スケーリングの両方に対応するように設計および構築。あらゆる規模でパイプラインのパフォーマンスを最適化。使用率を最大化してコストを節約し、価値実現までの時間を短縮

Dataflow 誕生の歴史



Simple distributed data processing

Logical pipelines & optimization

Low-latency streaming

Batch + Streaming Serverless Cloud

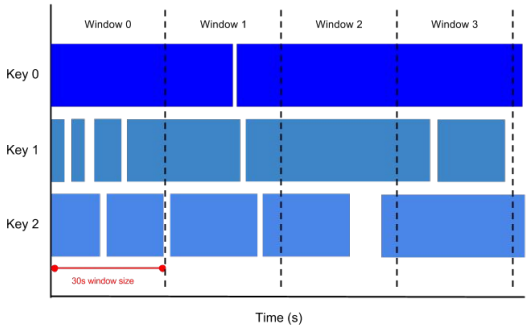
Apache Beam & Dataflow

バッチ処理の FlumeJava と Streaming の MilWheel を融合して単一システムに集約

- MillWheel は、□分にスケーラブルで、フォールトトレラントで、低レイテンシだがイベント時間セッションを簡単に計算できる□レベルのプログラミング モデルがない
- Dataflow の誕生
    - ウィンドウモデリング
    - トリガーモデル
    - 増分処理モデル

Dataflow の提供するウィンドウの概念

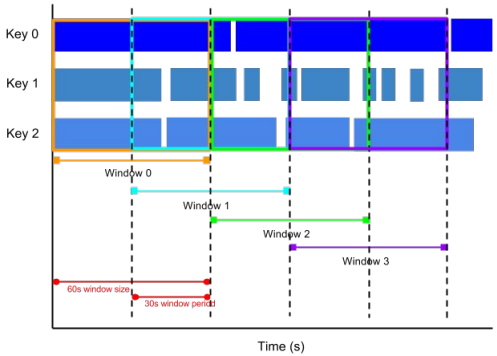
タンブリング ウィンドウ



固定ウィンドウのこと。データストリームを重なりなく分ける一定の時間間隔

```
fixed = (data | 'window' >>
beam.WindowInto(window.FixedW
indows(30)))
```

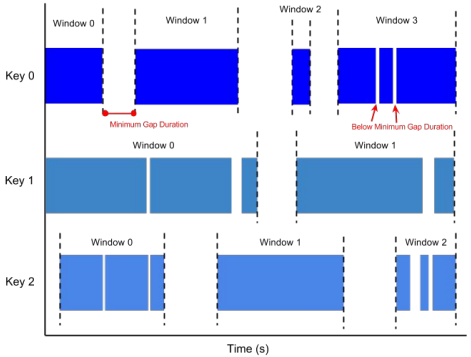
ホッピング ウィンドウ



スライディング ウィンドウのこと。一定の時間間隔のウィンドウが指定した期間で重なりあう

```
hopping = (data | 'window' >>
beam.WindowInto(window.Slidin
gWindows(60, 30)))
```

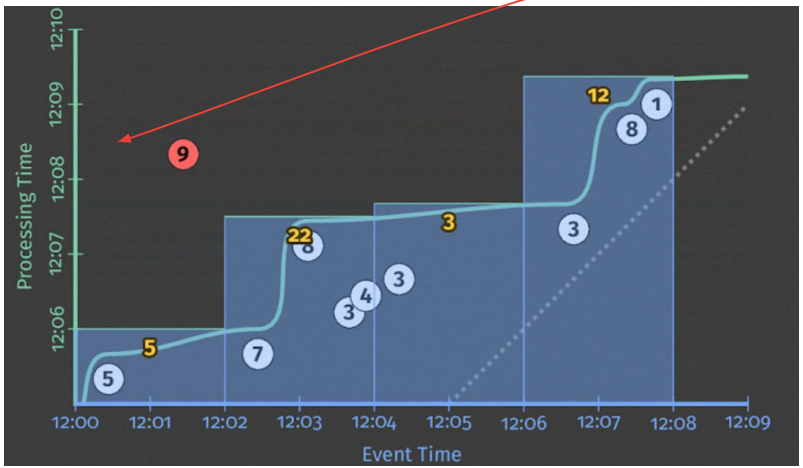
セッション ウィンドウ



あるキーの次の新しいデータが来るまでのギャップ期間を指定し、その間隔を超えると新しいセッションとしてウィンドを計算する

```
session = (data | 'window' >>
beam.WindowInto(window.Sessio
ns(600)))
```

ウォーター マークによるイベント時間と処理時間の違いを考慮した集計



ウォーター マークとは特定の**ウィンドウ内のすべてのデータがパイプラインに到着したと予想**されるシステム的な概念でこれを常に追跡

ウォーター マークを超えて到着した**遅延データの取り扱いも制御することが可能**

```
fixed = (data | 'window' >>
beam.WindowInto(window.FixedWindows(120),
allowed_lateness=1800
))
```

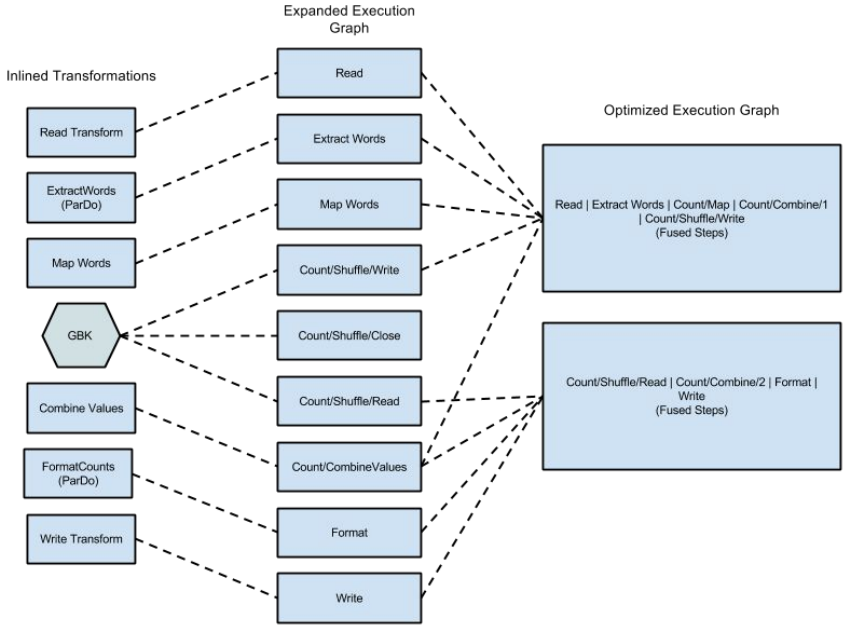
Dataflow 分散並列処理の最適化

1. 実行グラフの最適化
2. 水平/垂直 自動スケーリング
3. 動的作業再調整
4. Streaming Engine

完全に管理され自動構成なフルマネージドな
サービス

実行グラフの最適化

例) WordCount の実行グラフと最適化



融合の最適化

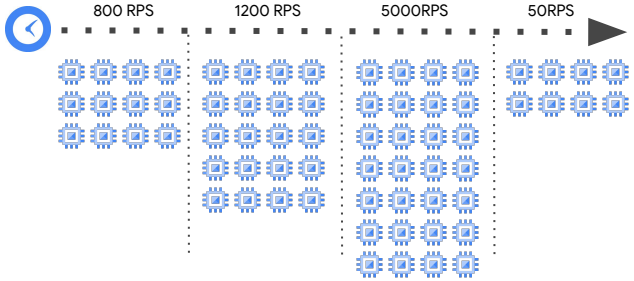
- 複数のステップまたは変換を単一のステップに融合することで、中間のデータを実体化する必要がなくなる
- 異なる順序や大きな変換の一部として実行し、効率的に実行する

結合の最適化

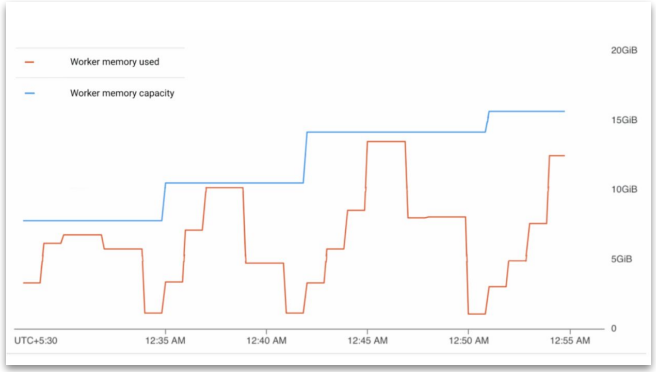
- 集約オペレーション中にインスタンスをまたがるデータを結合する前にデータをできるだけローカルで結合する。

水平 / 垂直自動スケーリング

水平自動スケーリング



垂直自動スケーリング



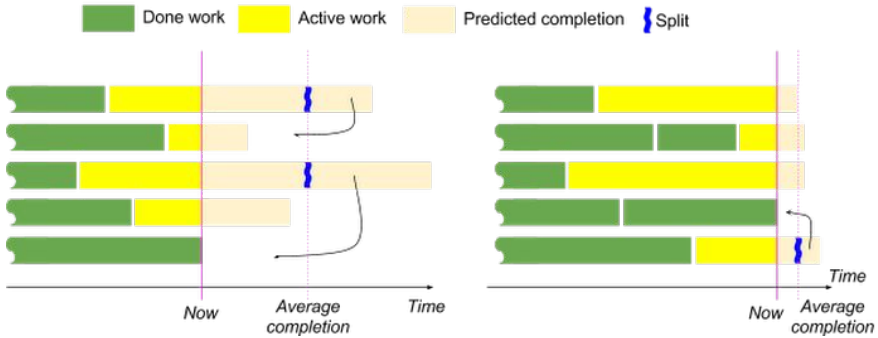
水平自動スケーリング

- 負荷に応じて動的にワーカー数の増減を行う
- バッチ、ストリーミングでは処理特性が違うのでワーカー数の計算方法は別
- ストリーミングはバックログの推定時間とCPU使用率で判断される（平均CPU 使用率が 20% 以上で数分間バックログされたままの場合、スケールアップ）

垂直自動スケーリング

- ワーカーが利用可能なメモリをジョブの要件にあわせて動的に増減する
- メモリ不足 (OOM) イベントとストリーミング パイプラインのメモリ使用量をベースにスケールされる

動的作業調整



ワーカーがアイドル状態になると、遅いワーカーの未処理作業が現在の平均完了時間までに完了するように分割される

並列分散処理では少数のワーカーが他より処理に時間がかかり全体の処理時間に影響を与える

- 作業割り当ての不均衡
- 異常に遅いワーカー
- データの一部が CPU インテンシブなデータ

work stealing / work shedding

- ワーカーの進捗を監視し遅いワーカーを特定する
- 遅いワーカーの未処理の作業を特定
- 未処理の作業をアイドルしているワーカーでスケジュールする

処理時間の短縮だけでなくコスト削減につながる

Streaming Engine

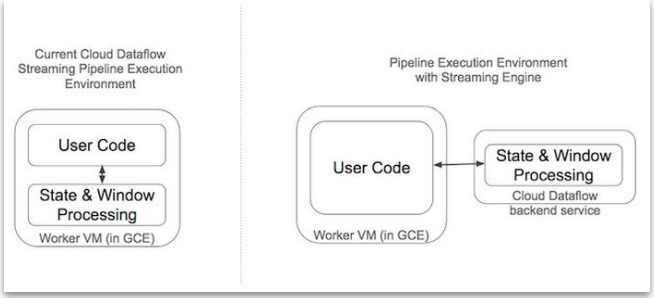


図 3: アクティブなワーカー (ストリーミング エンジンを使用しない場合)

図 4: アクティブなワーカー (ストリーミング エンジンを使用)

Shuffle 処理や Window の状態をワーカー VM 内ではなく Dataflow のバックエンドサービスで行う

- ワーカーはステートレスになる
- 自動スケーリングの応答性が上がり、よりスムーズで粒度の細かいスケーリングが可能に
- ワーカー VM リソースの削減 (CPU、メモリ、ディスク)
- サービス更新を適用するためにパイプラインを再デプロイする必要がないため、サポート性が向上

BigQuery のようにステート ストレージからコンピューティングを分離

まとめ

Google Cloud はリアルタイム分析に必要とされる要件を満たすテクノロジーを提供

堅牢で拡張性の高いデータ収集サービス

サーバレス アーキテクチャ

統合されたストリームとバッチ処理

ストリーミングに対応したデータウェアハウス