PostgreSQL 非同期レプリケーションの高性能な Serializability 実現法 Read-safe snapshot (RSS) の紹介

- 塩井 隆円, 東京科学大学 (Science Tokyo) (旧 東京工業大学)
- 引田 諭之, Scalar, Inc.

2024/12/06
PostgreSQL Conference Japan 2024

はじめに

- NEDOプロジェクト の支援で開発された Tsurugi RDBMS の
  HTAP 部分の理論的枠組み *Read safe snapshot (RSS) の部分の紹介

  - 東工大チームとしてプロジェクトに参加、Tsurugiの (神林さん、荒川さん、黒澤さんと共に)
    トランザクション理論の新しい枠組みとしてRSSを提案
  - RSSを最適化してPostgreSQL用にも定式化、プロトタイプとして実装・評価し論文化*
    - 今回は、そのポスグレ部分に焦点を当てて話します。

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

はじめに

* 論文は無料でダウンロードできます。
https://doi.org/10.1016/j.is.2024.102385

View PDF　Download full issue

ELSEVIER

Information Systems
Volume 124, September 2024, 102385

Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads

Takamitsu Shioi [a], Takashi Kambayashi [b], Suguru Arakawa [b], Ryoji Kurosawa [b], Satoshi Hikida [c], Haruo Yokota [d]

Show more

Add to Mendeley　Share　Cite

https://doi.org/10.1016/j.is.2024.102385　Get rights and content

Abstract

This paper proposes Read-Safe Snapshots (RSS), a concurrency control method that ensures reading the latest serializable version on multiversion concurrency control (MVCC) for read-only transactions without creating any serializability anomaly, thereby enhancing the transaction processing throughput under mixed workloads of online transactional processing (OLTP) and online analytical processing (OLAP). Ensuring

- HTAP：一つのデータベースシステムでOLTP (高速な更新) とOLAP (大規模な集計)を処理すること
  - HTAPシステムの評価軸として３つの側面がある。
    - OLTP/OLAP性能
    - データ整合性保証
    - データフレッシュネス
      - データフレッシュネス：OLTP側で更新したデータをOLAP側で読み出すまでの遅延時間
  - HTAPシステムといえば、性能とデータフレッシュネスに着目した話ばかり
    - 我々のプロジェクトは、この中でも先ずデータ整合性保証に着目。

はじめに

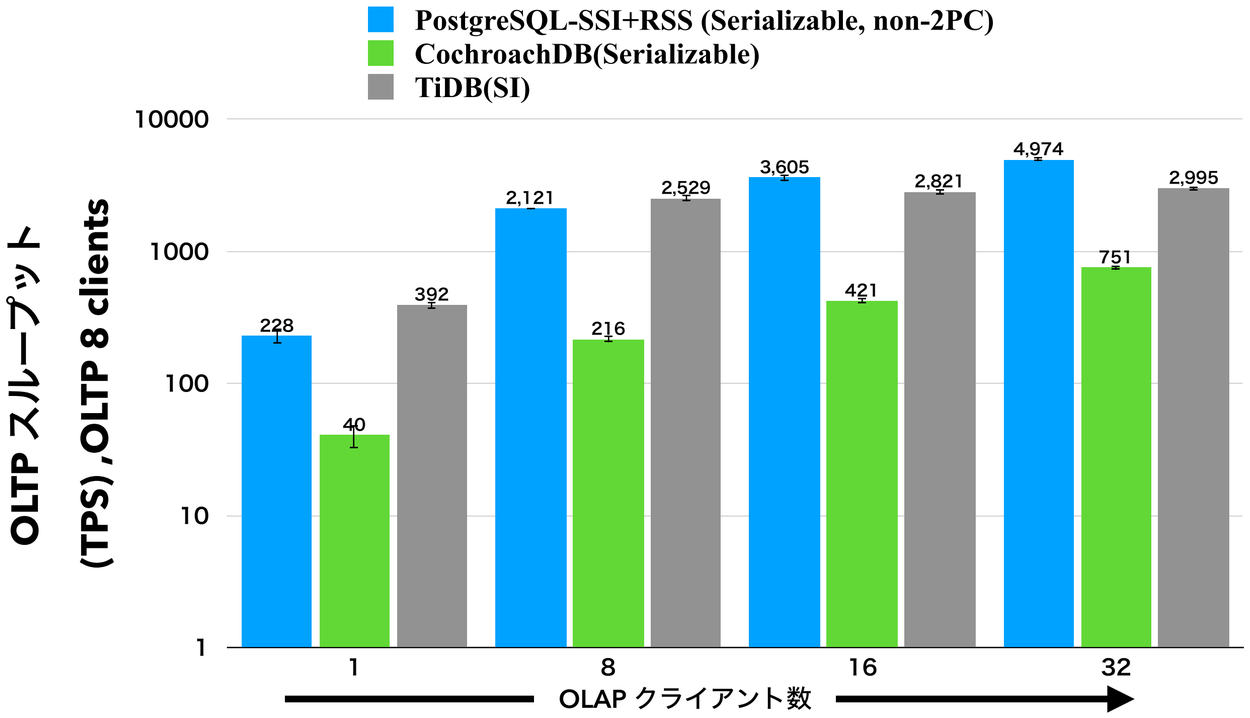
- データ整合性の保証
  - データベースではSerializabilityを保証することで、データ整合性を保証している。
  - Serializability：データベースから得られる結果に異常(Anomaly)が起こらないことを保証できる。
  - デフォルトでデータベースをインストールすると Serializable ではないことがほとんど。
    - 性能が出ない (そもそもSerializableを知らない、知ってても使わない。)
      - トランザクションがAbort/Waitされる（ただのエラーと思っている人が多い）
      - Abort/Waitによってデータの異常を事前に防いでくれる。
        - 高負荷なのにAbortしないならデータが壊れる。→アプリ開発者がなんとなく対応。（アプリ開発で忙しくてデータベースに構ってられん、性能が高ければ良い）
  - Serializable かつ 高性能 なデータベースシステムを作るのはとんでもなく難しい。

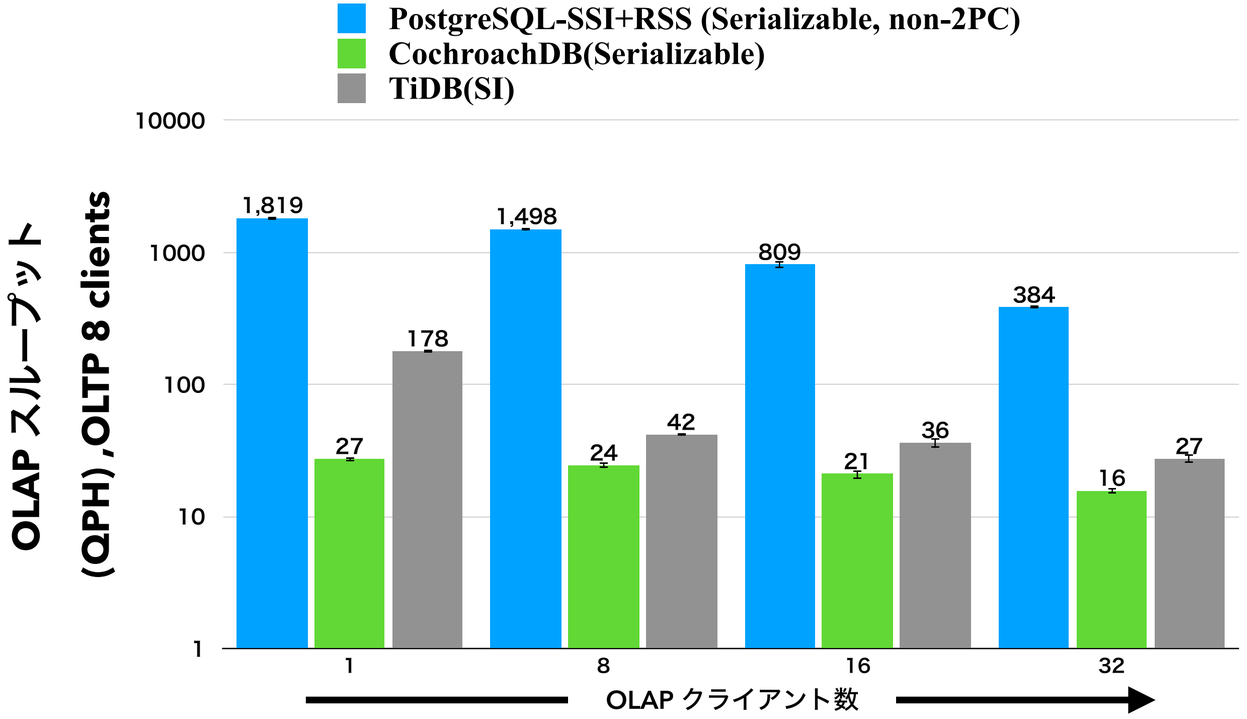
分散HTAPシステムとPostgreSQL-Serializable(RSS)の性能比較

- 今回紹介する手法RSSをPostgreSQLストリーミングレプリケーション環境に実装し、

  HTAP性能測定用ワークロードの実行結果 (*詳細は論文をご参照ください。差が大きいためログスケール)

- 分散HTAPデータベースシステムの CockroachDB (Serializable)、 TiDB (Non-Serializable) と同条件で比較





- (青色) RSSによって Serializable を実現しつつ高い性能を実現 (300秒間実行し、即終了)
- (緑色) Serializable かつ高性能にするのは大変：300秒間実行したトランザクションが全て返ってくるのに5時間掛かる等（これがSerializableHTAPを普通に実現するということ, Abortの代わりのWait手法)

自己紹介

- 塩井 隆円,
  - 特に何も持ってない人。（無駄に歳を取る）
    - 博士号（工学）を持った。（役立つ機会なし）
    - 職も持てた。(転職活動で自己紹介に疲れた)

- 引田 諭之, Scalar, Inc.
  - 博士（工学）東京科学大学（旧：東京工業大学）
    - すごく気合の入った良い人。
      米を沢山くれた。果樹園も営む。
    - 東工大チームで一緒にやった人。
      PostgreSQL実装で沢山助けてもらいました。

▸ ご飯を食べ損ねた時の写真

アウトライン

1. データの整合性が崩れた時に起こる異常 Anomaly

- データの並行アクセスによって起こる異常 (Anomaly)
  - Read only transaction anomaly の説明
- Read only Anomaly検出

2. HTAPだとより整合性保証が難しい

- HTAP
- PostgreSQL-SafeSnapshot
- Read Safe Snapshot (RSS)

3. RSSを利用しPostgreSQLレプリケーション環境を高性能かつSerializableなHTAPシステム化

- PostgreSQL-SSI-RSS 実装
- 性能測定 HTAPワークロード
  - シングルノード環境：Postgresql-SSI 単体のHTAP性能も見れます、SSI+SafeSnapshot, SSI+RSS
    マルチノード環境：RSSのオーバーヘッド

データの並行アクセスによって起こる異常 (Anomaly)

- 複数のデータを複数ユーザで読み書きした結果から起こるデータの異常。
  ダーティリードとかファントムリードとかあの辺。
  - データベース分野のanomalyの例
    - Lost Update :
      - 自分が書いたデータが他の人に上書きされその後読み出せない状態。
    - Non-repeatable Read (Fuzzy Read) :
      - 一度読んだデータを他の人が上書きし再度読み直しても読めなくなる現象。
    - Write Skew anomaly :
      - 二人が同じデータを読み、その二人が別々のデータを制約を無視して書き込めてしまう現象。
    - Read only transaction anomaly
      - 今回紹介して、Read only anomaly発生率を測定した結果を見せます。

A read only transaction anomaly [1]

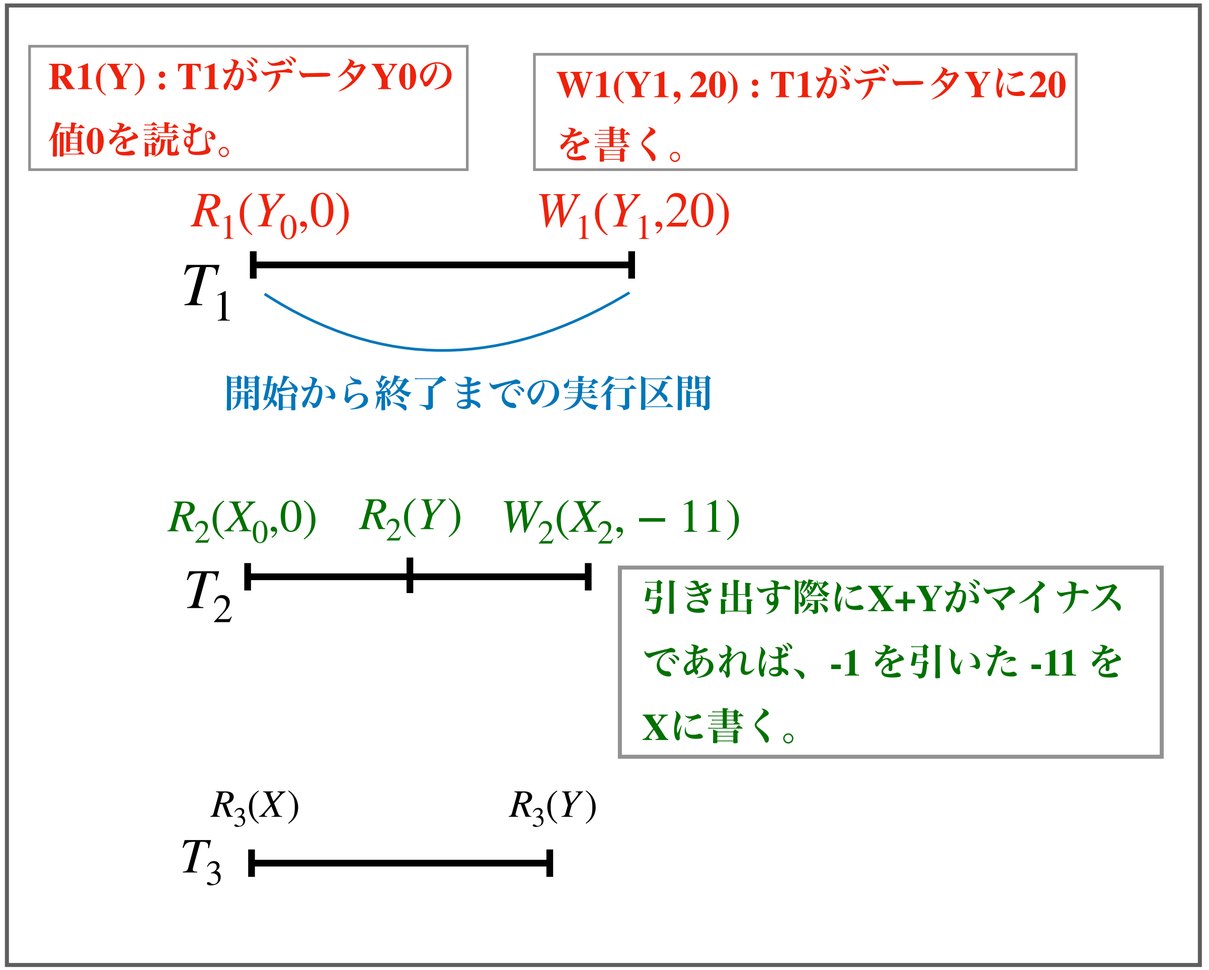
- **Read only anomaly の例**
  - **X を当座預金口座, Y を普通預金口座の残高として表す。**
  - **二つの口座 X,Y の初期値は 0。**
  - **もし X + Yがマイナスになれば、当座貸越を受け入れ、貸越手数料 -1 が掛かる。**

トランザクション T1：Y に20を入金。

トランザクション T2：X から10を引き出す。

トランザクション T3：X, Y の値を印刷する。（顧客に説明するために）

❖トランザクションの実行イメージ図

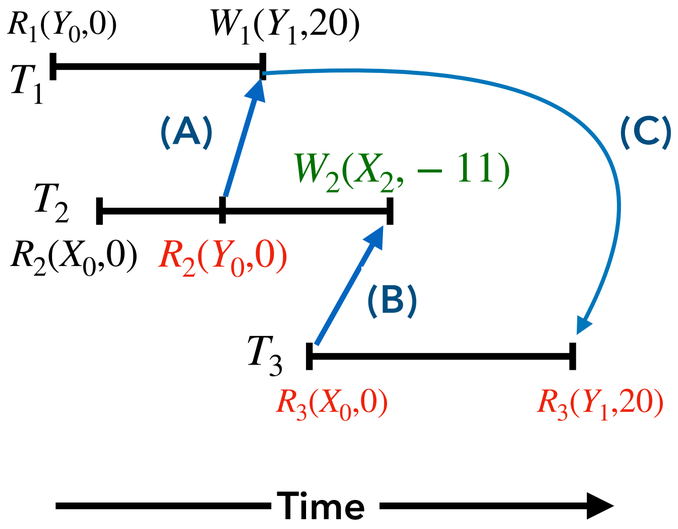


[1] Alan Fekete, Elizabeth O'Neil, and Patrick O'Neil. 2004. A Read-Only Transaction Anomaly under Snapshot Isolation. *SIGMOD Rec.* 33, 3 (Sept. 2004), 12–14. https://doi.org/10.1145/1031570.1031573

A read only transaction anomaly [1]

- トランザクションを図の順序で実行する

❖ Read only transaction anomaly



- トランザクションの並行実行と競合(conflict)

  (A) T1がYに書き込む前に、T2が開始しYを読む。

  (B) T2がXに書き込む前に、T3が開始しXを読む。

  (C) T3が最新のYの値を読む。

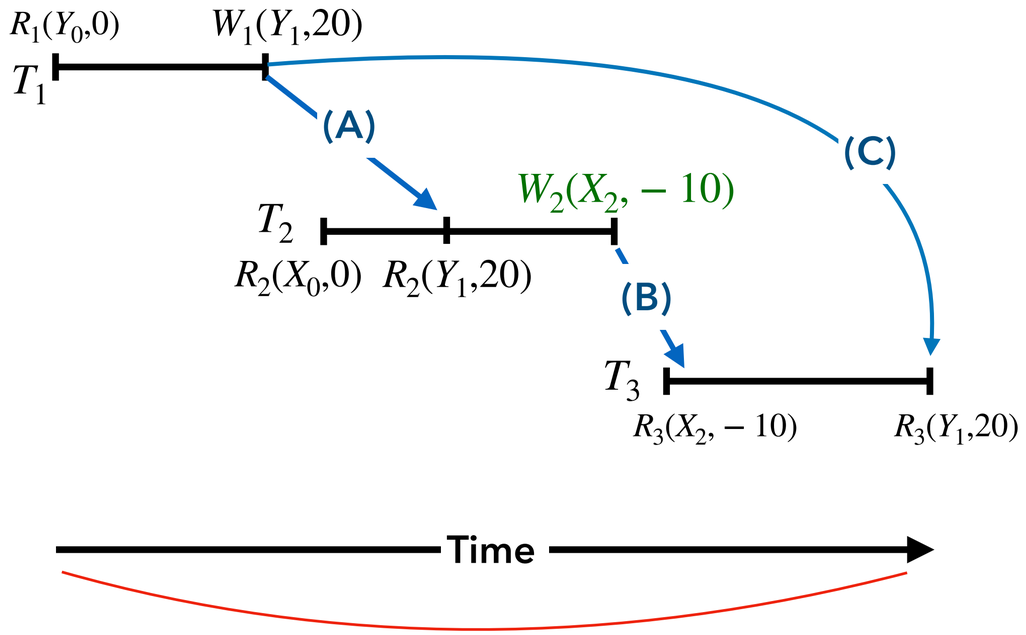
- T2 はX+Yの値を見て、-10を書く前に手数料をつけて -11 を書き込む。
- T3時点の出力によって手数料は無く安全です、と顧客に説明する。
  - 実際には手数料が引かれた結果が書き込まれている。

[1] Alan Fekete, Elizabeth O'Neil, and Patrick O'Neil. 2004. A Read-Only Transaction Anomaly under Snapshot Isolation. *SIGMOD Rec.* 33, 3 (Sept. 2004), 12–14. https://doi.org/10.1145/1031570.1031573

Anomaly回避とパフォーマンス低下

- 例えば一個ずつトランザクションを逐次実行して防ぐ方法はあるが、性能が劣化してしまう。
  - ロック(2PL) によって各トランザクションが競合したらブロックするのも同じ結果になる。

❖ 逐次実行



▸ Anomalyを防ぐために性能が犠牲になる。

(A) T2がT1の終了を待ち、T1の書いたYを読む。

(B) T3がT2の終了を待ち、T2の書いたXを読む。

(C) T3が最新のYの値を読む。

- T2 はX+Yの値を見て、手数料なしと判断し、-10を書き込む。
- T3の出力によって手数料は無く安全と顧客に説明する。

おさらい：データベースの整合性保証

- データベースで理論上Anomalyを起こさないことを保証するには：Serializabilityを保証する
- Serializabilityを保証することでトランザクションを並行実行してもデータ整合性が保たれる。

- Serializable isolation
  - データベースから得られる結果に異常（anomaly）が無いことを保証できる。
  - 実現方法やワークロード次第で性能が大きく低下する：HTAP。
    OLTPだけなら競合が少なく、性能低下も少ないかも。
- Non-Serializable
  - Read committed, Snapshot Isolation(SI), etc.
  - データベースのデフォルト設定で多い。
    性能が落ちるためSerializableにしない。
    - Anomalyは謎のエラーとして
      アプリケーション開発者が頑張って防ぐ。

❖ Serializable とパフォーマンスの関係

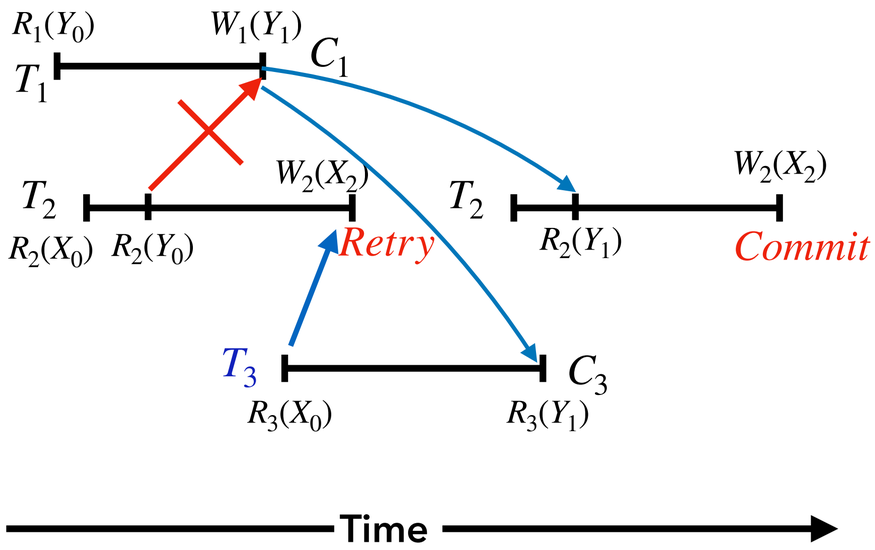
Isolation Level	整合性	性能
Serializable	○	×?
Non-Serializable	×?	○?

- “?” はワークロード次第で変わる。Serializabilityを保証する仕組み次第で性能は変わる。
  Abort/Waitをいかに減らせるか

PostgreSQLのSerializability 実現方法

- SSI (Serializable Snapshot Isolation [2]) 方式
  - SI (Snapshot Isolation) の環境下で Anomaly が起こる場合、必ず二つの連続する上書きの競合 (Dangerous structure) があるため、これができたらAbort/Retryする。

❖トランザクションの実行イメージ、例



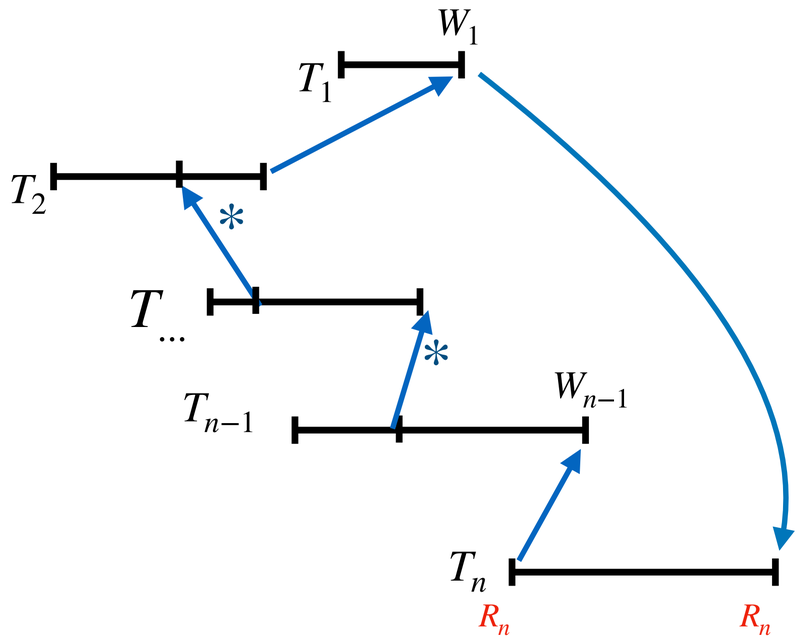
- 競合が一つあっても Abortしないため、 普通のロック(2PL)よりもSerializable かつ 性能が高くなる。
  - 2PLのような方法よりも理論上のパフォーマンスが良い。
  - 2PL だと二つのトランザクションが競合しただけでやり直し。
  - SSI だと 二つのトランザクションが競合してもやり直しにならない。

[2] M. J. Cahill. Serializable Isolation for Snapshot Databases. PhD thesis, University of Sydney, 2009.

Read only anomaly の定義

- Serializableなトランザクション群に Read-only transaction が参加して起こる Anomaly として Read-only anomaly を定義*
  - トランザクションとトランザクションのデータ競合による依存関係を有向グラフとして表した時にサイクルが起こると競合ベースのSerializableではないことが分かる。それがRead only transaction によって起こされると Read only anomaly
- SSI では Read only anomaly を防げる：二つ競合が連続したらabortするため。

❖ Read only anomaly のイメージ図

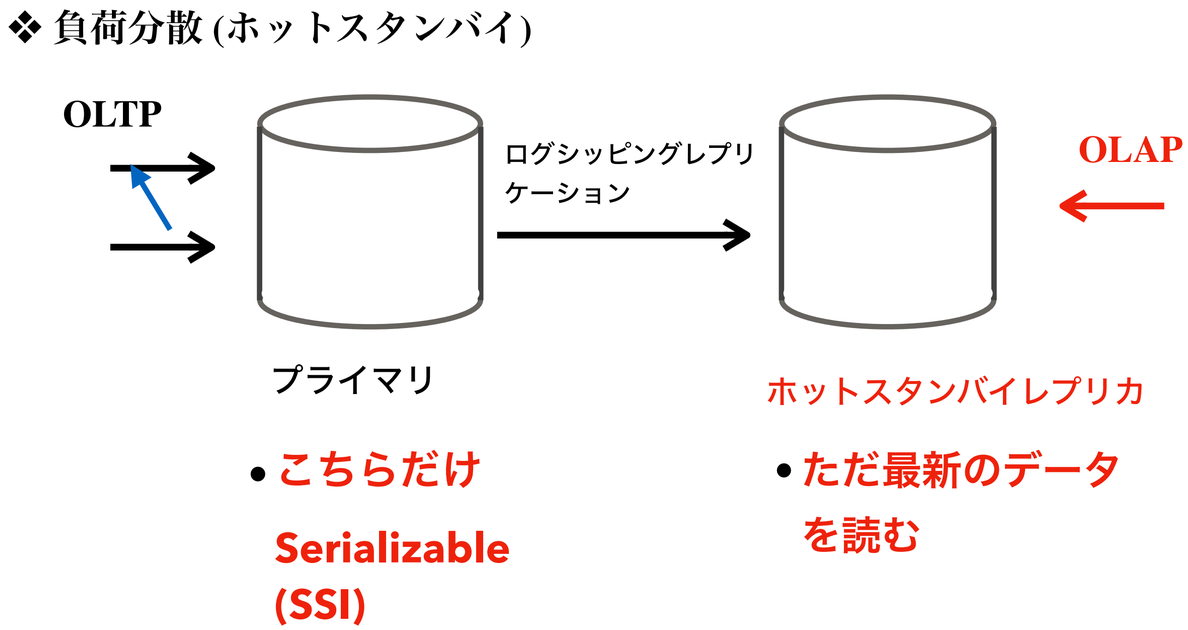


- Read only transaction は最新のデータを読んだだけ

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

よくあるプライマリレプリカ構成の場合

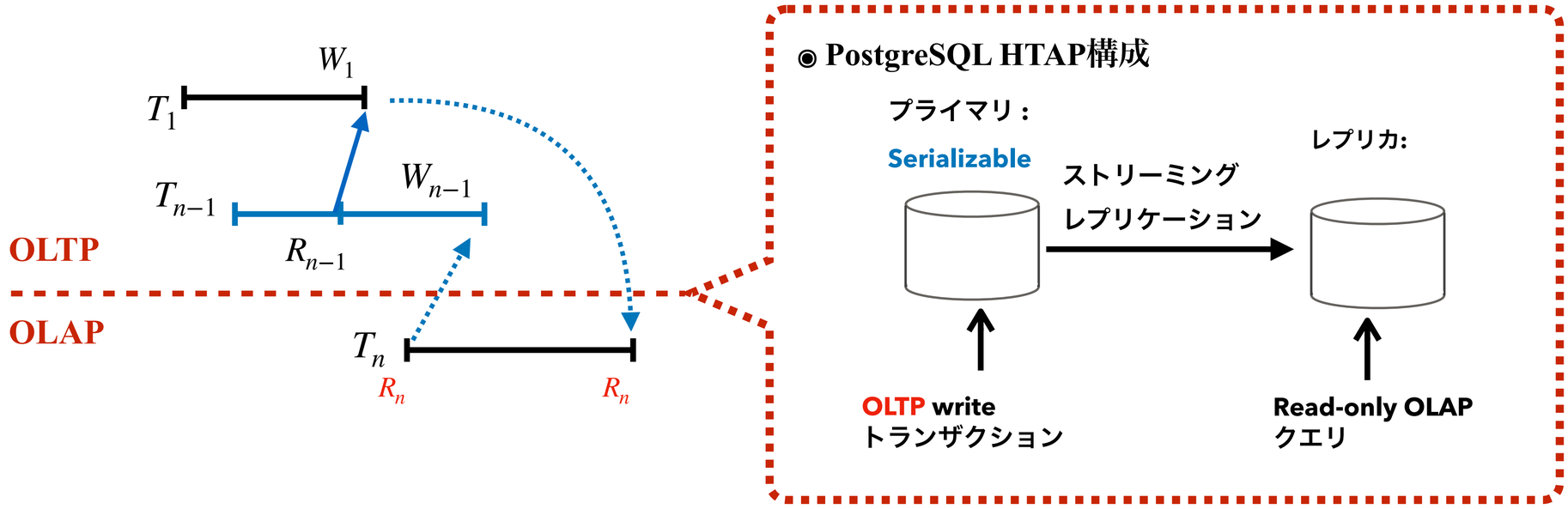
- よくあるリードレプリカ構成で考えると、
  - 書き込みのプライマリ側だけSSI、リードレプリカはそのまま
- プライマリ・リードレプリカ分離環境で Read only anomalies がどれくらい起こるか実際に調べる。データベースのリードレプリカを作って、Read only クエリを投げるような負荷分散環境



よくあるプライマリレプリカ構成の場合

- リードレプリカ環境でHTAP用ワークロードを実行し Read-only anomaly 発生率を調べる。
  - HTAP用ワークロード：CHBenchmark
    - データベースの更新トランザクション群のOLTPと、分析系クエリ群のOLAPを同時に実行しパフォーマンスを計測する。

❖ PostgreSQLレプリケーション下での Read only anomaly の発生数を計測

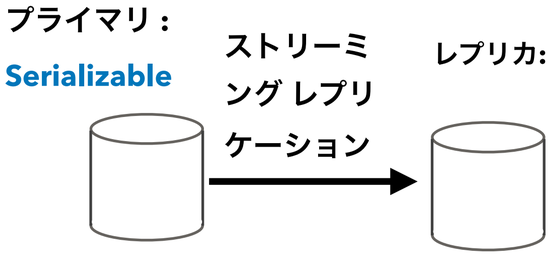


Read only Anomaly の起こりやすさ

- HTAPワークロードでの Read-only anomaly 発生率
  - 先ほどのストリーミングレプリケーション環境で 約10% の Read-only anomaly が発生。
    - HTAPワークロード下ではanomalyが起こりやすい。

❖ CHBenchmark (Scale Factor 10) での Read-only 異常の発生率

計測項目	OLAP クライアント数	OLTP クライアント数		
		16	24	32
Read-only anomaly サイクル発生率	16	9.3%	12.1%	13.6%
	24	7.9%	10.5%	12.3%
	32	6.6%	10.7%	12.5%
OLTP スループット (TPS)	16	3422.1	4329.1	4760.4
	24	3366.5	4334.7	4686.7
	32	3344.7	4148.7	4514.9
OLAP スループット (QPH)	16	11270	10917.3	10890
	24	11569.3	11881	11781.8
	32	11815.5	11577.5	11419.1



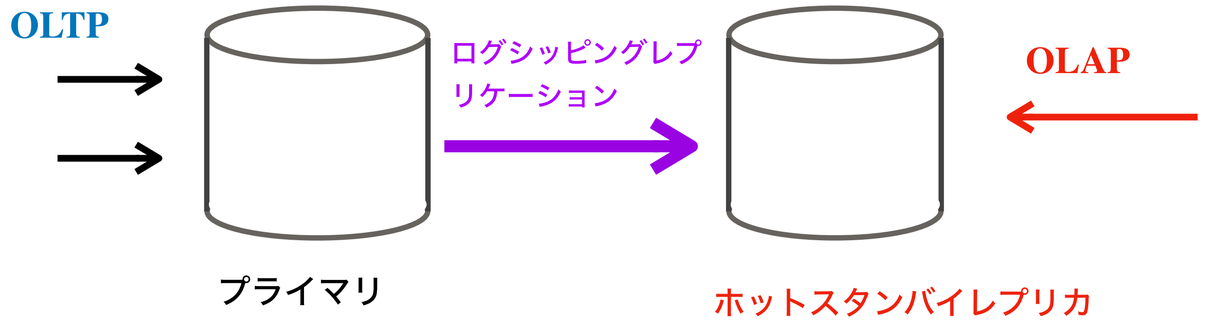
- シングルノード構成だと防ぎやすいが、分離型構成の方がSerializabilityの達成が困難

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

HTAP (Hybrid Online Transactional/Analytical Processing)

- 想定される利用例：直近で更新されたデータも含めてリアルタイムにデータ分析をしたい。
- 簡単にストリーミングレプリケーションでHTAP環境が作れる。（性能・フレッシュネス）
  - 前のページで見たように分離環境だとAnomalyが出てしまうため、性能・フレッシュネスだけでなく、データ整合性にも着目すべき。

❖ 負荷分散 (ホットスタンバイ)



- リアルタイムに分析したい。
  - 高いデータフレッシュネスを求める。
  - OLTPと分離、高いOLAP性能
  - Read only anomaly

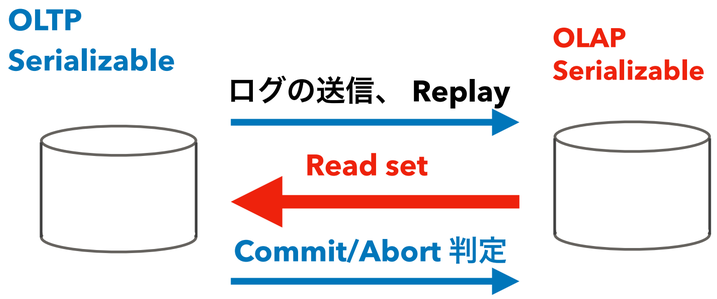
- こちらだけ Serializable (SSI)

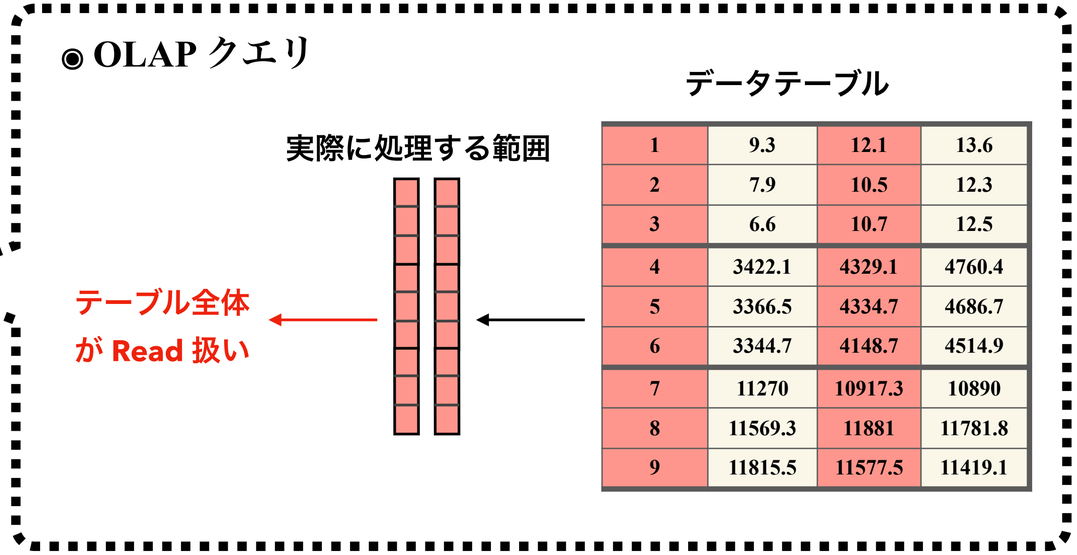
- レプリカ側で読んだだけで Anomaly が10%も起こる。
  全体のserializabilityが壊れる

HTAPの整合性保証と通信コスト

- 分離環境でデータ整合性を保証するには今読んでいるデータの上書きを通信して確認し合うことになりがち
  - 普通にSerializableにしようとすると、OLTP性能がダメになる。
    データ分析用のSQLで大量のデータを読むため、OLTP側で大量のトランザクションが アボート or ブロック される。
    - OLAP (Online Analytical Processing) データベースのテーブル全体をスキャンするようなクエリを発行し、データ分析に利用する。

❖ Serializable HTAP 環境のRead コスト



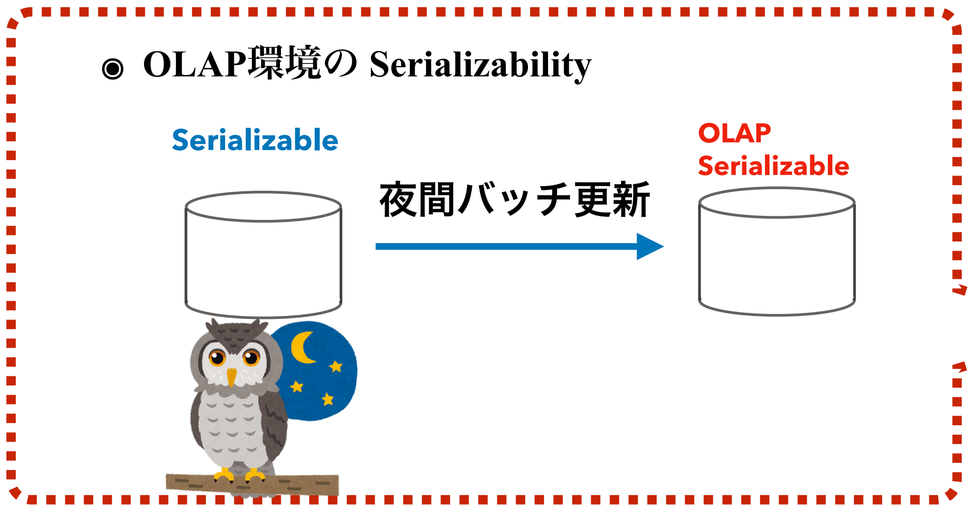


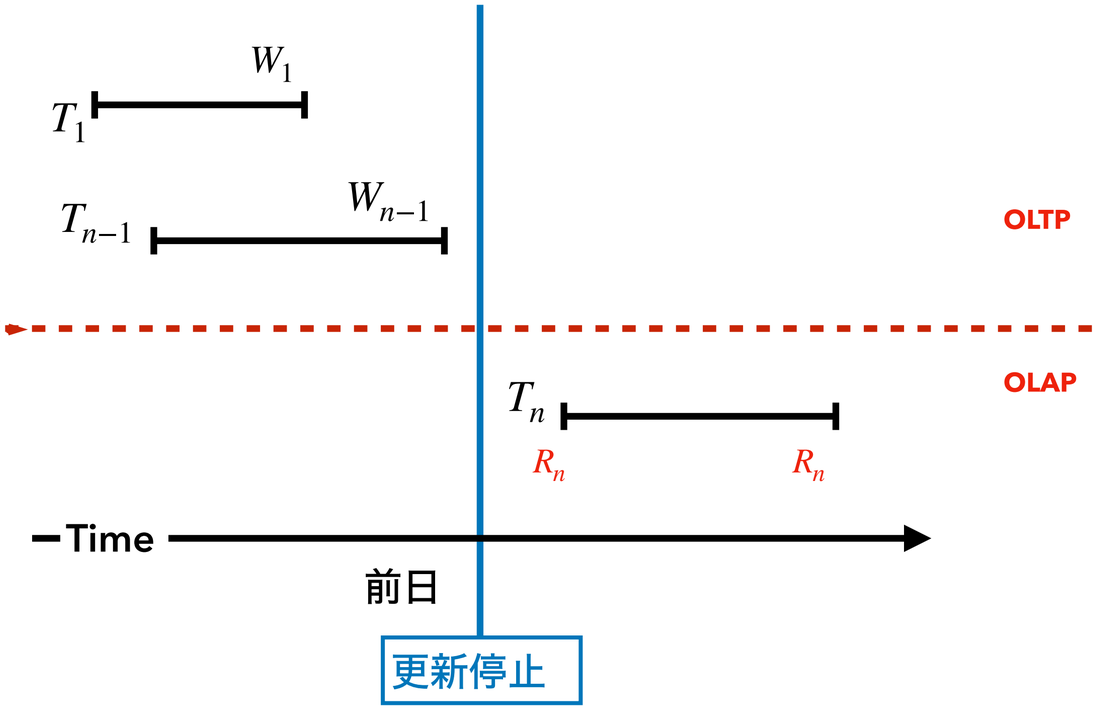
- OLAP の参加でOLTP側のコスト増大。
- OLTPを優先するとOLAP側がAbortされやすい

HTAP前：夜間バッチ更新によるOLAP側データの整合性

- 夜間バッチ：日中更新した前日分のデータを夜間にまとめて分析用システムにコピーする。
- OLAPの巨大な Read set を送り返す必要がない。
  - Serializableだが、一日前のデータしか分析できない。
    - データフレッシュネスがダメになる。

❖夜間バッチとデータ分析環境



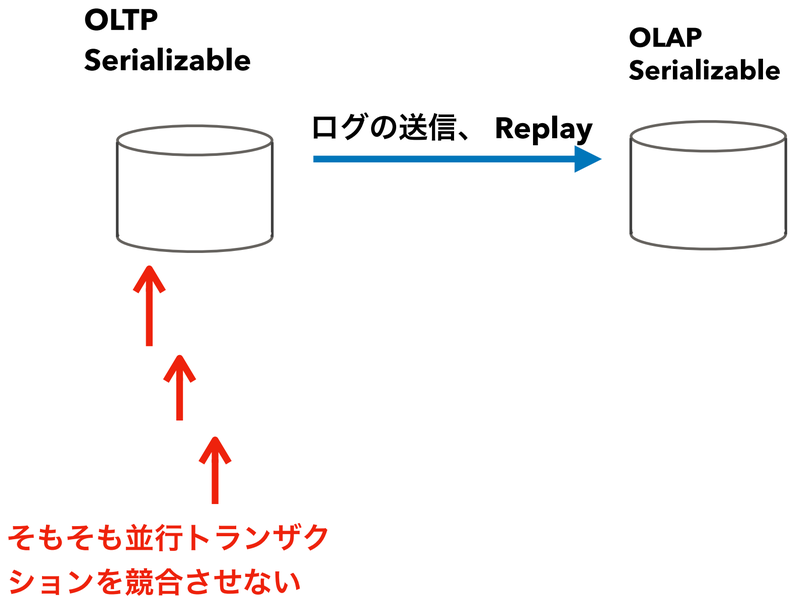


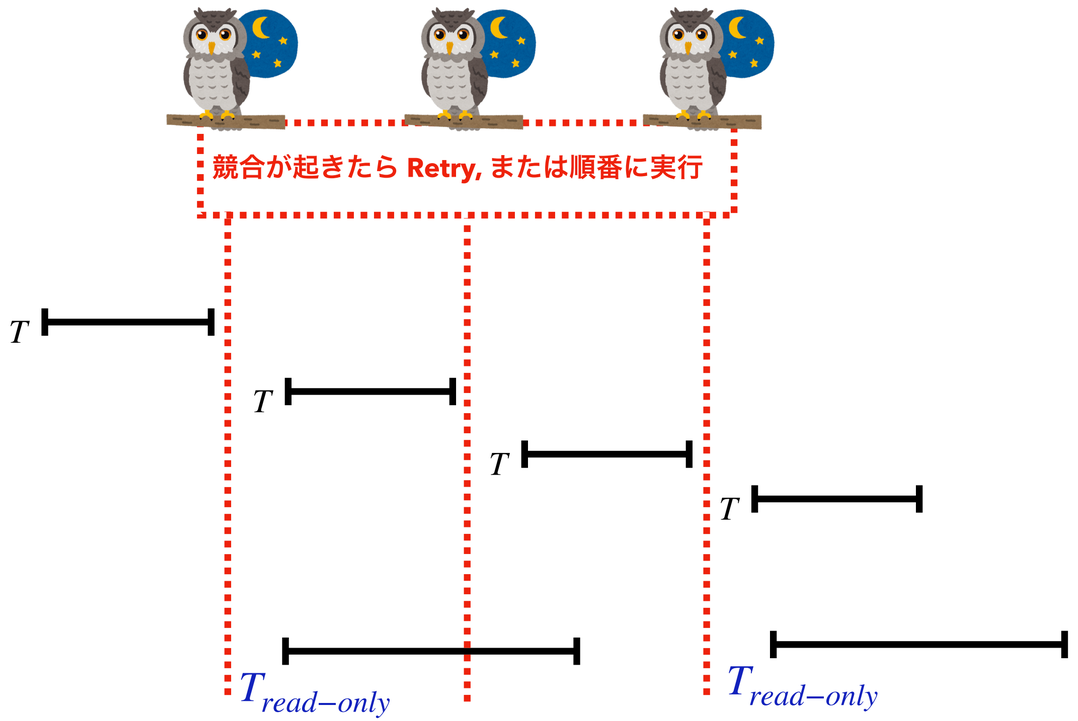
- 夜間バッチなどによる前日分までのデータ分析環境では Serializability が崩れない。

HTAP システムのSerializability

- HyPer [4, 5]：初期HTAPシステム、かつ、HTAP用にserializableの手法を作っていた珍しいシステム。
- 実質的に夜間バッチのような方法になってしまう。
- 逐次実行方式：2PL, 2PC (分散合意系) 二つとも同じような狭いスケジュール空間
  - 競合したらAbort/Retry、または 競合しないように待たされる。
  - または最初からWrite側を逐次実行させる。

❖一般的な Serializable HTAP システム





▸ OLAPをいつ始めてもAnomalyを起こさない

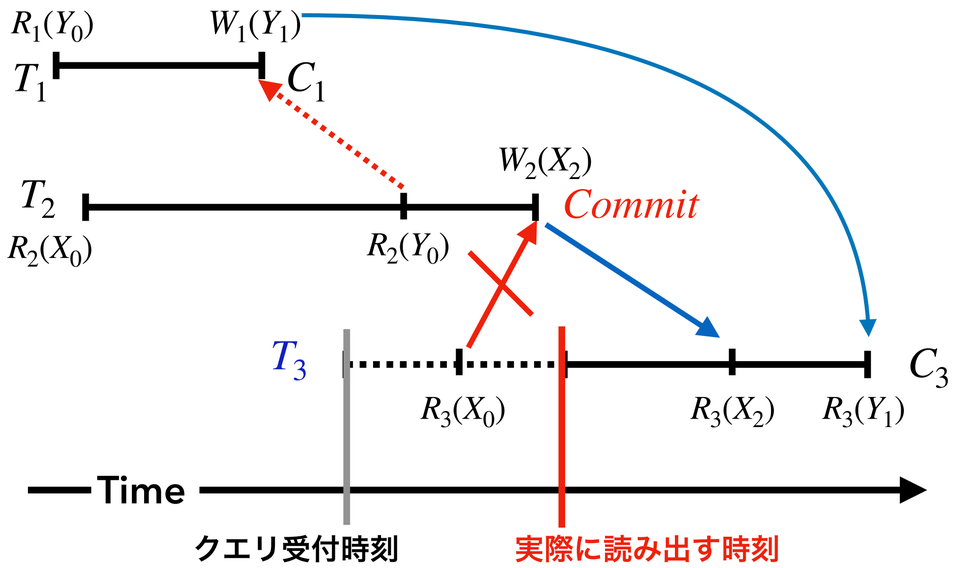
[4] H. Mühe, A. Kemper, T. Neumann, How to efficiently snapshot transactional data: Hardware or software controlled? in: Proceedings of the Seventh International Workshop on Data Management on New Hardware, DaMoN '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 17–26, http://dx.doi.org/ 10.1145/1995441.1995444.

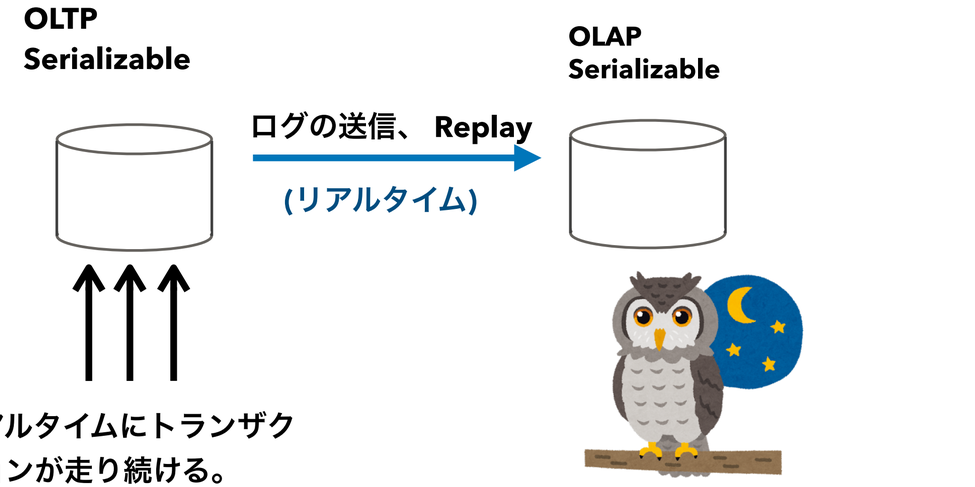
[5] T. Neumann, T. Mühlbauer, A. Kemper, Fast serializable multi-version concurrency control for main-memory database systems, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 677–689, http://dx.doi.org/10.1145/2723372.2749436

PostgreSQL-SSI-SafeSnapshot [3]

- Read-only 最適化：SSIのSI-Readフラグを取らないようにするもの。
- Read only transaction が結局Concurrent なトランザクションが終わるのを待って読むしかない。後から書いて競合が起きる可能性があるため。Concurrentなトランザクションが無限に終わらないと無限に待つ可能性がある。カスケードし続ける等の場合もあるため。
- そもそもストリーミングレプリケーション環境では機能しない。ストリーミングレプリケーション環境で実装するなら、レプリカ側でずっと待ち続ける、またはログを送り返してOLTP側を止めさせる必要がある。
  - OLAP 性能がダメになる。

❖ PostgreSQL-SSI-Safesnapshot



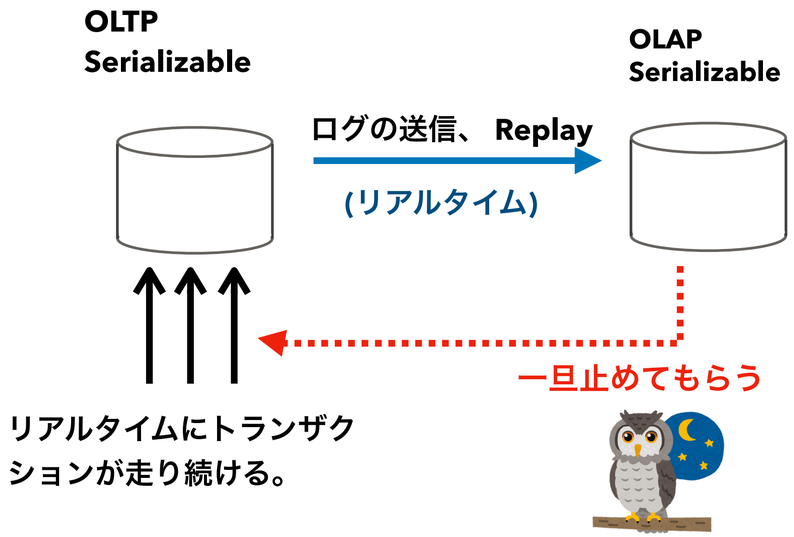


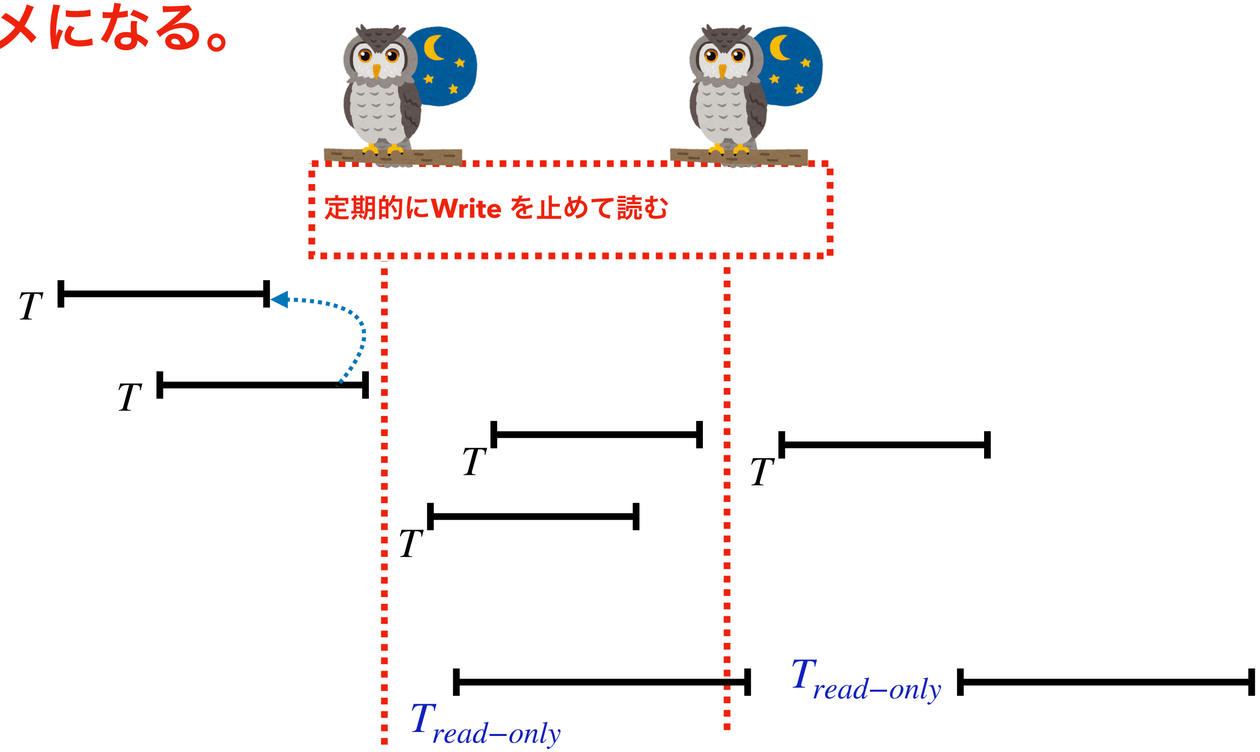
[3] Dan R. K. Ports and Kevin Grittner. 2012. Serializable Snapshot Isolation in PostgreSQL. *Proc. VLDB Endow.* 5, 12 (Aug. 2012), 1850–1861. https://doi.org/10.14778/2367502.2367523

PostgreSQL-SSI-SafeSnapshot [3]

- 整合性を保証しようとすると、実質的に夜間バッチのような方法になってしまう。
  - OLAP側でやり直しても同じ状況が続く。
  - 今度は、Writeトランザクションを止めて読む (PostgreSQL-SafeSnapshot, Silo-Snapshot Epoch)
  - OLTP性能とフレッシュネスがダメになる。

❖一般的な Serializable HTAP システム





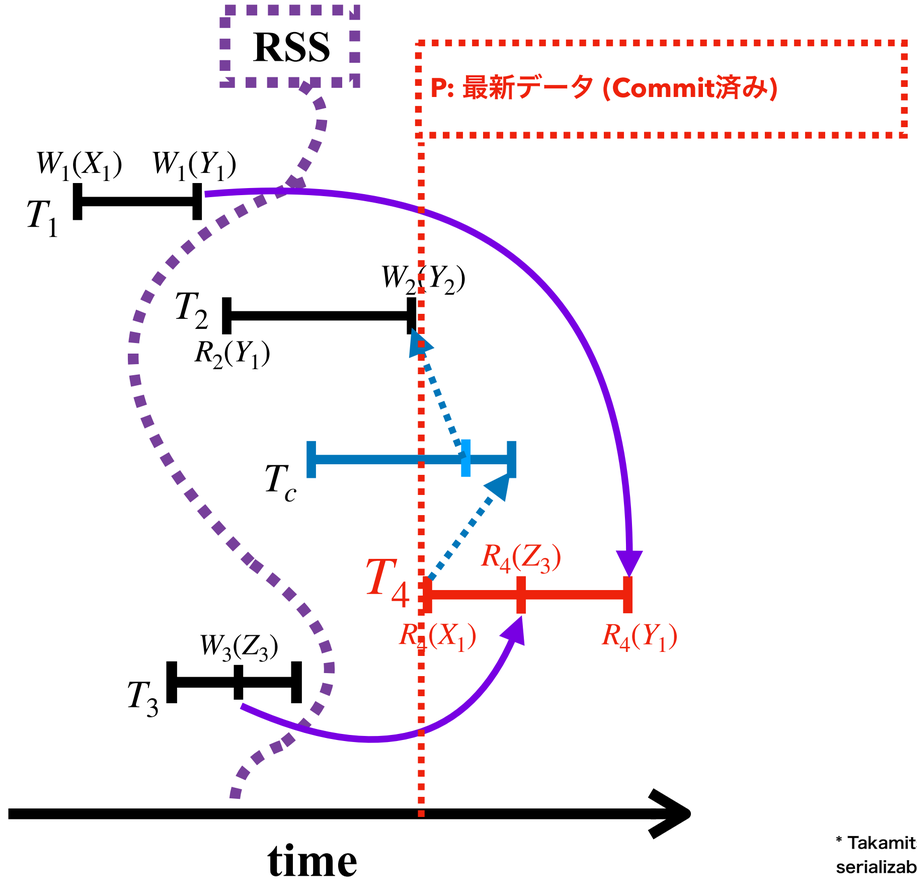
- OLAP側が無限に入り込めない可能性があるため。
- 過去のトランザクション群に競合しないデータを読む。

[3] Dan R. K. Ports and Kevin Grittner. 2012. Serializable Snapshot Isolation in PostgreSQL. *Proc. VLDB Endow.* 5, 12 (Aug. 2012), 1850–1861. https://doi.org/10.14778/2367502.2367523

提案手法：Read safe snapshot (RSS)

- Read only transaction が Serializable かつ最新のバージョンを読めるトランザクション群
  （定義や証明は論文をご参照ください）
  - 現在走っているトランザクションは無視できる：Concurrent トランザクションが何を読み書きして競合があっても、anomalyを起こさずSerializableかつ最新のバージョンを読む。Abort/Waitなしで読める。

❖ RSS; 読んでもSerializabilityを崩さないトランザクション境界



- Tc (Concurrent トランザクション) から競合が起きる可能性がある。
- RSSではTc からどんな競合が起きたとしても Serializableに読むことができる。
- 既存の方法ではAbort/Retry によってパフォーマンスが落ちやすい。

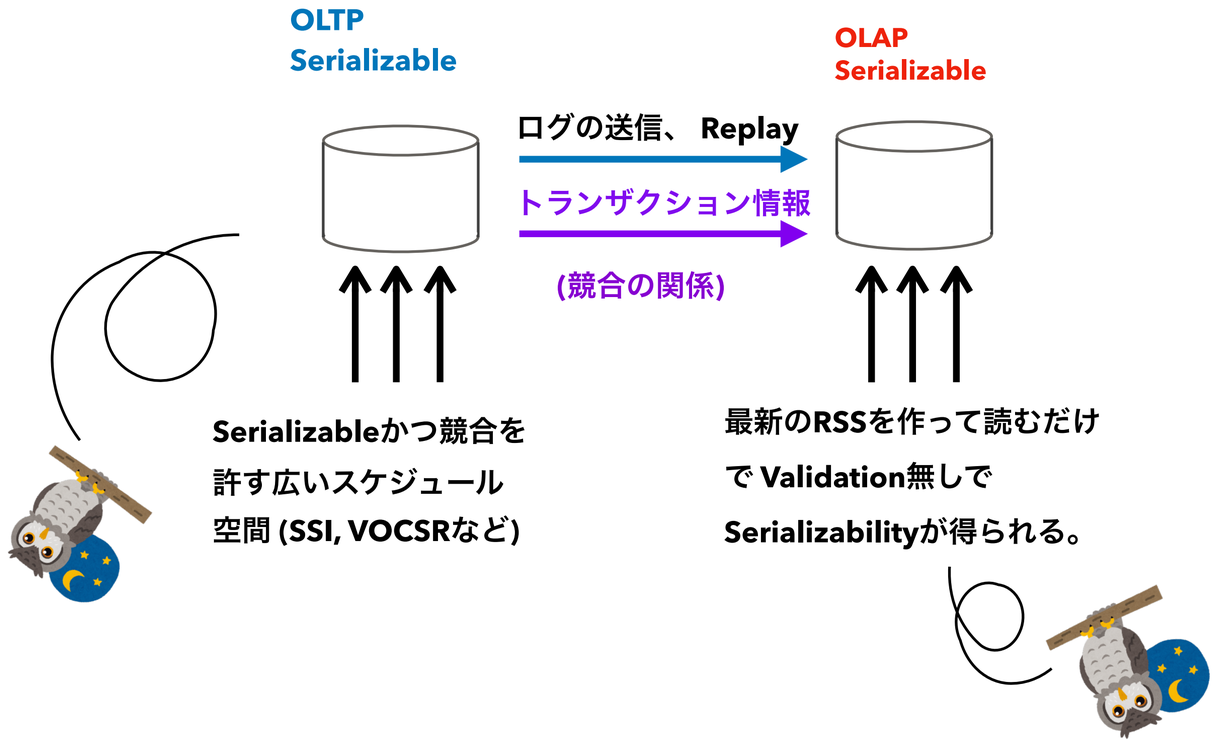
- RSS: どのトランザクションも read only トランザクションのために Abort/Wait させずに、Serializableに読める。
- 分散システムのConsistencyではAbort/Waitによって同じスケジュールは得られず、OLTP/OLAP分離環境にそのまま適用できない。

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

RSSのHTAPシステムへの適用性

- RSSをHTAPシステムに適用すると高性能かつSerializable化できる。例：postgreSQL-SSI
  - OLTPの並行処理パフォーマンスはSSIの方が2PL採用のシステムよりも理論上は高くなる。ただし、HTAP環境でSerializabilityを達成しようとするとSSIが仇になりOLAPが張り込む余地が無い。
    - RSSは HTAPと相性が良く、OLTP側はOLAP側を気にせずSerializableで自由に広いスケジュール空間のトランザクションモデルを採用できる。

❖ RSSベースの Serializable HTAP システム

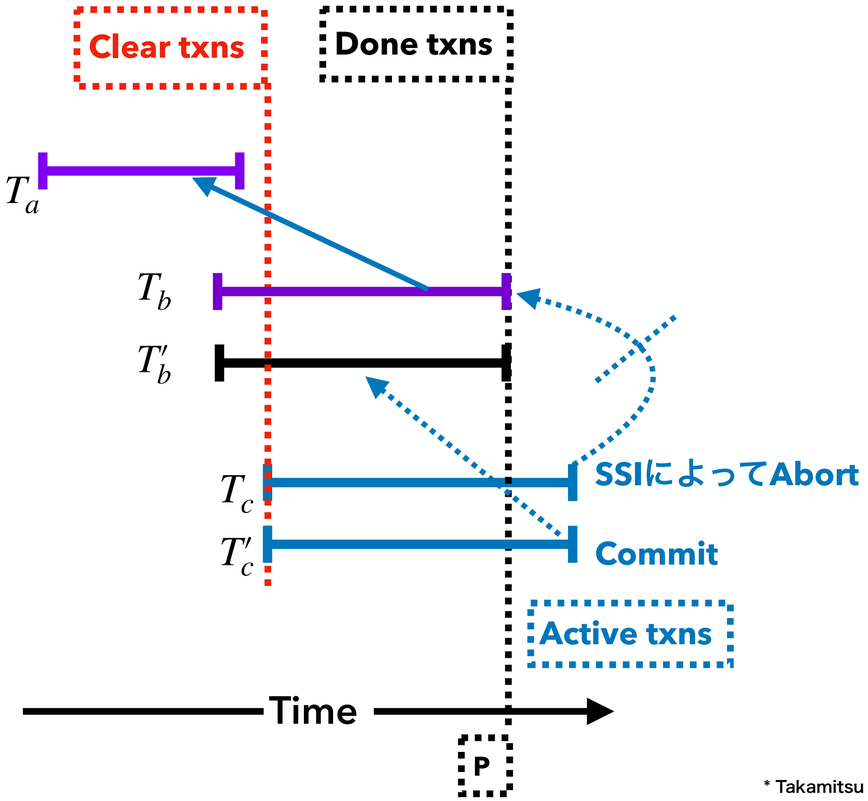


- 性能劣化が少なく、理論上最新のSerializableなデータが読める。
  - Validation-free: Serializability のためにOLAP側からOLTP側に送り返す情報が不要。
  - Abort/Wait-free: OLAPの参加によってOLTP側をAbort/Wairさせない。OLAP自身もAbort/Wait-free
  - 犠牲になるのはデータフレッシュネス（実験は後ほど）

SSIベースのRSS構築方法の提案

- SSIを前提にすると、より簡単にRSSを作ることができる。
  - RSSは理論上広い空間を前提としているため、SSIの実際のシステムで作る用に最適化する。
  - SSIのデンジャラスストラクチャーでAbortされるため、トランザクション群から一つ依存関係を取るだけで、簡単にRSSを構築できるモデルを提案。 (定義と証明は論文をご参照ください*)

❖ SSI下でのRSS構築



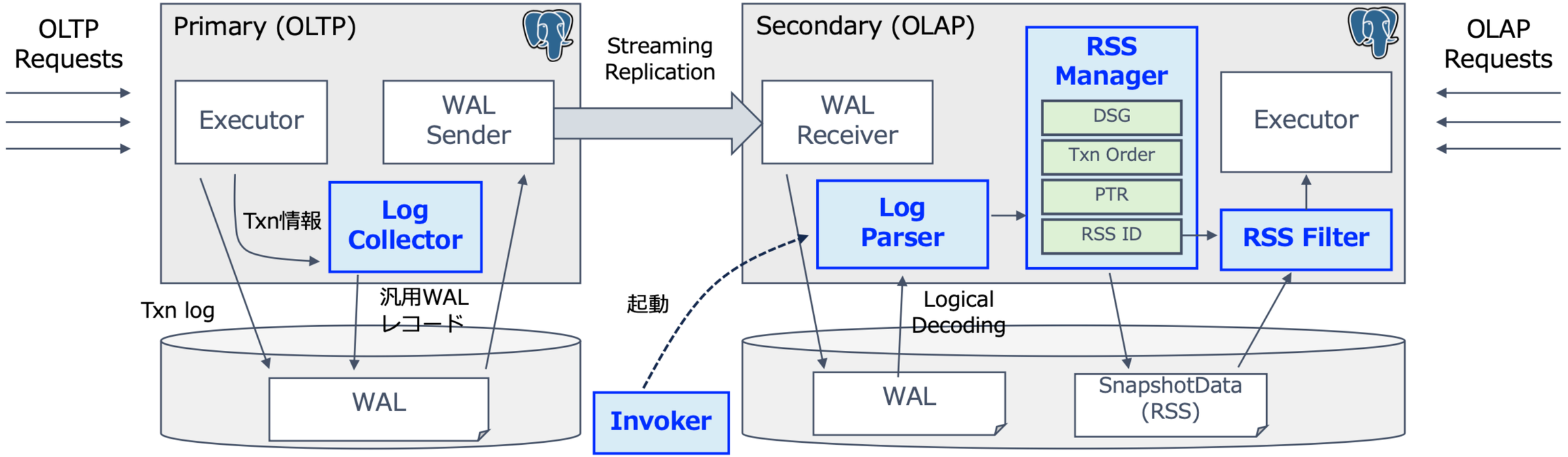
- Active txns : 開始してるが終了してないトランザクション群。
- Clear：Active txns から直接 競合が無いトランザクション群。
- Done : Commit 済みトランザクション群。
  - トランザクションが Active から Clear に到達するには Done を経由する。
  - SSIによって 競合が二つ連続するものは Abortされる。
  - Done から Clear に競合があれば、RSSに含める。
  - **SSIのシステムから簡単に RSS が作れる。**
  - **OLAPは最新のRSSを読むだけでSerializable**

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

PostgreSQL-SSI RSS

- PostgreSQL v12をベースにRSSのプロトタイプを実装
  - SSIの仕組みを利用することでRSSの実装を効率化することが可能
- Primary (OLTP) でトランザクション情報を汎用WALレコードとしてWALに書き込み、Secondary (OLAP) でロジカルデコーディングによりトランザクション情報を取得しRSSを構築
- OLAPクエリを処理する際にRSS Filterによりスナップショットが RSSに置き換えられる

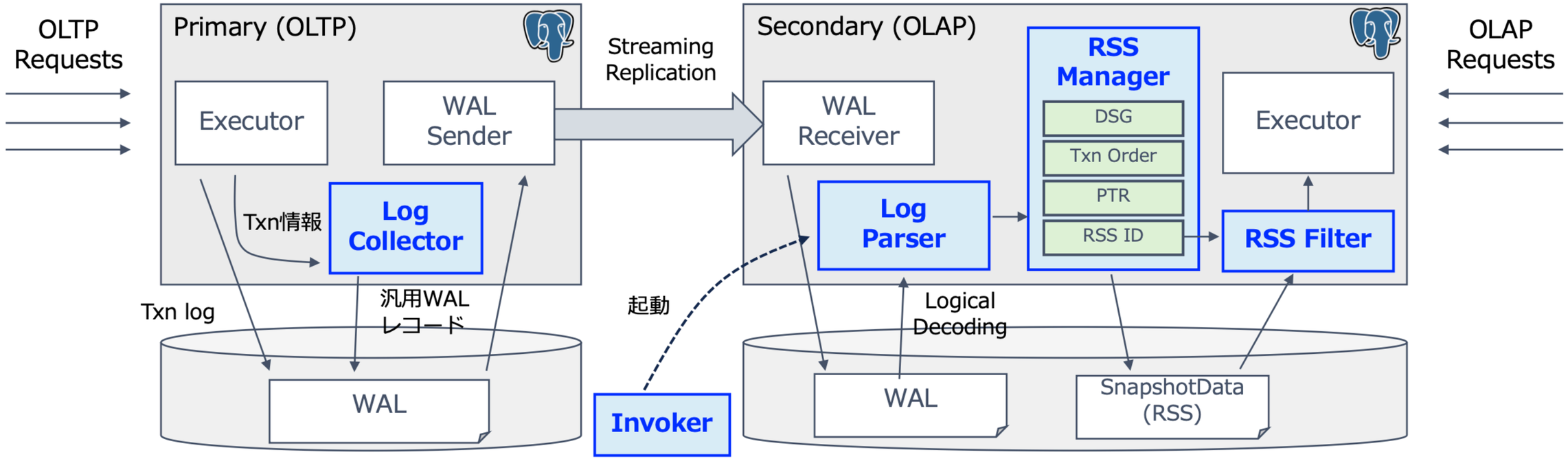
※ RSSの実装部分（+既存コードのごく一部を変更）



PostgreSQL-SSI RSS モジュール構成

- **Log Collector / Log Parser**：OLTPトランザクション情報の抽出・取得
- **RSS Manager**：依存グラフ(DSG)、トランザクション順序情報、RSS構造(PTR)の管理
- **RSS Filter**：RSSのSnapshotDataを取得しOLAPクエリのSnapshotに置換
- **Invoker**：周期的にLogcal DecodingおよびRSS構築処理を起動

※ RSSの実装部分（+既存コードのごく一部を変更）



Log Collector / Log Parser

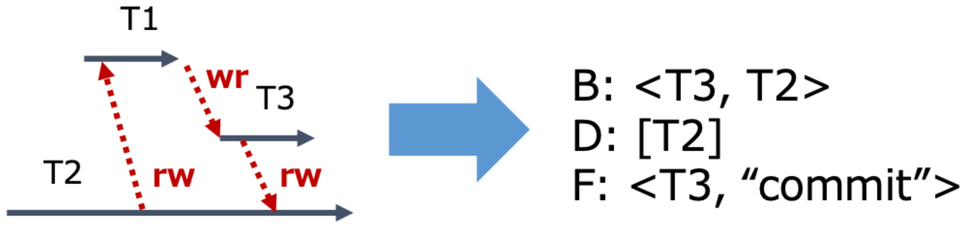
Log Collector

- Primary (OLTP) のトランザクション情報を抽出し汎用WALレコードとしてWALに書き出す
- Streaming Replicationによりトランザクション情報をSecondaryへ非同期に転送

トランザクション情報

- 開始情報(B) ：<xid※, xmin>
- rw依存情報(D)：[xid_i,xid_j, …]
- 完了情報(F)：<xid, {“commit”|”abort”}>

例：T3のTxn情報

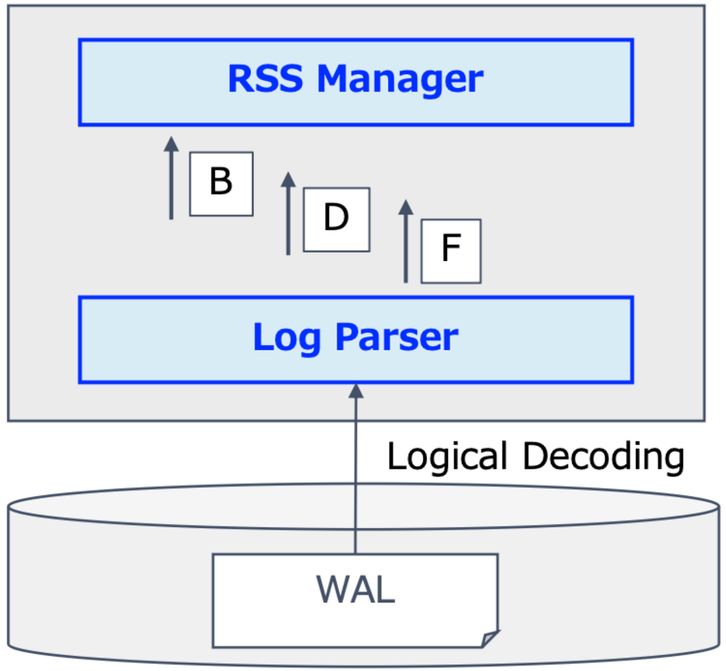


B: <T3, T2>
D: [T2]
F: <T3, “commit”>

※ Read only Transaction の場合は virtual xid を利用

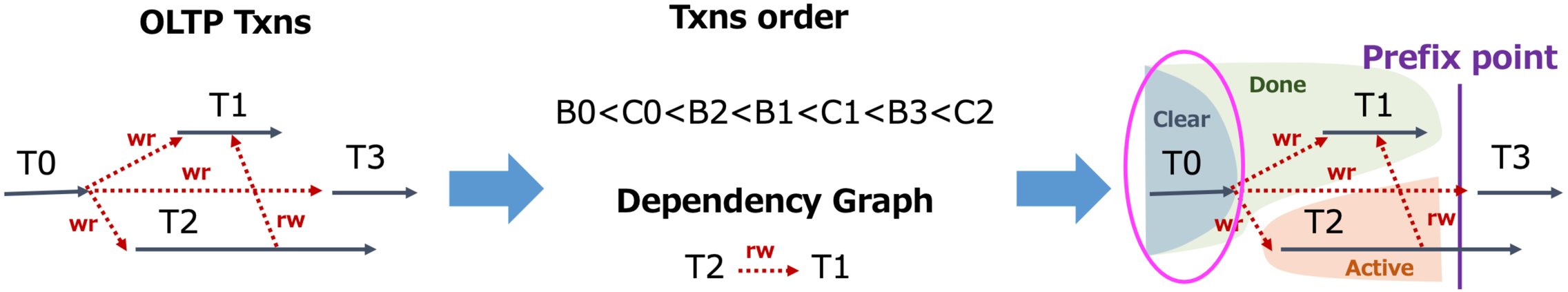
Log Parser

- 汎用WALレコードに格納されているトランザクション情報を解析し適切なデータ構造に変換してRSS Managerに渡す
- 汎用WALレコードの読み出しはPostgreSQLのLogical Decodingの仕組みを利用



RSS Manager：RSS構築

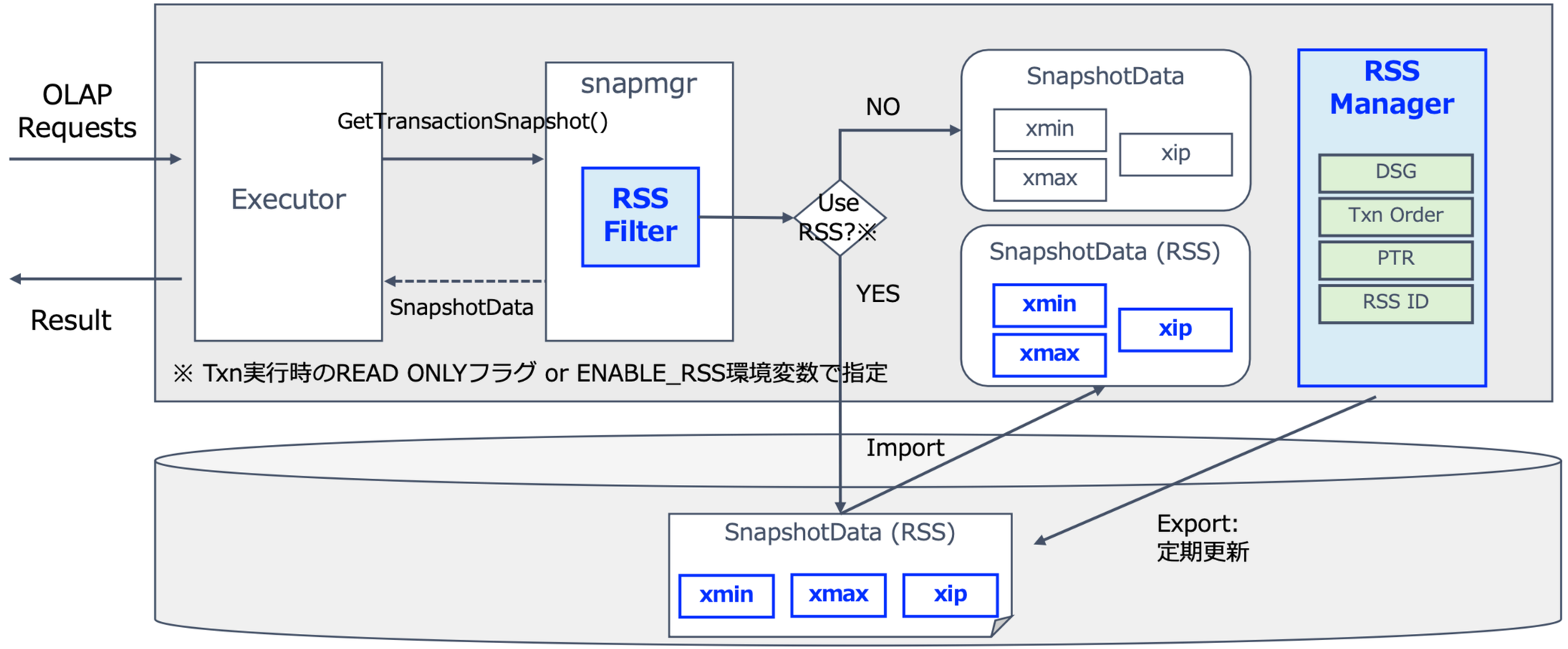
- トランザクション情報（開始/完了情報）からトランザクション順序を再構成
- 依存情報から依存グラフを作成
- トランザクション順序と依存グラフを使ってトランザクションを仕分け
  - **Active (P_i)** ：P_i 時点で実行中のトランザクションの集合
  - **Done (P_i)**： P_i 時点で完了(Commit/Abort) しているトランザクションの集合
  - **Clear (P_i)** ： P_i 時点で完了済かつ Active(P_i) から依存グラフで辿れないトランザクションの集合
  - **PTR※ (P_i)**：Clear (P_i) と、 Done (P_i) のうち、Clear (P_i) のトランザクションに依存があるもの



※ PTR: Protected Transaction Region

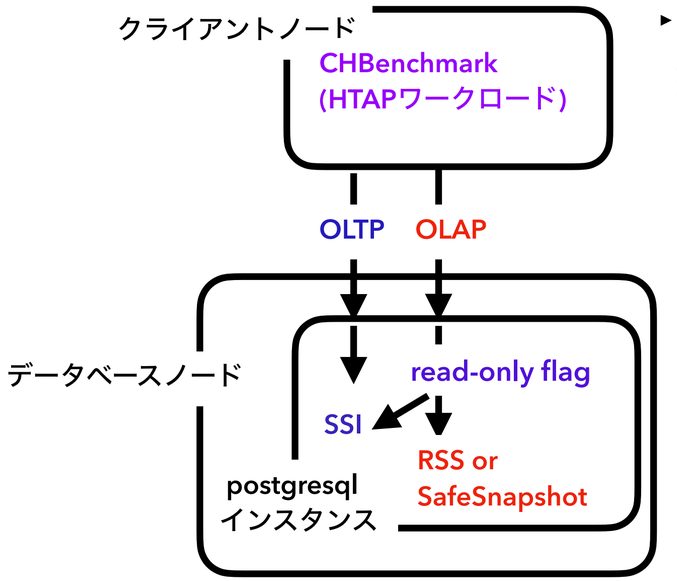
RSS Filter：RSSによるSerializableなRead

- SELECT実行時のSnapshot作成処理をフック（RSS FilterでRSSを使うかどうかをチェック）
  - RSSを使う場合はRSSのSnapshotDataをインポート
  - RSSを使わない場合は通常のSnapshotDataを使用



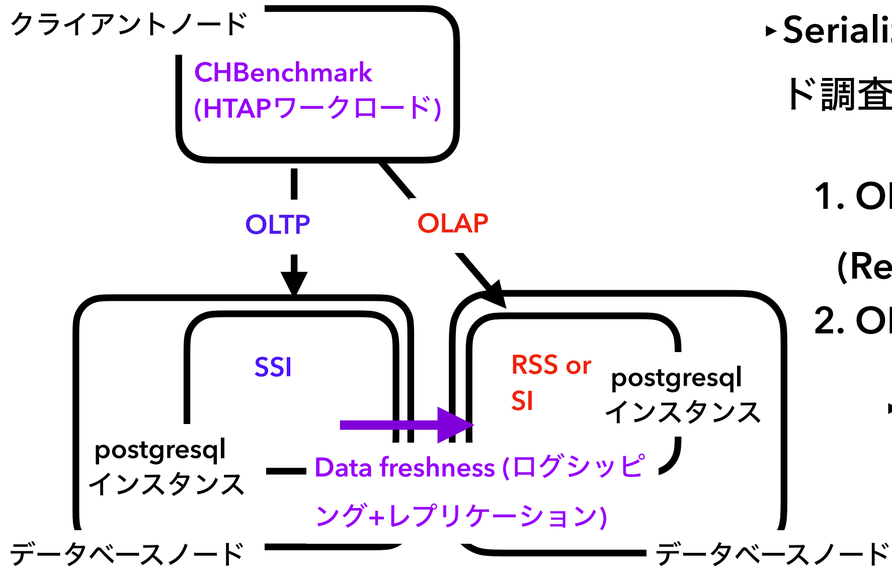
PostgreSQLベースラインとの比較

- シングルノード構成



- 各Serializableの実現方法の性能比較
  - 性能比較対象のシステム構成
    1. SSI のみでHTAP
    2. SafeSnapshotをOLAP側に使用
    3. RSSをOLAP側に使用

- マルチノード構成



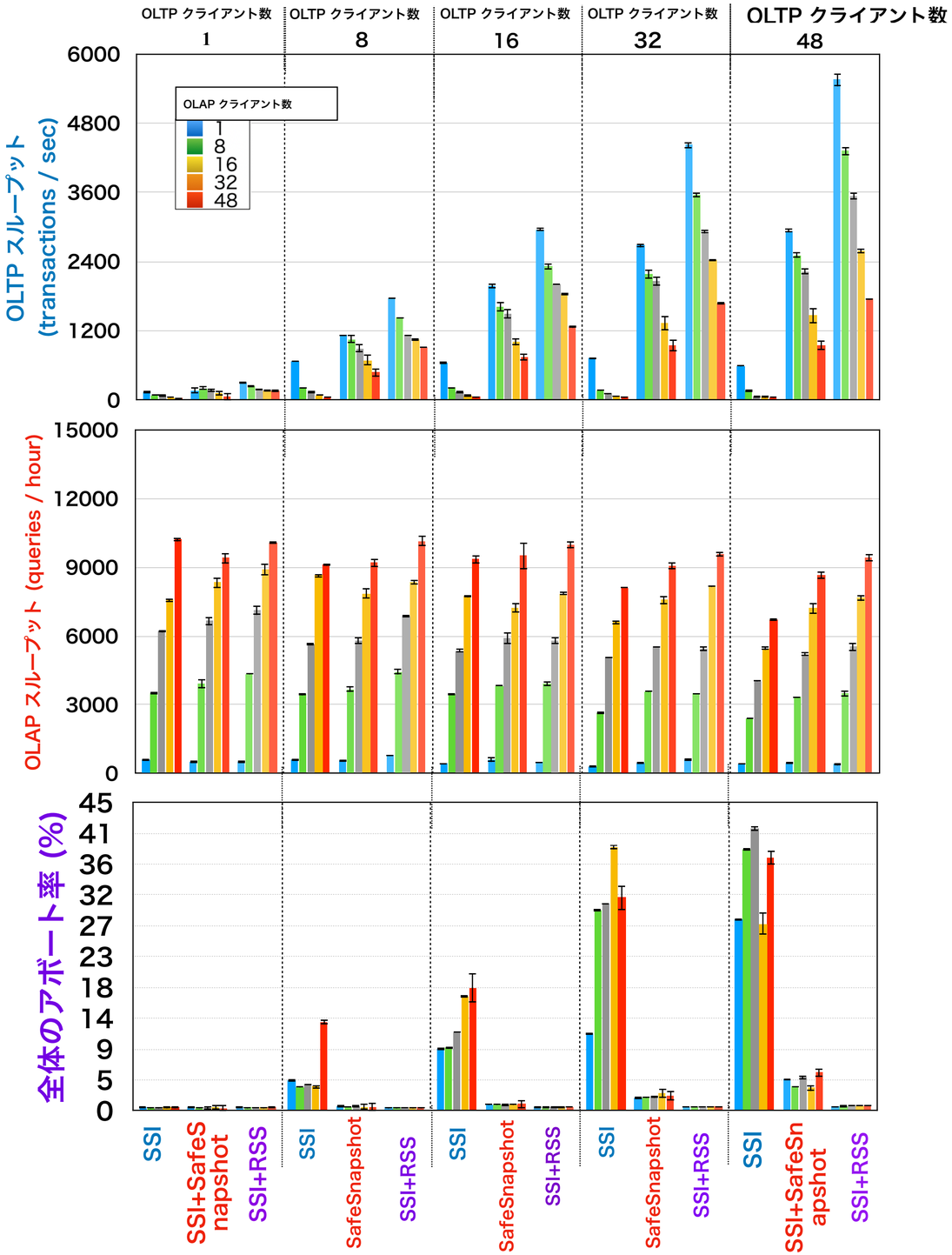
- Serializableのオーバーヘッド調査用システム構成
  1. OLAP側はSI (Repeatable Read)
  2. OLAP側をRSS
     - RSSのみSerializable

マシンスペック

- クライアントノード
  - CPU： Intel(R) Xeon(R) Platinum 8176 2.10 GHz * 4, 28 物理コア, 2ソケット
    - キャッシュ：32-KB L1i, 32-KB L1d, 1024-KB L2, 39.424-MB L3
  - DRAM : 512GB
  - Storage : 440GB SSD
  - OS：Ubuntu 18.04.3

- データベースノード (2ノード共通)
  - CPU：Intel(R) Xeon(R) Platinum 8176 2.10 GHz * 2, 28 物理コア, 2ソケット
    - キャッシュ：32-KB L1i, 32-KB L1d, 1024-KB L2, 39.424-MB L3
  - DRAM : 1TB
  - Storage : 440GB SSD
  - OS：Ubuntu 18.04.3

各Serializable実現手法の性能比較



- シングルノード構成: SSI, SSI+SafeSnapshot, SSI+RSS (CHbenCHmark, サイズ[SF100]、実行時間[300秒間]、並行実行クライアント数[1,8,16,32,48]の組み合わせをそれぞれ10回ずつ実行)

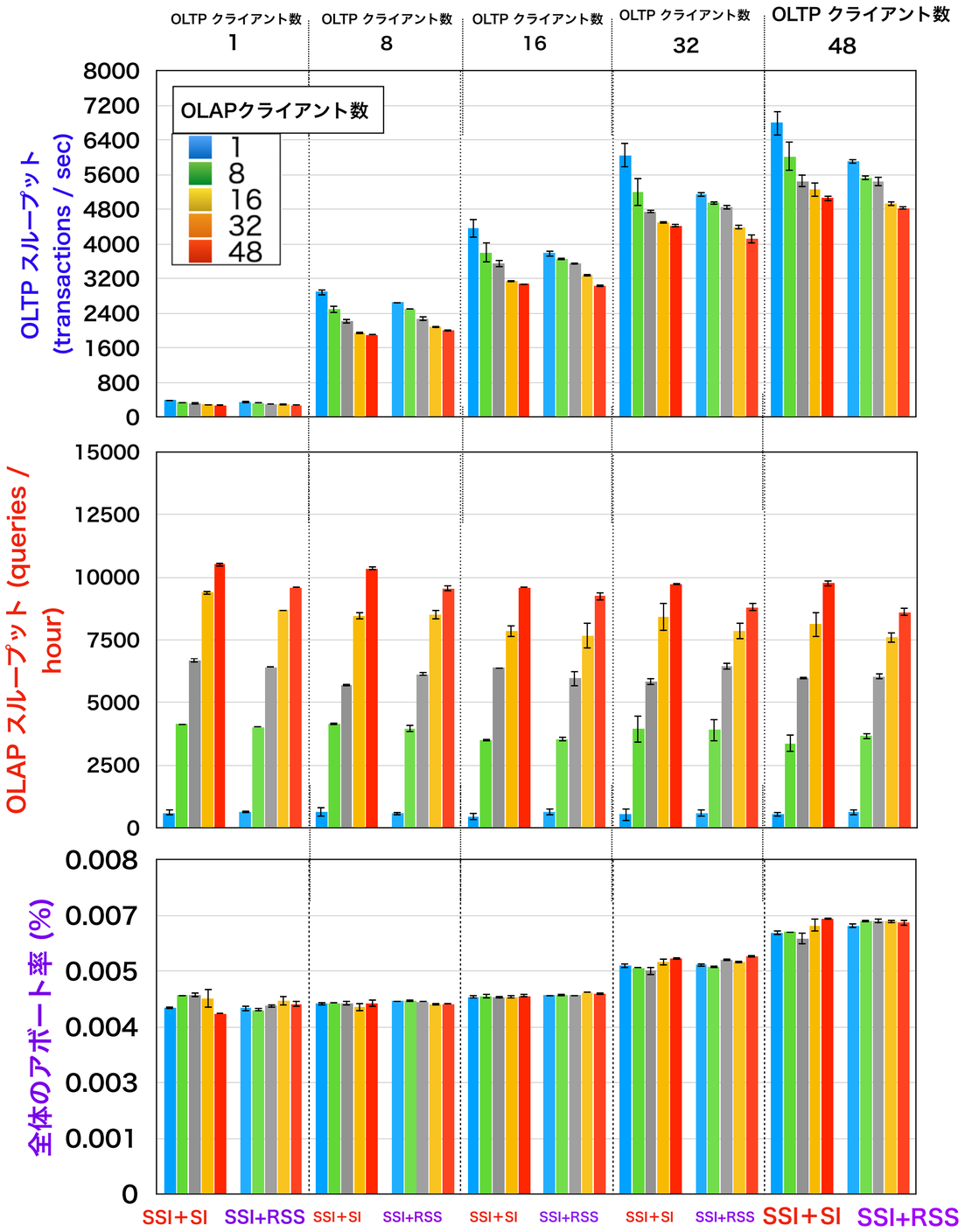
- SSI : OLAPクエリが一つでも実行されると Abort率が増加しOLTP側の性能が大きく劣化する
  - ‣HTAPとSerializableは相性が悪い

- SSI-SafeSnapshot: ベンチマークの性質上 OLTP側トランザクションがすぐに終了するためOLAP側に影響が出難い。

- RSS: OLAP側からOLTPへの影響が少ないため、Abort率が少なく、OLTP側の高い性能を維持できる。

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385

RSS実装のオーバーヘッド調査



- マルチノード構成: SSI+SI, SSI+RSS (CHbenCHmark, サイズ[SF100]、実行時間[300秒間]、並行実行クライアント数[1,8,16,32,48]の組み合わせをそれぞれ10回ずつ実行)

- SSI-SI : OLAP側がSerializableではないため PostgreSQLのSerializableの本来の性能がそのまま出る。

- SSI-RSS : SSI-SIから約10~17%ほどのOLTP性能が低下でHTAP全体のSerializableを実現。 Read-only anomaliesが発生しないことを確認。

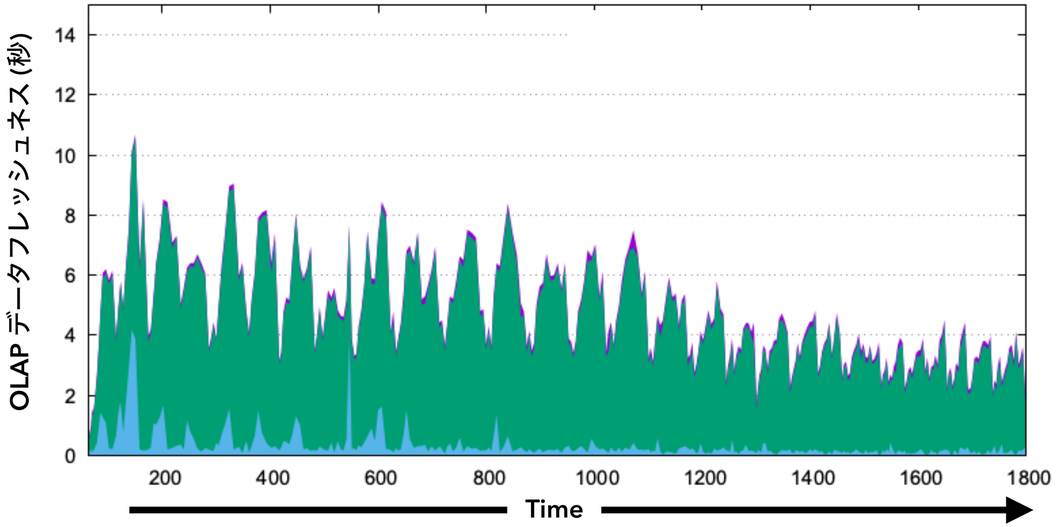
  - PostgreSQIレプリケーションの特性：リードレプリカ側で古いデータを読んでいるとプライマリ側でそのデータを残す必要があり、インデックスとバージョントラバースの影響でOLTP性能が落ちる。

データフレッシュネスの評価 (マルチノード構成)

- データフレッシュネスって何：OLTPでコミットされたデータをOLAP側で読むまでの遅延時間。
  先のマルチノード構成の実験を30分間実行
- マルチノード構成のOLAP用ノードで SI or RSS を通して読み、その中の最新コミットタイムスタンプを持つタプルの持刻と実行時の現在時刻の差を全体のフレッシュネスとして測り、SIとRSSの差を見る
  - ❖レプリカ上でRSSを構築して読むまでの時間の内訳

計測項目	平均(秒)	最小遅延(秒)	最大遅延(秒)
SI Snapshot を読むまでの遅延	0.37	0.009	4.175
ロジカルデコーディング時間	3.667	0.074	8.233
RSS用ログのデコード時間	0.281	0.006	2.057
RSS構築時間	0.129	0.039	0.654
Serializableかつ最新データと現在時刻の差	0.029	0.004	0.259
RSS全体のデータフレッシュネス	4.414	0.374	10.689

- SI (Non-serializable)で読んだ最新データの遅延は最大4.17秒
- PostgreSQLのロジカルデコーディング機能を使用：
  RSS作成用のログをデコードするのに掛かる最大時間は2.06秒
- RSS構築時間：シングルノードの場合はこの程度の遅延で済む。



- 夜間バッチ更新のフレッシュネス：前日
  - ➜ 普通のSerializableなOLAP、HTAPシステムではOLTPで性能劣化
- RSSのフレッシュネス： SerializableかつOLTP性能劣化も少なく、平均4.41秒。遅延時間の多くは実装の簡略化のために使用したロジカルデコーディング
  - ➜ PostgreSQLの汎用WALだけを読み出すために物理WALも走査する不要な遅延。

PostgreSQLレプリケーション環境の感想

- PostgreSQL12.0 レプリケーション環境の設定と性能への影響
  - WAL conflict：リードレプリカ側で読むデータがプライマリ側で消されているとOLAP側でAbortされる。RSSでは理論上Serializableかつ最新データを読むが、PostgreSQLではHOTやautovacuumで消されるため対策が必要だった。vacuumもオフ。
    - hot_standby_feedback：これだけ設定しても WAL conflictが起こるケースがあった。
    - replication slot ：これが設定されてないと最初のロード後に空のデータを読んでOLAPが超速く見えるケースがあった。
  - リードレプリカ側で古いデータを読んでいるとプライマリ側でそのデータを残す必要があり、インデックスとバージョントラバースの影響でOLTP性能が落ちる(20%)
    - たった20%の低下でPostgreSQLでSerializableなリードレプリカが作れる。
  - TPC-C の性能は競合が起こりづらいため当てにならない。PostgreSQLならもう少し複雑なOLTPベンチマークの方が、他のデータベースよりもSSIの利点が出やすい。
  - PostgreSQLは改造しやすいが、最新のオープンソースデータベースが増えている。

まとめ

- 我々の提案したRSSという理論に基づいて、PostgreSQL-SSIの非同期ストリーミングレプリケーション環境に適用すると、Serializableかつ高性能なHTAPシステム化が実現可能。
  - Serializable かつ高性能なHTAPはすごく難しい。
    - HTAPの３つの軸：OLTP/OLAP性能、データ整合性保証、データフレッシュネス
    - この中で見落とされていたデータ整合性もしっかりと理論で保証しつつ高い性能を実現。
      - ただ古いバージョンを読むだけではSerializableにならない：理論上の整合性保証が必要→RSS
      - Tsurugi DB * のような理論上広いトランザクション空間だとRSSの理論的枠組みが適用可能、Serializableかつ高性能なHTAP環境が実現可能となってきている。

* https://www.tsurugidb.com/

謝辞

- *この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP16007）の結果得られたものです。

- また、博士課程の長い間ご指導を賜ったノーチラステクノロジーズの
神林飛志 さん、荒川傑 さん、黒澤亮二 さん、
東京科学大学 (旧東京工業大学)の
横田治夫 先生、宮崎純 先生に、心より感謝を申し上げます。

* Takamitsu Shioi, Takashi Kambayashi, Suguru Arakawa, Ryoji Kurosawa, Satoshi Hikida, Haruo Yokota : Read-safe snapshots: An abort/wait-free serializable read method for read-only transactions on mixed OLTP/OLAP workloads. Inf. Syst. 124: 102385 (2024). https://doi.org/10.1016/j.is.2024.102385