NTT

NTT DATA

書き込みスケールを実現した Postgres-XC クラスタ

PostgreSQL Conference 2011
25th February, 2011

Koichi Suzuki

講演内容

- Postgres-XC の考え方とゴール
- 読み書き双方をスケールさせる方法
  - グローバルトランザクション制御 (分散MVCC)
- Postgres-XC の構造
- 実装と検証の現状
- 課題とロードマップ

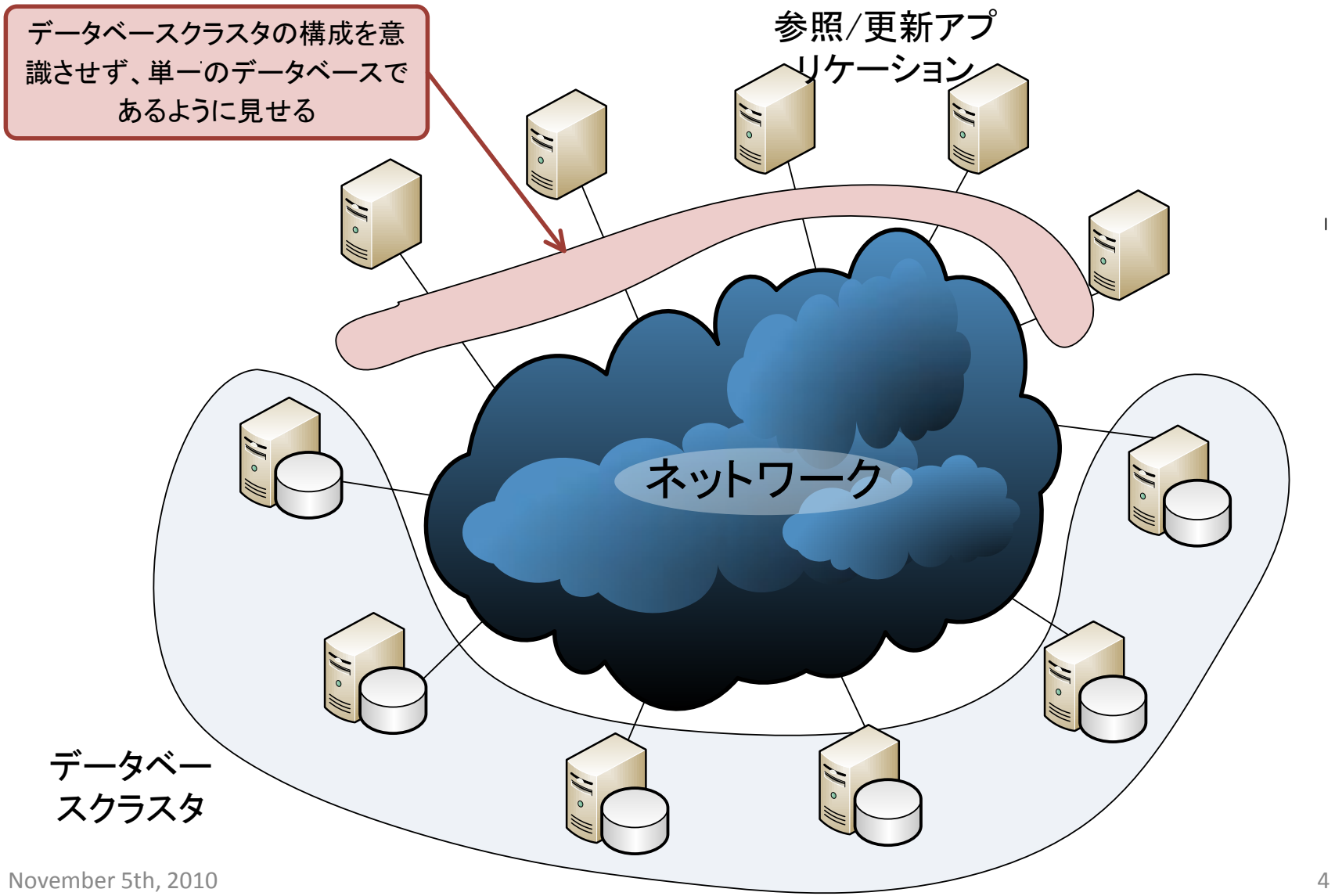
Postgres-XC (Extensible Cluster) とは？

- 読み書き双方をスケールさせる PostgreSQL クラスタです
  - 5台のサーバで、PostgreSQL単体に比べ約3倍性能がスケールします(DBT-1)
  - OSSでは唯一のソリューションです
  - 商用DBでもほとんど例がありません

- どのサーバとでも接続できます
  - どのサーバに行った変更も即時に他のすべてのサーバから見えるようになります.
  - 単純な「レプリケーション」ではありません
    - テーブル毎にレプリケーションさせるか、分割配置するかを指定できます。

- NTT とEnterpriseDB が協力して実装を行ってます
  - 2010年5月に、PostgreSQLコミュニティ会合で発表、大きな反響をよびました。
    - XC の実装内容で、PostgreSQL本体にも活用できるものが出てきています。積極的に活用していきます。
  - 2011年3月末にはV.1.0をリリース予定です。

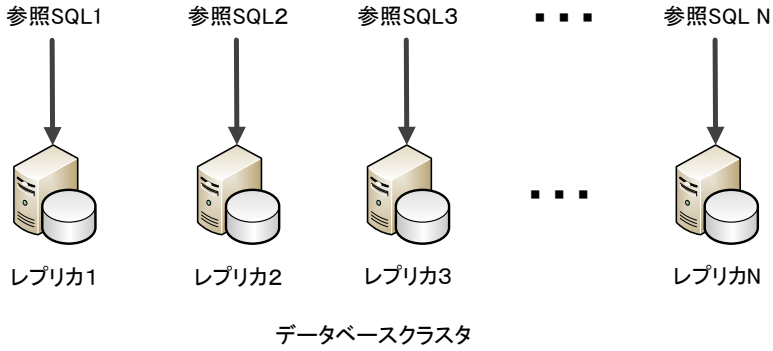
Postgres-XC の目標



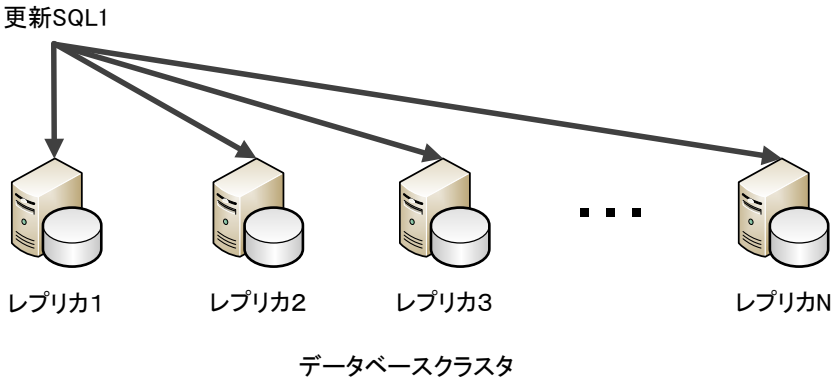
現状

- Version 0.9.3 – 10/29 開放
  - 開発ページ: http://sourceforge.net/projects/postgres-xc/
  - プロジェクトページ: http://postgres-xc.sourceforge.net/
  - データベース間の結合演算を含むSQL文のサポート
    - 一部制約あり。WITH句など。
  - ユーザ定義関数
  - C/Java インタフェース (カーソルの機能の一部は年末予定)
  - クラスタ構成及び設定ユーティリティ
  - ダンプ・リストア
  - １週間連続運転での安定性確認 (DBT-1)

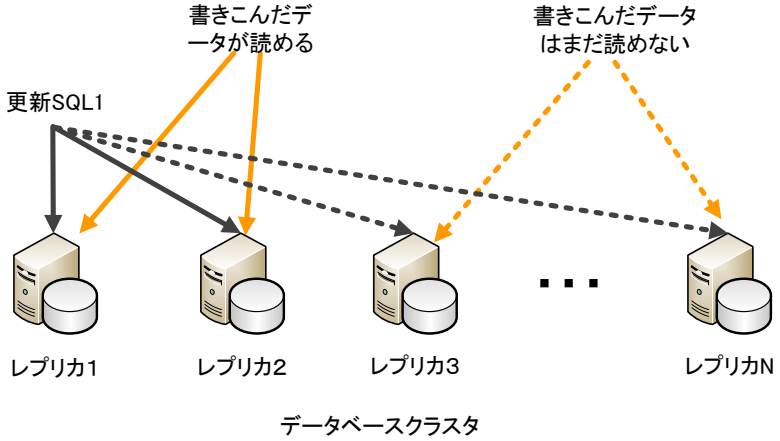
レプリケーションでも参照スケールは可能だが、、、



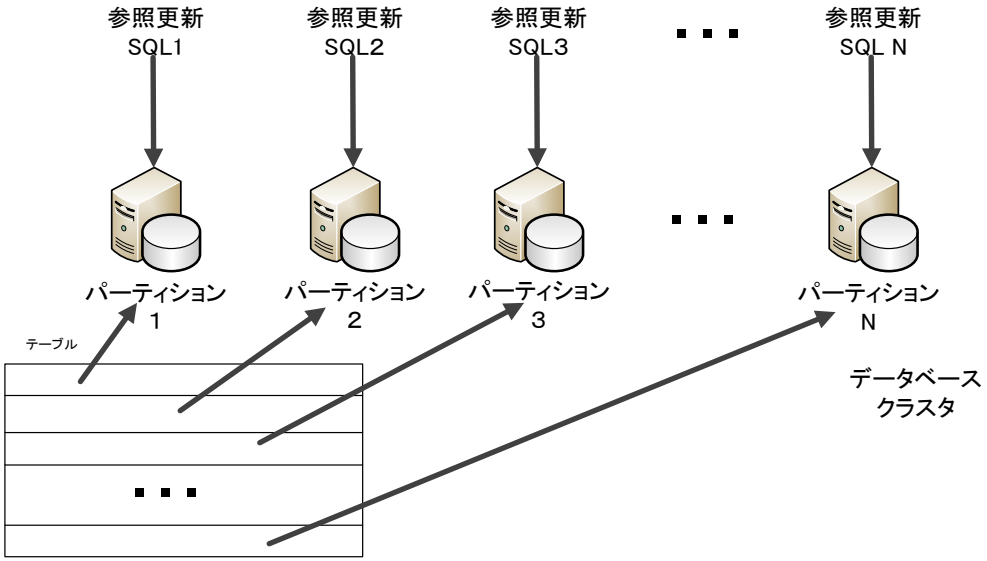
書き込みスケールは難しい



書き込み同期の問題も



各データベースにテーブルを分割



これでクラスタ全体がうまく同期できれば、、、

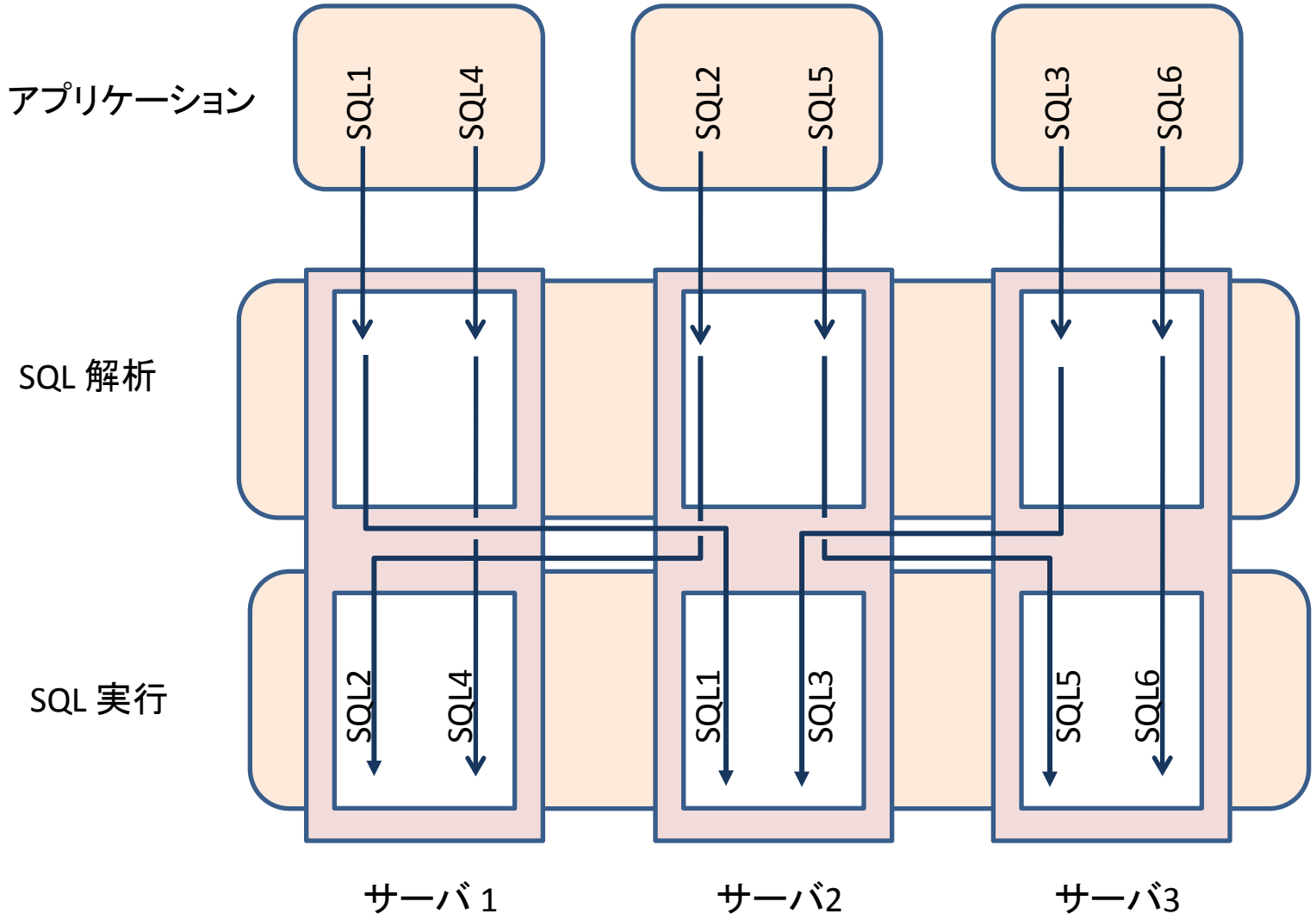
PostgreSQL のトランザクション制御を取り出してクラスタ全体に適用する

PostgreSQL はこれが可能な構造をしている

MVCC(*1)

(*1) Multi-Version Concurrency Control

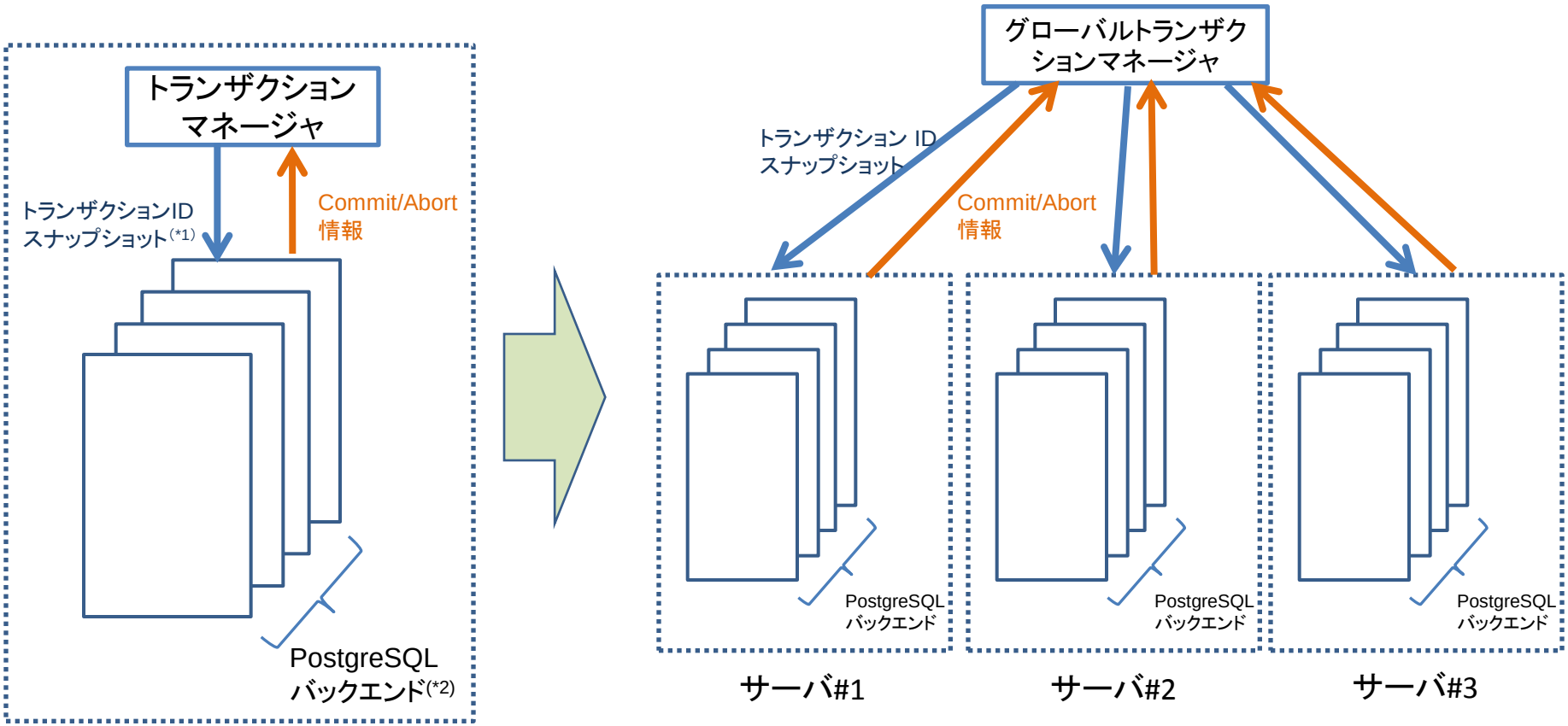
トランザクションの並列実行の様子



分散MVCCの概要

スケールアウトを可能とするキーメカニズム

PostgreSQL のトランザクションを取り出して、クラスタ全体を制御させる



(*1) 実行中のトランザクションのリスト
(*2) それぞれがトランザクションを処理しているプロセス

読み書き双方をスケールさせる方法

- 並列処理
  - データベースクラスタ内でトランザクションを分散並列実行
- 分散トランザクション制御、MVCC
  - トランザクションタイムスタンプ (トランザクション ID)
  - MVCC の可視性制御
- グローバルな値を供給
  - シーケンス
  - タイムスタンプ

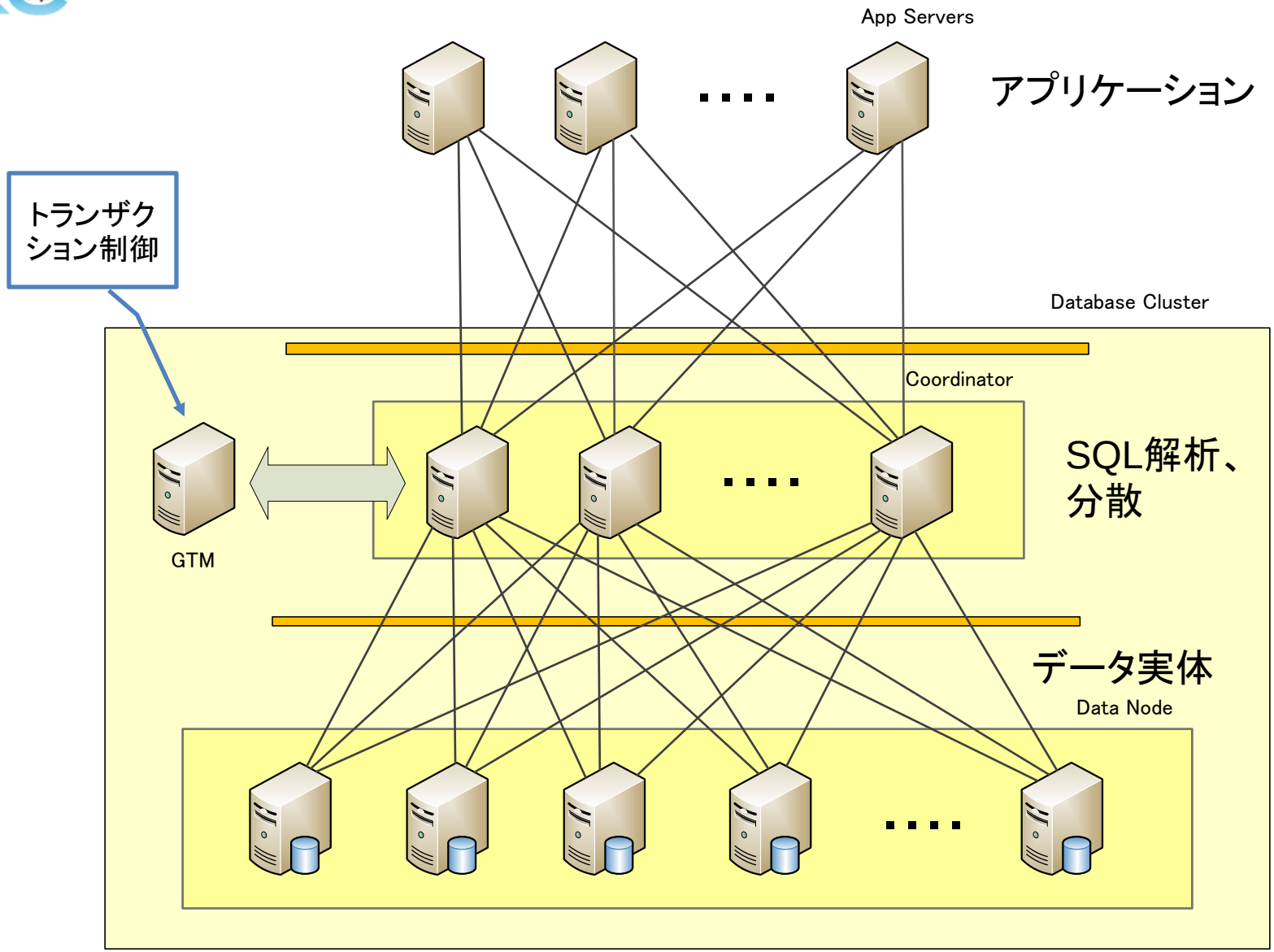
Postgres-XC でのテーブル設計

- レプリケーションするか分割するかを選択する
  - レプリケーションテーブル (replicated table)
    - テーブル全体を各データノードでコピー
    - レプリケーションは、SQL文ベースで一貫性を維持
  - 分割テーブル(distributed table)
    - 各行ごとにどのデータノードに格納するかを決める分割キー(distribution key)のカラム値で決める
      - » ハッシュ
      - » ラウンドロビン
      - » レンジ (未実装)
      - » ユーザ定義(未実装)

テーブルの分割・レプリケーションの決め方

- 更新が頻繁なトランザクションテーブルは分割する→ 各トランザクションを実行するデータノードを少なくできる。→ 書き込みスケール
- 安定しているマスターテーブルはレプリケーションにする→ 読み出しスケール。

Postgres-XCのリファレンスアーキテクチャ



Postgres-XC のコンポーネント

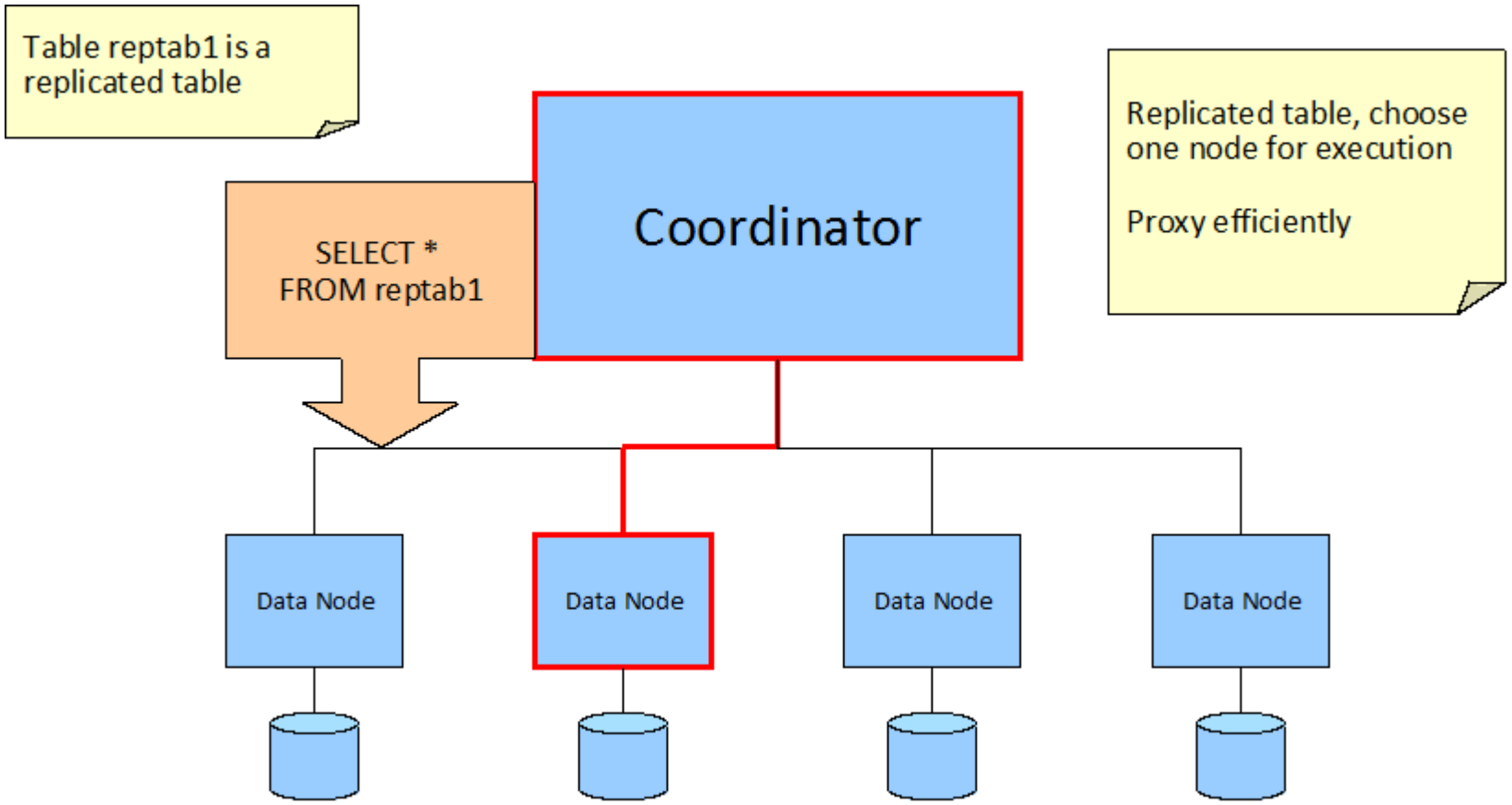
- GTM (グローバルトランザクションマネージャ)
  - クラスタ全体の一貫したトランザクション管理
    - トランザクション ID
    - スナップショット
  - クラスタ全体で一貫した値の提供
    - シーケンス
    - タイムスタンプ
- コーディネータ
  - SQLの解析とデータの所在の特定
  - データノード用のSQLを生成、転送
  - アプリケーションとのインタフェース(PostgreSQLと同一)
- データノード
  - 実際のデータを格納
  - コーディネータから来たSQL文の実行

アプリケーション

- Postgres-SQLに見えます
  - テーブル定義時：テーブルを分割するかレプリケーションするかを指定
    - CREATE TABLE X (a int, b ….) DISTRIBUTED BY [HASH] (a);
    - CREATE TABLE Y (c int, d ….) DISTRIBUTED BY REPLICATION;
  - その他のSQL文はPostgreSQLのままで使用できます
    - 分割キーをうまく使うと効率が上がることがあります。
    - 一部未サポートの機能があります。
  - 関数は、コーディネータで動かすか、データノードで動かすかを指定しておく必要があります。
  - 集約関数も新たに定義できます。
    - 複数のデータノードから集めたデータを集計する関数を新たに定義する必要があります。
  - トリガは年度末までに追加サポート予定
    - データノードで動作させることを予定しています。

コーディネータ/データノードの処理例

レプリケーションテーブルの参照



分割テーブルの参照(1)

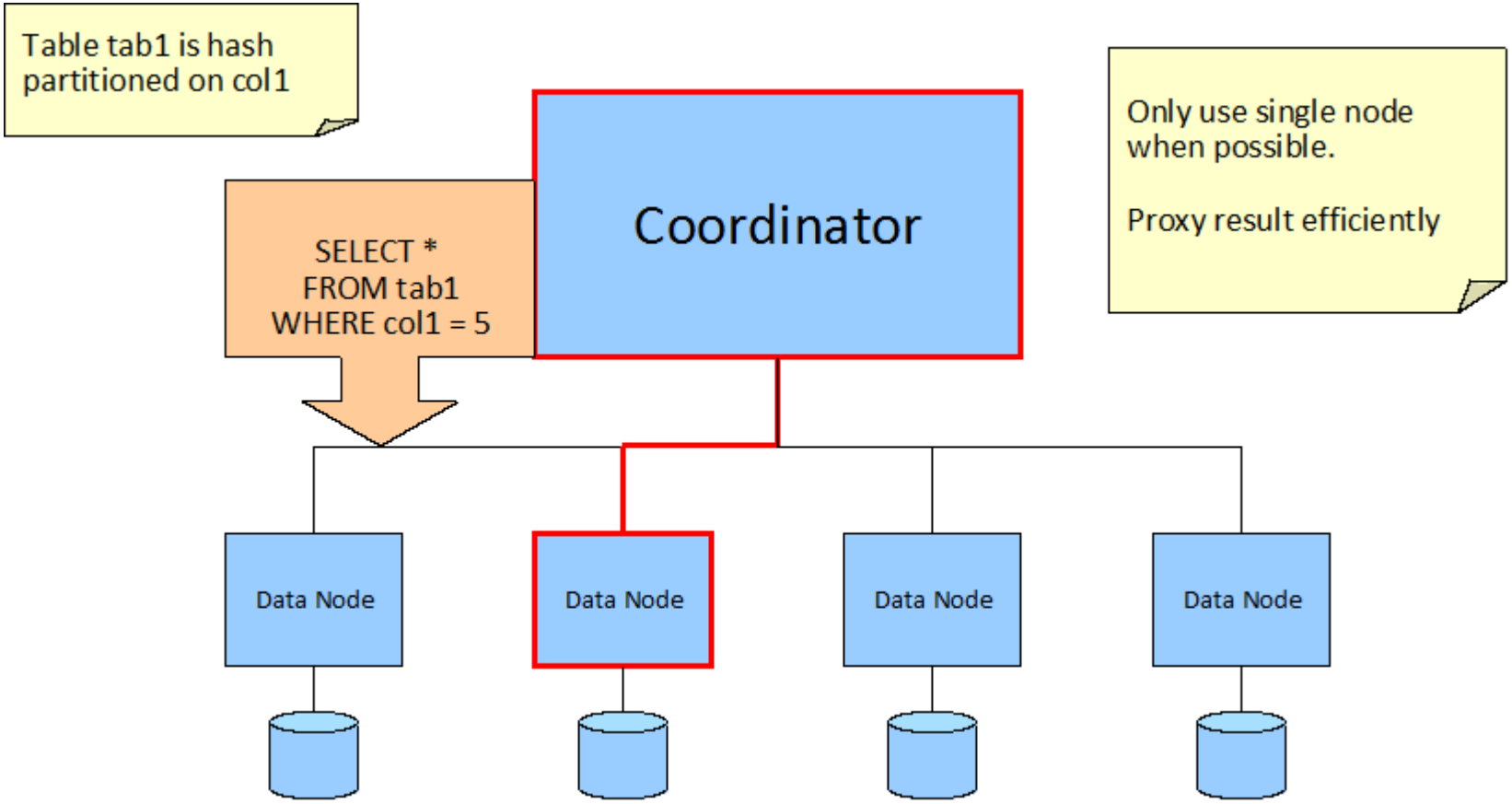


Table tab1 is hash partitioned on col1
SELECT *
FROM tab1
WHERE col1 = 5
Coordinator
Only use single node when possible.
Proxy result efficiently
Data Node
Data Node
Data Node
Data Node

分割テーブルの参照(2)

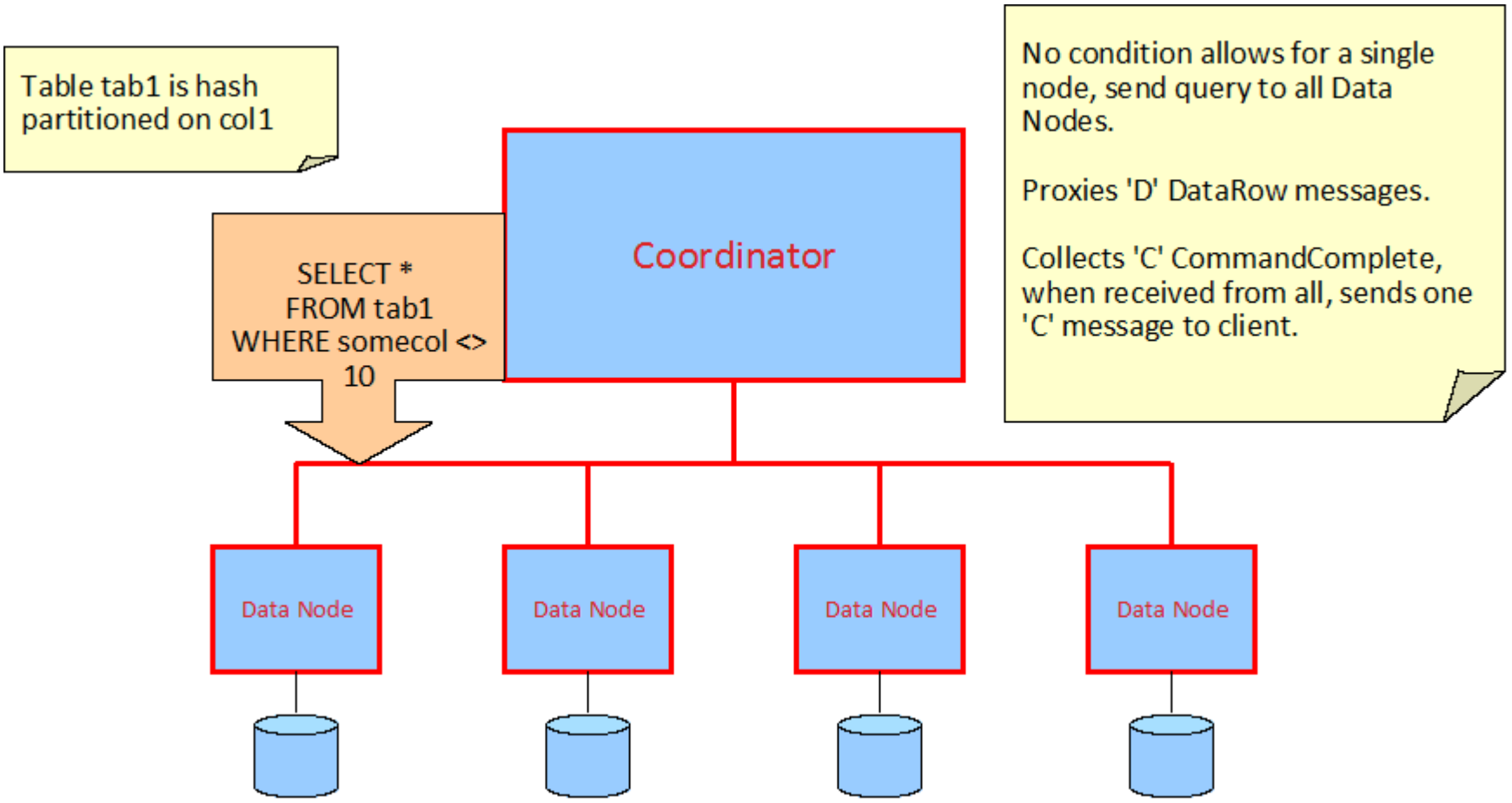
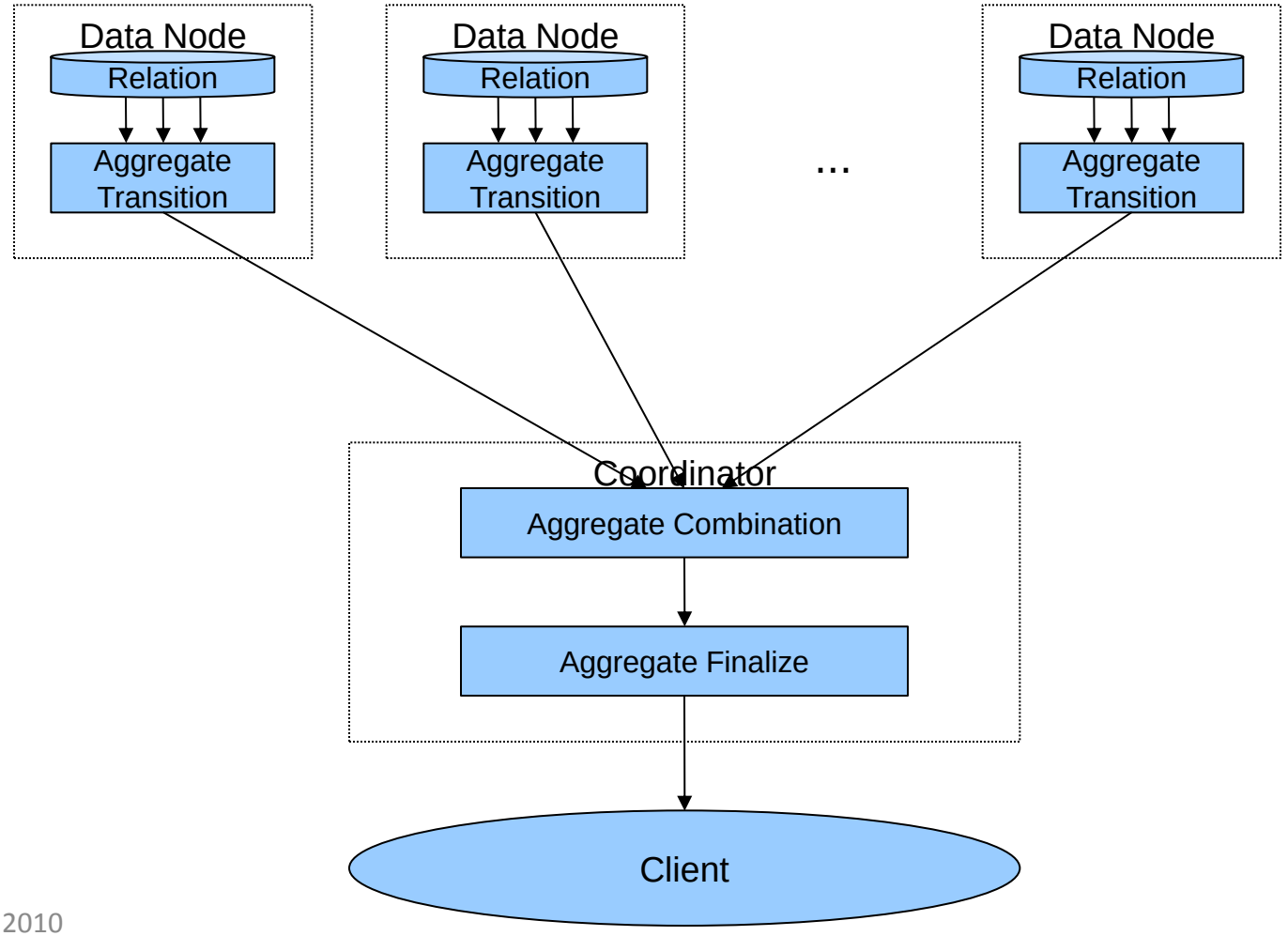


Table tab1 is hash partitioned on col1
SELECT *
FROM tab1
WHERE somecol <> 10
Coordinator
No condition allows for a single node, send query to all Data Nodes.
Proxies 'D' DataRow messages.
Collects 'C' CommandComplete, when received from all, sends one 'C' message to client.
Data Node
Data Node
Data Node
Data Node

集約関数の処理

Postgres-XC Aggregate Flow



Aggregate Handling - AVG

- AVG (Average) needs to sum all elements and divide by the count
- Transition

```
arg1[0]+=arg2;
arg1[1]++;
return arg1;
```

- Combiner (only in Postgres-XC)

```
arg1[0]+=arg2[0];
arg1[1]+=arg2[1];
return arg1;
```

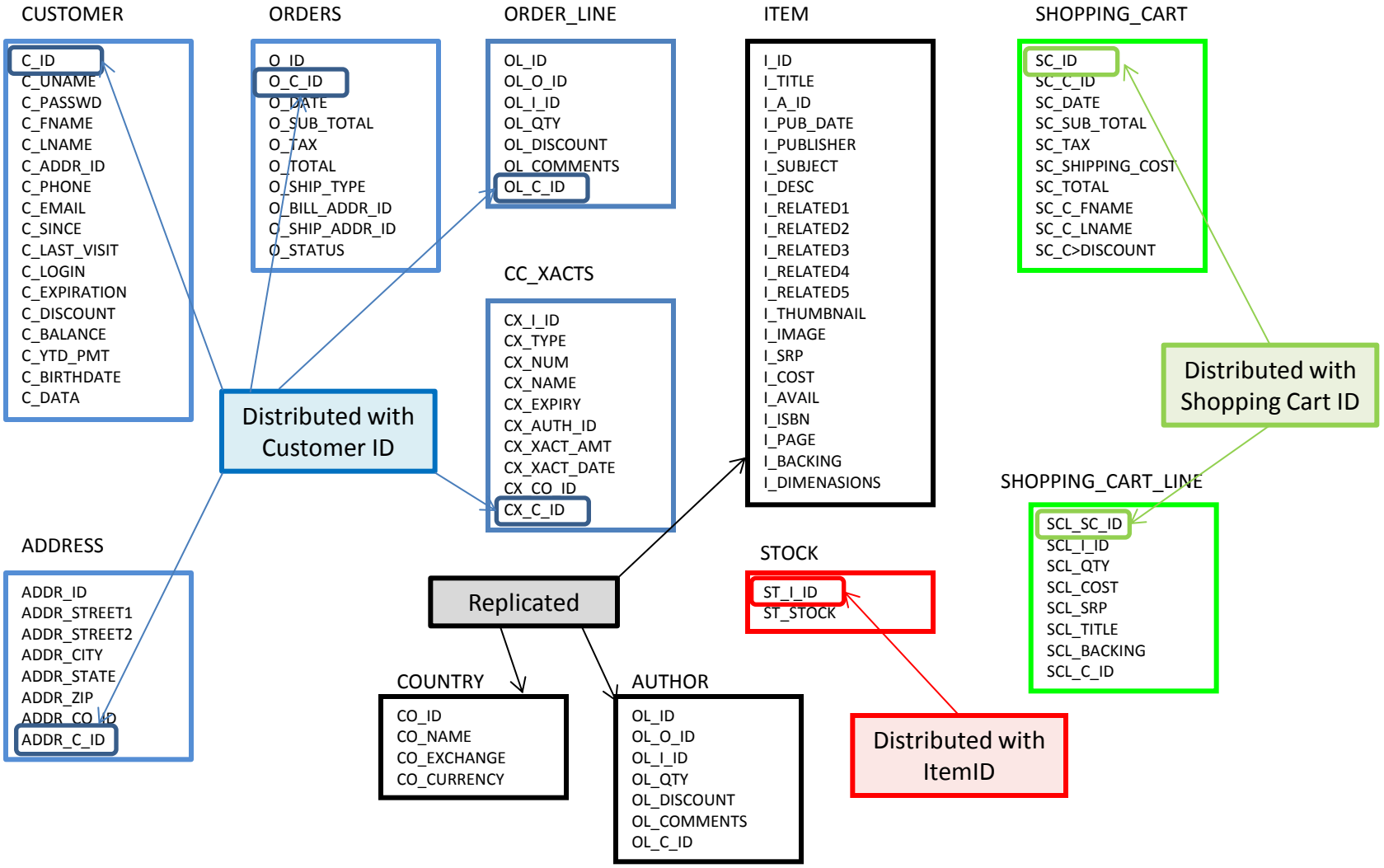
- Finalizer

```
return arg1[0]/arg1[1];
```

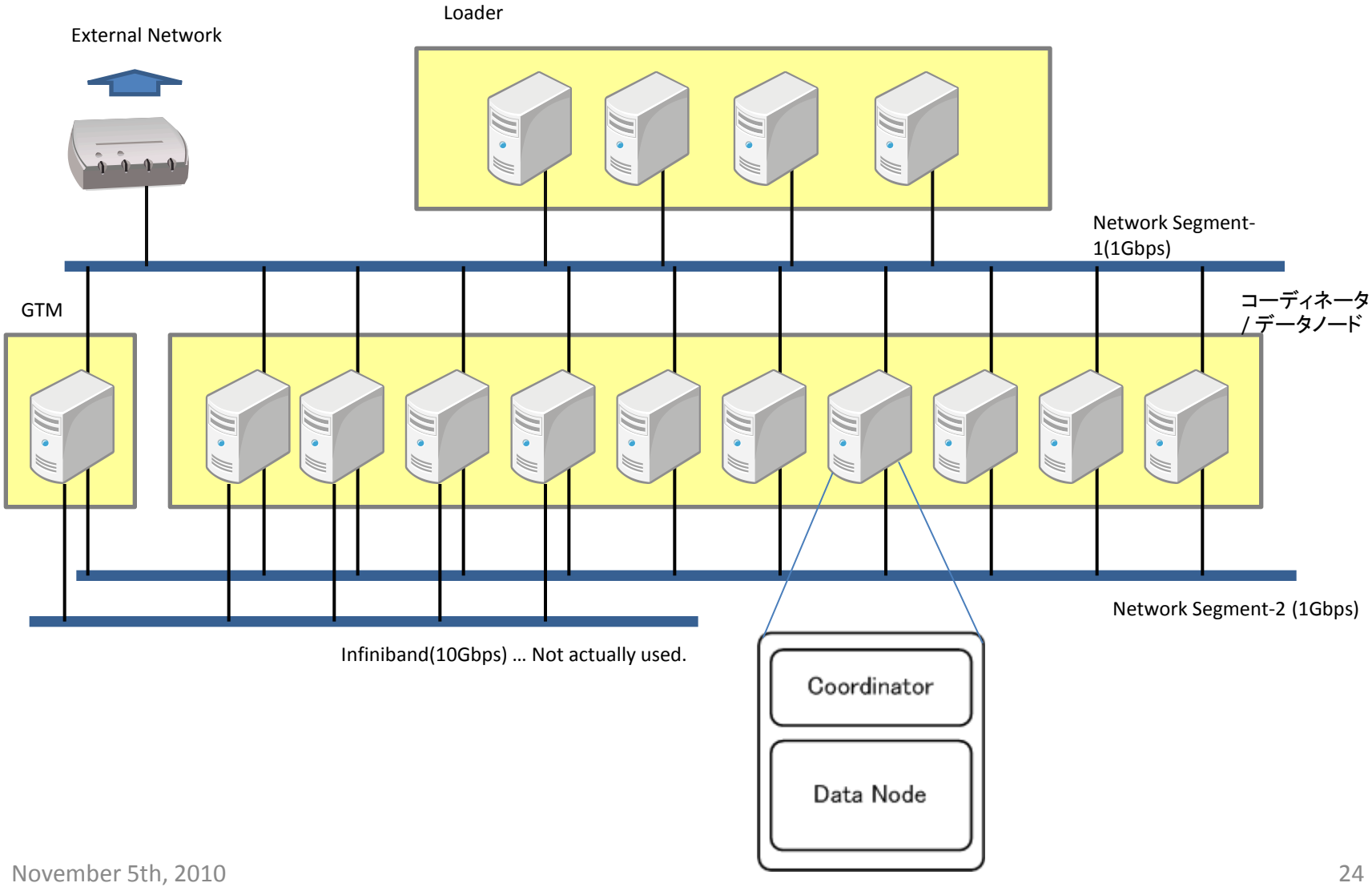
Get the sum of the sums and the sum of the counts from the Data Nodes

検証

DBT-1のテーブル構成



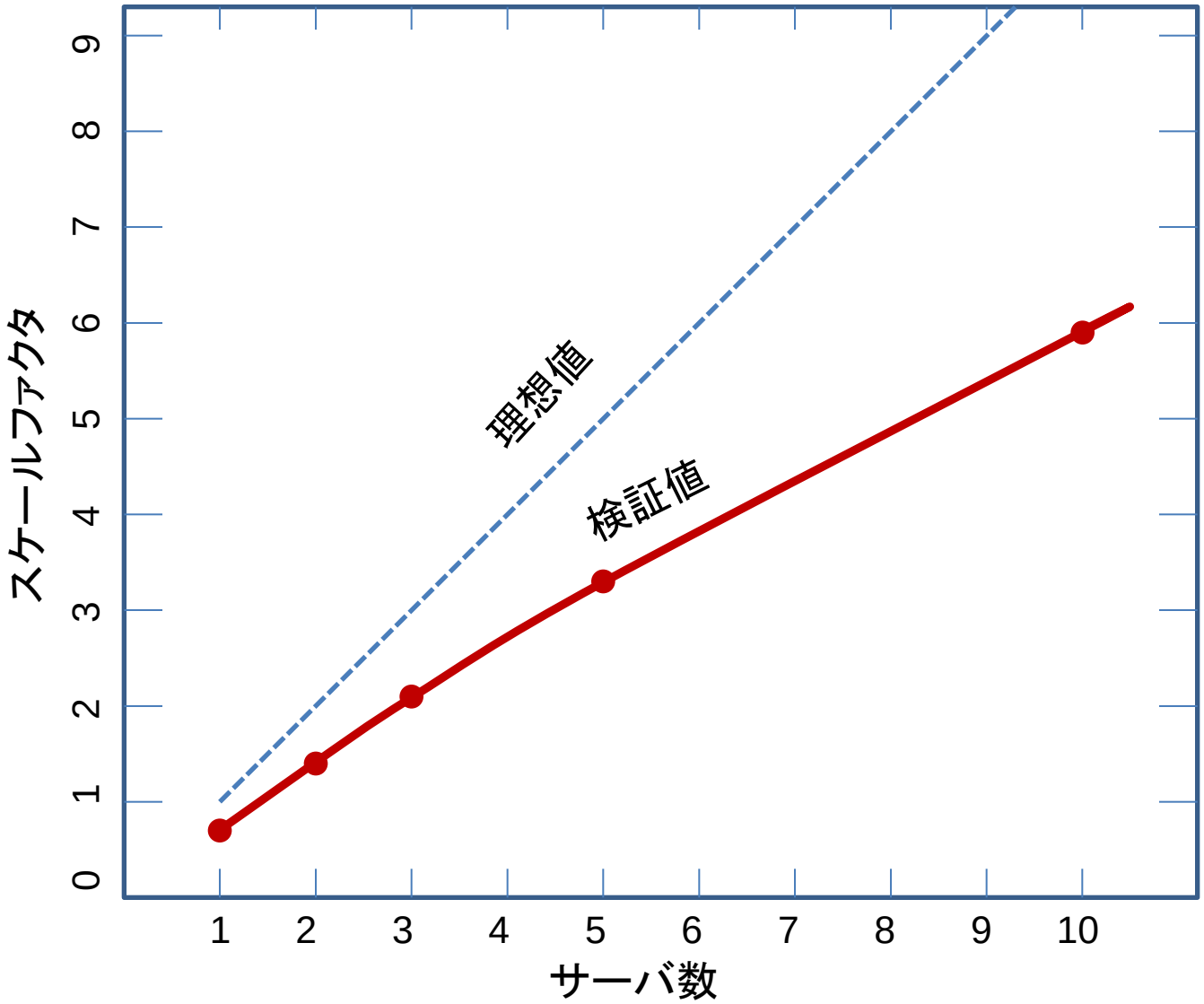
検証環境


External Network
Loader
Network Segment-1(1Gbps)
GTM
コーディネータ / データノード
Network Segment-2 (1Gbps)
Infiniband(10Gbps) … Not actually used.
Coordinator
Data Node

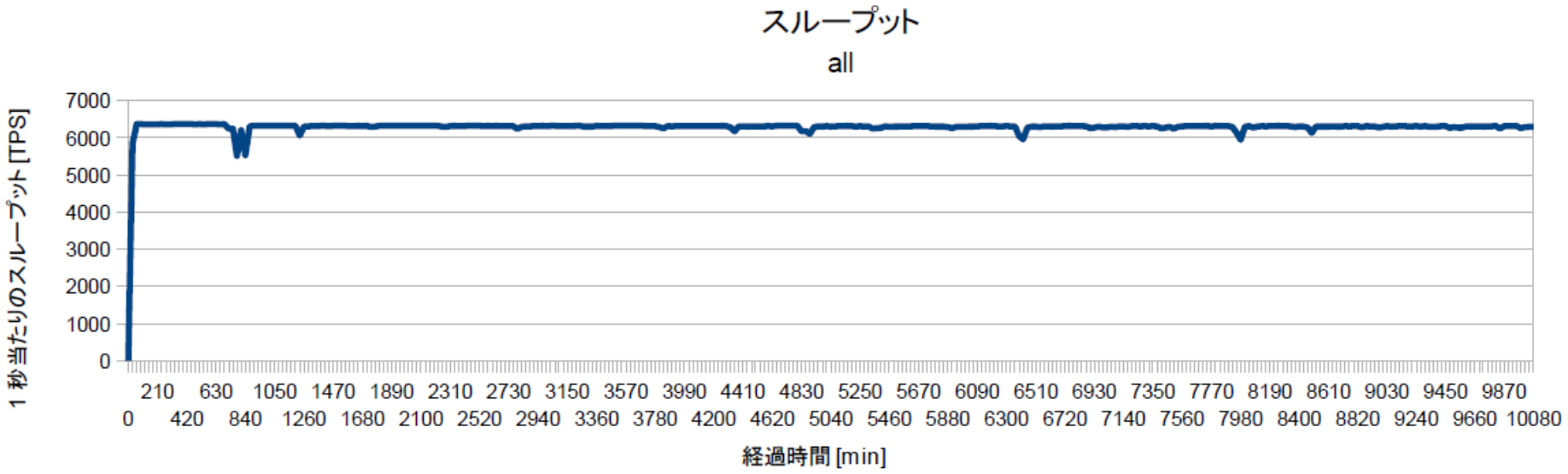
サーバスペック

	コーディネータ/データノード	GTM/Loader
モデル	HP Proliant DL360 G6	HP Proliant DL360 G5
CPU	Intel® Xeon® E5504 2.00GHz x 4	Intel® Xeon® X5460 3.16GHz x 4
キャッシュ	4MB	6MB
メモリ	12GB	6GB
ディスク	146GB SAS 15krpm x 4 ea	146GP SAS 14krpm x 2 ea

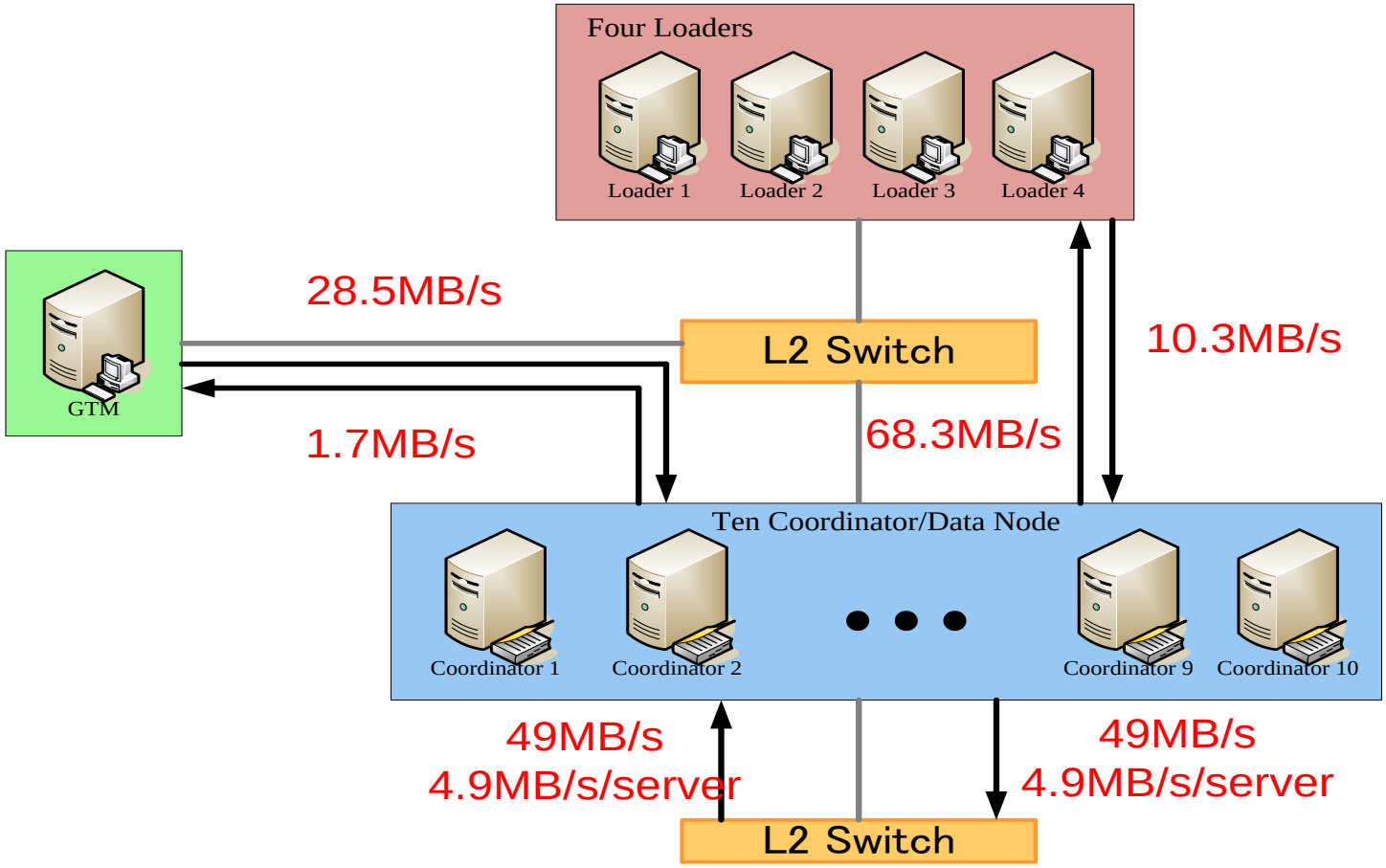
スケールファクタ



一週間連続運転


スループット
all
1 秒当たりのスループット [TPS]
7000
6000
5000
4000
3000
2000
1000
0
0 210 420 630 840 1050 1260 1470 1680 1890 2100 2310 2520 2730 2940 3150 3360 3570 3780 3990 4200 4410 4620 4830 5040 5250 5460 5670 5880 6090 6300 6510 6720 6930 7140 7350 7560 7770 7980 8190 8400 8610 8820 9030 9240 9450 9660 9870 10080
経過時間 [min]

ネットワーク負荷


Four Loaders
Loader 1
Loader 2
Loader 3
Loader 4
GTM
28.5MB/s
1.7MB/s
L2 Switch
68.3MB/s
10.3MB/s
Ten Coordinator/Data Node
Coordinator 1
Coordinator 2
Coordinator 9
Coordinator 10
49MB/s
4.9MB/s/server
49MB/s
4.9MB/s/server
L2 Switch

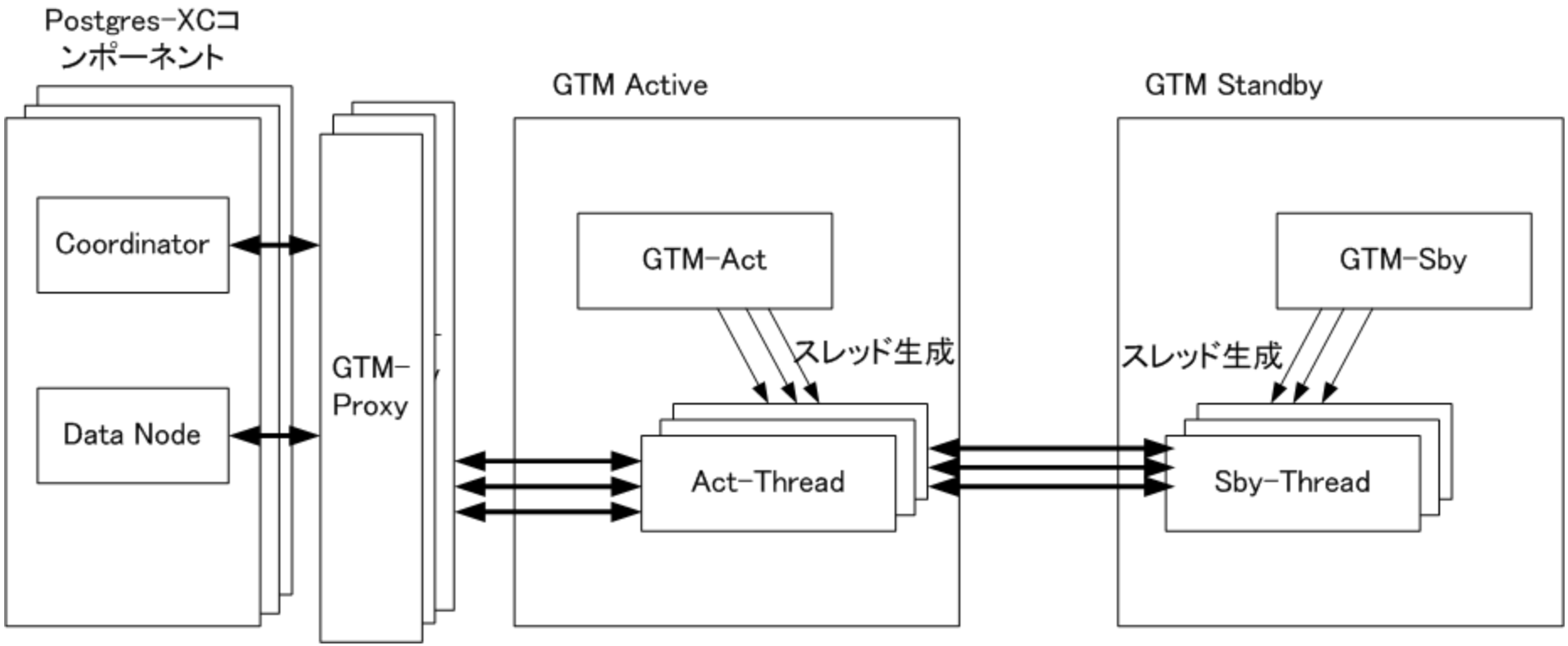
Postgres-XCのHA対応

- 現在開発中

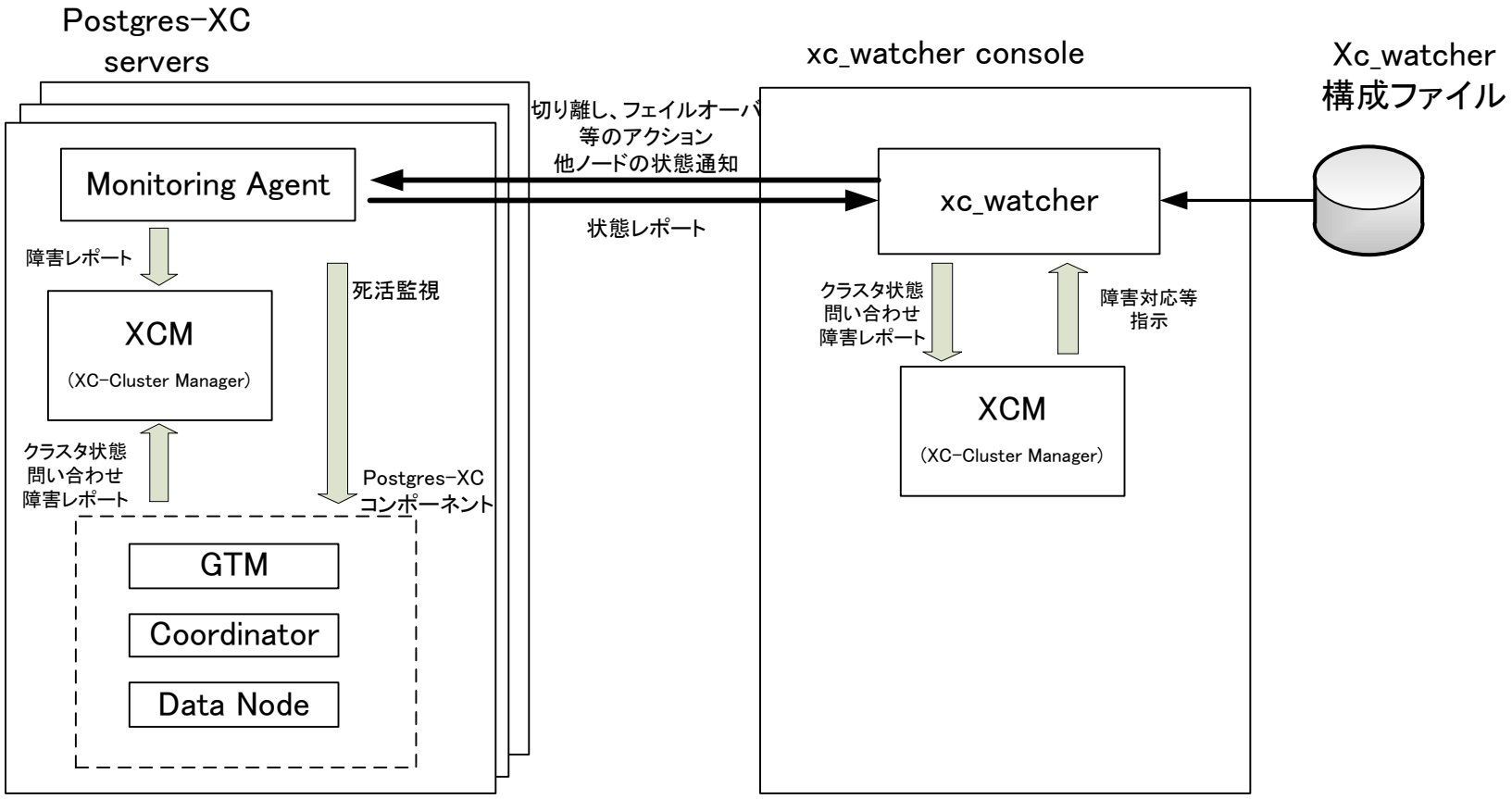
コンポーネント	HA対応機能	障害時の対応	切り戻し
GTM	再起動・リカバリ	他のコンポーネントは停止せず、GTMのみを再起動。実行中のトランザクションはアボート。完了したトランザクションは保証 (PREPAREも含む)。	再起動
	アクティブ Standby による無停止運転	(詳細別途)	
コーディネータ	複数のコーディネータを並列運転	当該コーディネータを切り離し、他のコーディネータで運転継続。当該コーディネータ以外は無停止。コーディネータ上のトランザクションはアボート。	計画停止 →無停止切り戻しは2011年度以降
データノード	ミラーによる無停止運転	当該ミラーを切り離し。データノードとしては無停止。トランザクションロスなし。	

- ハードウェアレベルの監視
  - 汎用のミドルウェアを使用予定

GTM-Standby


Postgres-XCコンポーネント
Coordinator
Data Node
GTM-Proxy
GTM Active
GTM-Act
スレッド生成
Act-Thread
GTM Standby
GTM-Sby
スレッド生成
Sby-Thread

HA監視切り替え方式


Postgres-XC
servers
xc_watcher console
Xc_watcher
構成ファイル
切り離し、フェイルオーバ
等のアクション
他ノードの状態通知
状態レポート
Monitoring Agent
xc_watcher
障害レポート
死活監視
XCM
(XC-Cluster Manager)
クラスタ状態
問い合わせ
障害レポート
障害対応等
指示
クラスタ状態
問い合わせ
障害レポート
Postgres-XC
コンポーネント
XCM
(XC-Cluster Manager)
GTM
Coordinator
Data Node

経緯と今後

- 2009年3月
  - EnterpriseDB社との協業開始
  - GTMのコンセプト検証開始
- 2009年11月
  - 読み書き双方のスケールアウトを検証
  - PostgreSQLクラスタ開発者会議(東京)にて紹介
- 2010年5月 – コンセプト検証レベルでリリース(V0.9)
  - クラスタの全体的な機構はここで完成。
  - PGCon2010で発表。大きな反響。その後、CHAR(10) (Oxford) にて招待講演。
  - PostgreSQL開発者会議にて、クラスタの所要機能を順次本体にも反映させていく方向。
  - PostgreSQLハッカーの間でも、GTMの利用価値の議論がなされている。
- 2010年7月 – DDL/ダンプ・リストア、関数、集約機能などをリリース
  - 国外からプロジェクト参加者が出始める
    - バイナリパッケージの作成
    - デバグ協力
  - ソースコードの提供者も現れる
- 2010年10月 – さらに機能拡充し、SQL文のカバーレンジを大幅に拡張
  - インストール・構成ツール
  - クラスタ全体の運転サポートコマンド
- 2011年3月 – HA機能、JAVA やCアプリケーションに必要な機能を補充
- 2011年4月以降 – さらに可用性、運用性を向上
  - GTMのノンストップ化
  - 単一障害の場合の完全ノンストップ運用、ノンストップ切り戻し

2010年5月以降、開発は急速に加速している。PostgreSQLコミュニティとも密接な関係を保っている。

Postgres-XCメンバ

- NTTオープンソースセンタ/NTTデータ先端技術
  - プロジェクトリーダ
  - 方式設計
  - HA, DDL, 2PC, インストーラその他の実装
  - 検証
- EnterpriseDB
  - 各コンポーネントの実装、細部のアルゴリズム
- 外部の貢献者が増えている
  - Sourceforge を使ったオープンな開発
    - バイナリパッケージの作成
    - 評価
    - コードの貢献

おわりに

- 皆様のご参加をお待ちしております
  - 検証
  - 要件
  - コメント
  - ご意見
  - ソースコード　等々
- ご連絡は次までお願いします。

  koichi@intellilink.co.jp　(技術)
  contact@oss.ntt.co.jp　(一般)
- 詳細情報は下記から取得できます

  http://postgres-xc.sourceforge.net/
  http://sourceforge.net/projects/postgres-xc/
- 紹介記事

  http://thinkit.co.jp/story/2010/10/26/1828

ご清聴ありがとうございました