NTTソフトウェア

PostgreSQLでスケールアウト
-Postgres-XCの紹介-

2013年11月8日

NTT Software Corporation
Tomonari　Katsumata

Agenda

- Postgres-XCとは
  - 開発の歴史
  - アーキテクチャ
  - データ分散処理
  - クエリ処理
- Postgres-XCを使いこなす
  - QueryOptimization
  - HighAvailavility
  - Backups
- デモンストレーション
- ロードマップ
  - 今後の展開
  - 関連情報
- まとめ

自己紹介

勝俣　智成（かつまた　ともなり）
NTTソフトウェア株式会社　主任エンジニア

- 2002年同社入社。
- 数年間は全文検索に関する業務を担当。
- PostgreSQLとの出会いは2004年。
- PostgreSQLに全文検索機能やXML検索機能などを拡張する開発に従事。
- 以降、開発・国内外のPostgreSQLカンファレンスへの参加、社内外でのPostgreSQL研修の講師などを行っている。

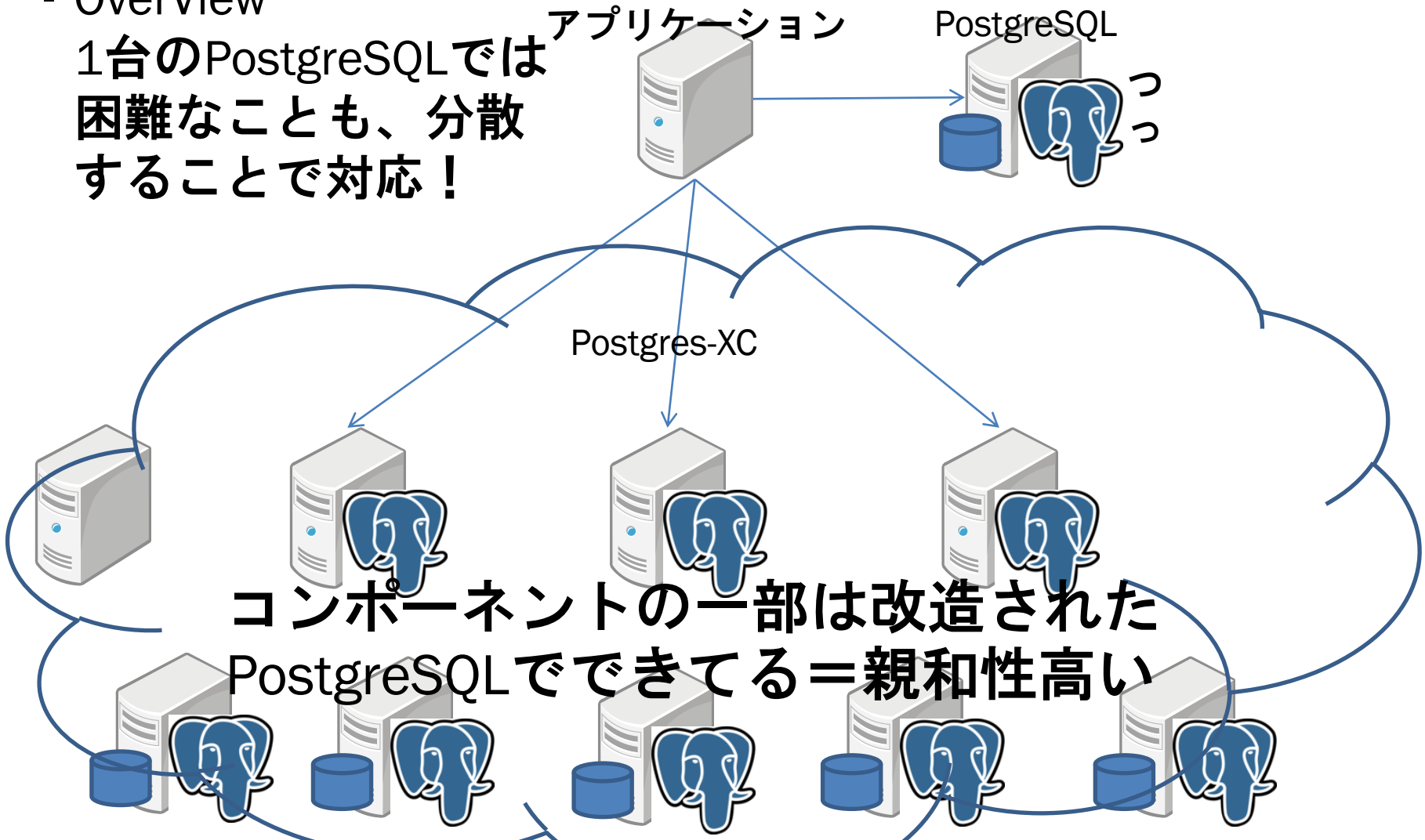
Postgres-XCとは？-開発の歴史-

- 2009年
  - NTT OSSセンタ、EnterpriseDB社によりプロジェクト発足
  - OracleRACの代替としてPostgreSQLベースのクラスタ・ソリューションが目標
  - NTT、EnterpriseDB社がリードする形で、コミュニティベースでの開発を開始
- 2011年
  - ライセンスをPostgreSQLと同じPostgreSQLライセンスに変更
- 2012年
  - バージョン1.0をリリース！
- ～現在
  - コミュニティベースで開発継続中(最新版は1.1)。

-http://sourceforge.net/apps/mediawiki/postgres-xc/index.php?title=Main_Page

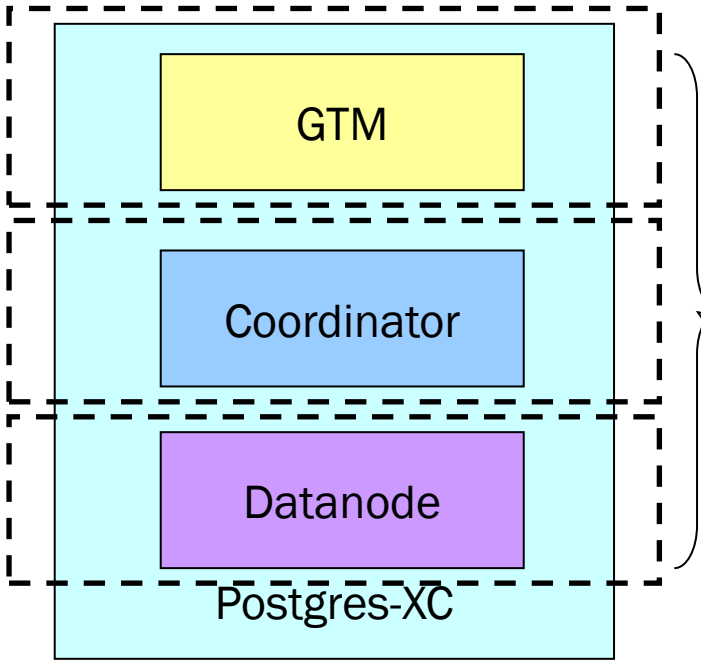
Postgres-XCとは-アーキテクチャ-（1/5）

- OverView
  1台のPostgreSQLでは
  困難なことも、分散
  することで対応！



Postgres-XCとは-アーキテクチャ-(1/5)

- Components

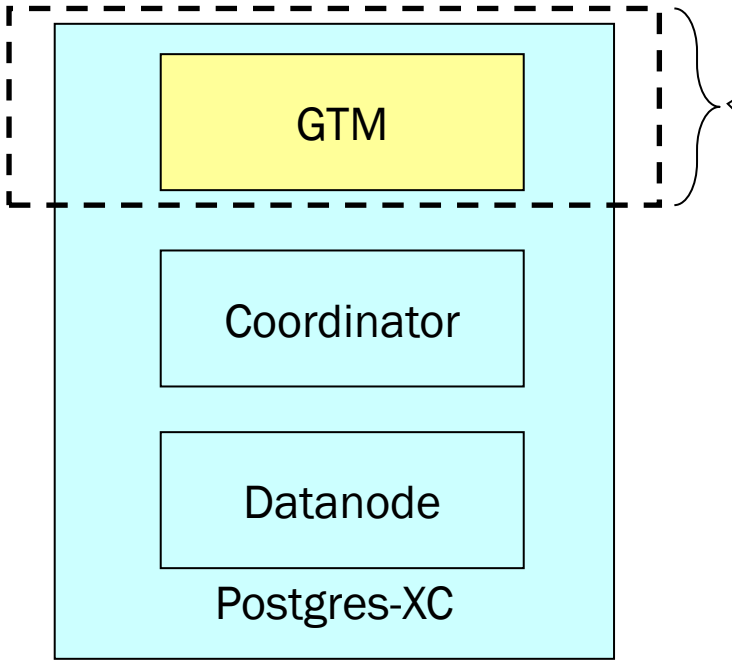


PostgreSQLの「トランザクション管理部」、「SQL処理部」、「データ処理部」をコンポーネントとして切り出し、別々のサーバ上に配置可能としている。
GTMが「トランザクション管理部」、Coordinatorが「SQL処理部」、Datanodeが「データ処理部」に相当する。

※Coordinator、Datanodeは改造されたPostgreSQL。
GTMはPostgres-XC用に新規開発されたもの。

Postgres-XCとは-アーキテクチャー(2/5)

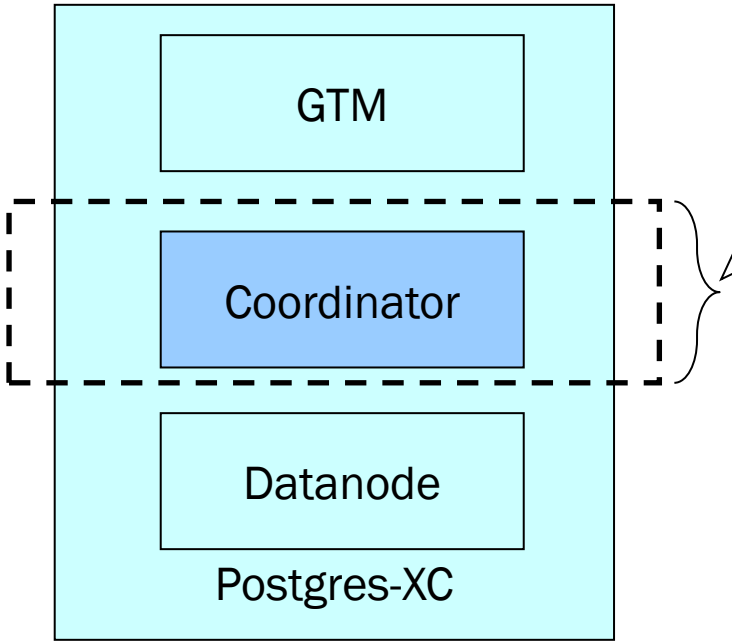
- GTM



GTM：
　Postgres-XC全体を通してのトランザクションID(GXID)の払い出しなどを司るコンポーネント。
　そのほか可視性担保のためのSnapshotやPostgres-XC全体で管理するシーケンスやタイムスタンプ値を払い出す。

Postgres-XCとは-アーキテクチャー(3/5)

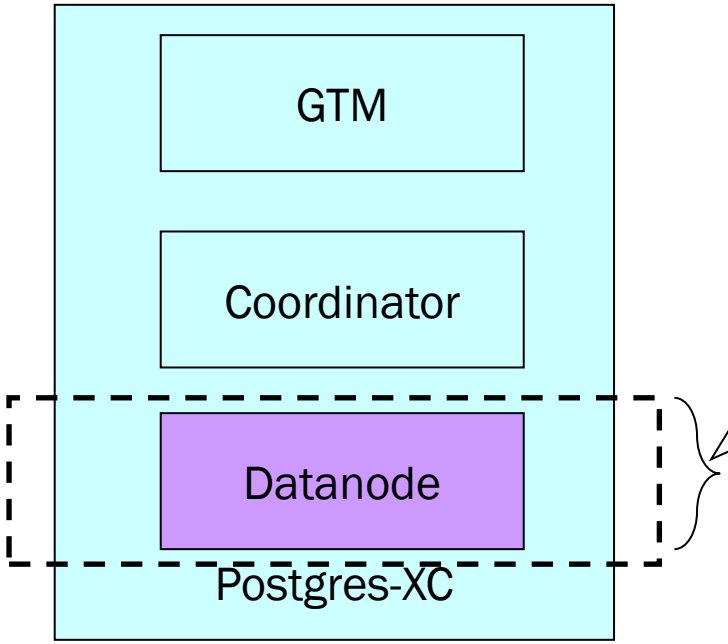
- Coordinator



Coordinator：
　AP/クライアントの窓口を担うコンポーネント。
　SQLの構文解析を行い、どのノードに振り分けるかの決定する(2PC管理も行う)。
　取得したデータは必要に応じてマージし、AP/クライアントへ返却する。
　複数台配置することで処理の分散を図ることができる。

Postgres-XCとは-アーキテクチャー(4/5)

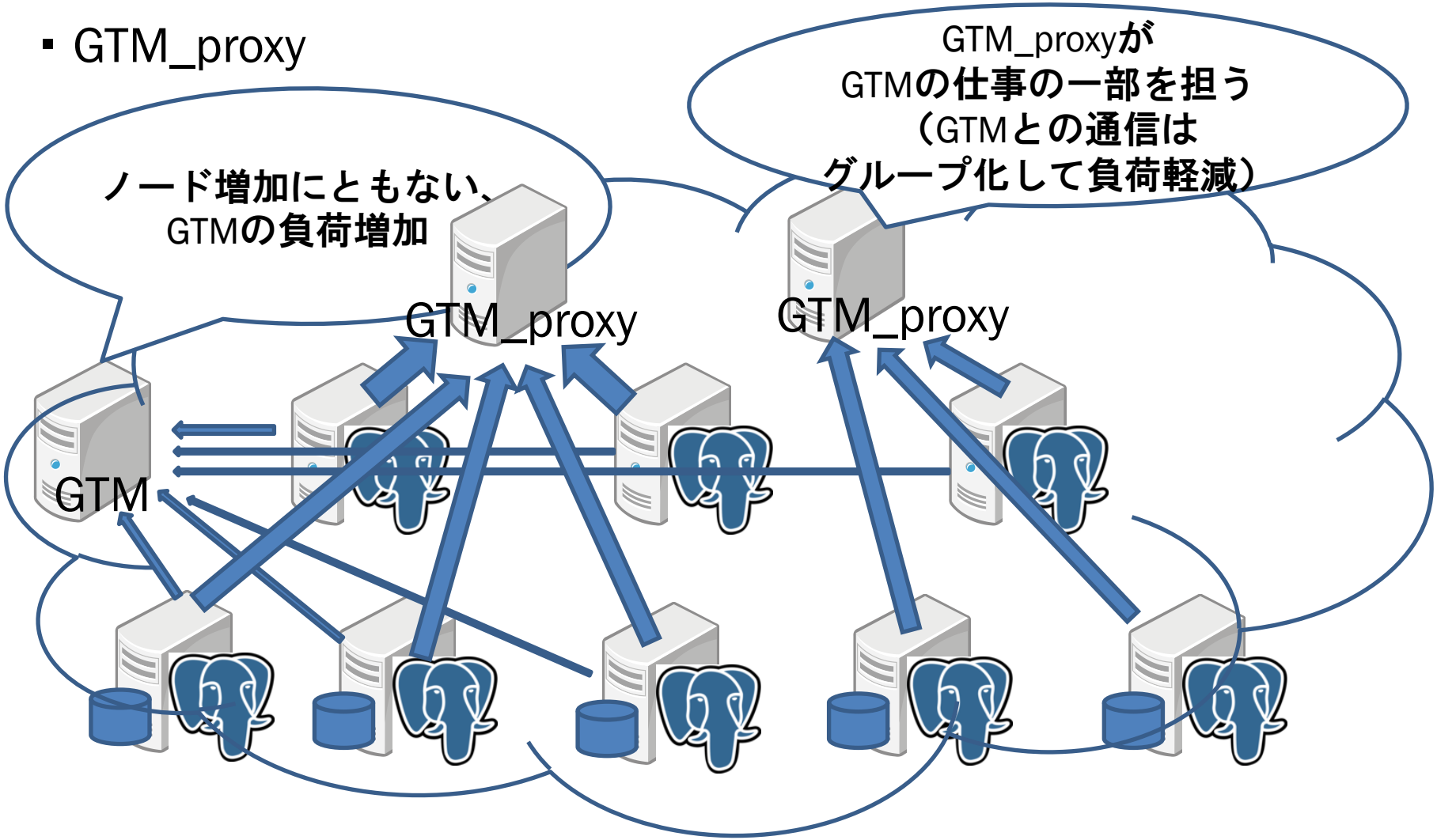
- Datanode



Datanode：
　データの格納先となるコンポーネント。
　Coordinatorからの問い合わせを実行し、結果を返す。
　複数台配置することで、データの分散配置ならびに負荷分散を図ることができる。

Postgres-XCとは-アーキテクチャー(5/5)

- GTM_proxy



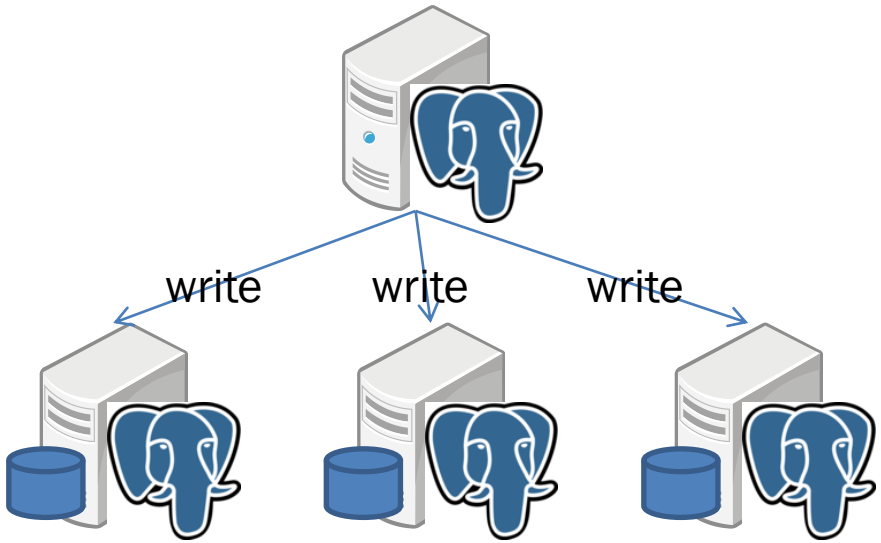
Postgres-XCとは-データ分散処理-(1/3)

Datanodeでの**データ格納方式**は、大きく分けて「レプリケーションテーブル」と「分散テーブル」がある。分散テーブルの分割方法を含めて4つの方式をとれる。

- レプリケーションテーブル
  - レプリケーション
- 分散テーブル
  - ハッシュ分割
  - モジュロ分割
  - ラウンドロビン分割

Postgres-XCとは-データ分散処理-(2/3)

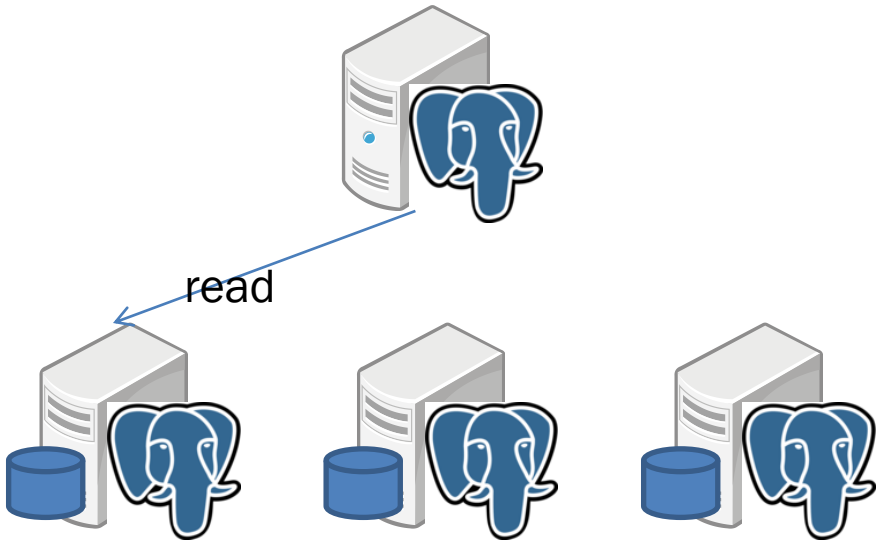
・レプリケーションテーブル
  ・同一データが格納される



col1	col2
1	45
2	23
3	34

col1	col2
1	45
2	23
3	34

col1	col2
1	45
2	23
3	34



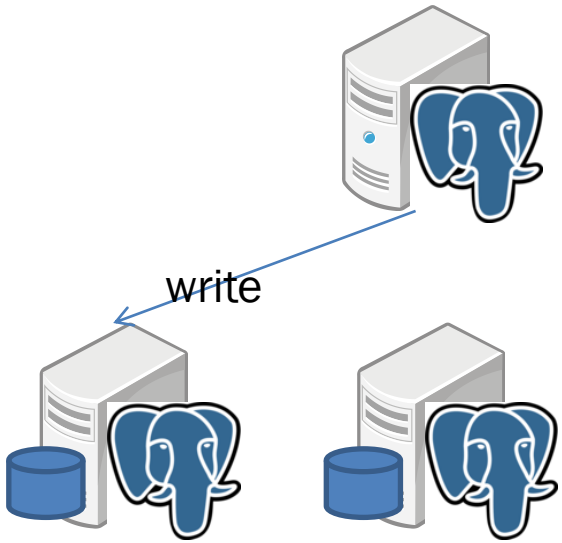
col1	col2
1	45
2	23
3	34

col1	col2
1	45
2	23
3	34

col1	col2
1	45
2	23
3	34

Postgres-XCとは-データ分散処理-(3/3)

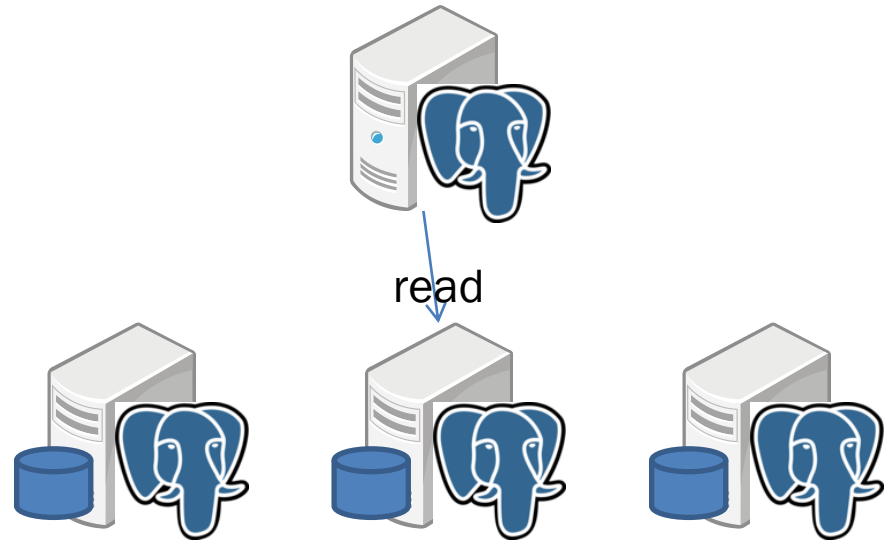
- 分散テーブル
  - 各ノードにデータが分散して格納される



col1	col2
1	45
2	23
3	34

col1	col2
101	654
201	8
301	124

col1	col2
102	26
202	96
302	6



col1	col2
1	45
2	23
3	34

col1	col2
101	654
201	8
301	124

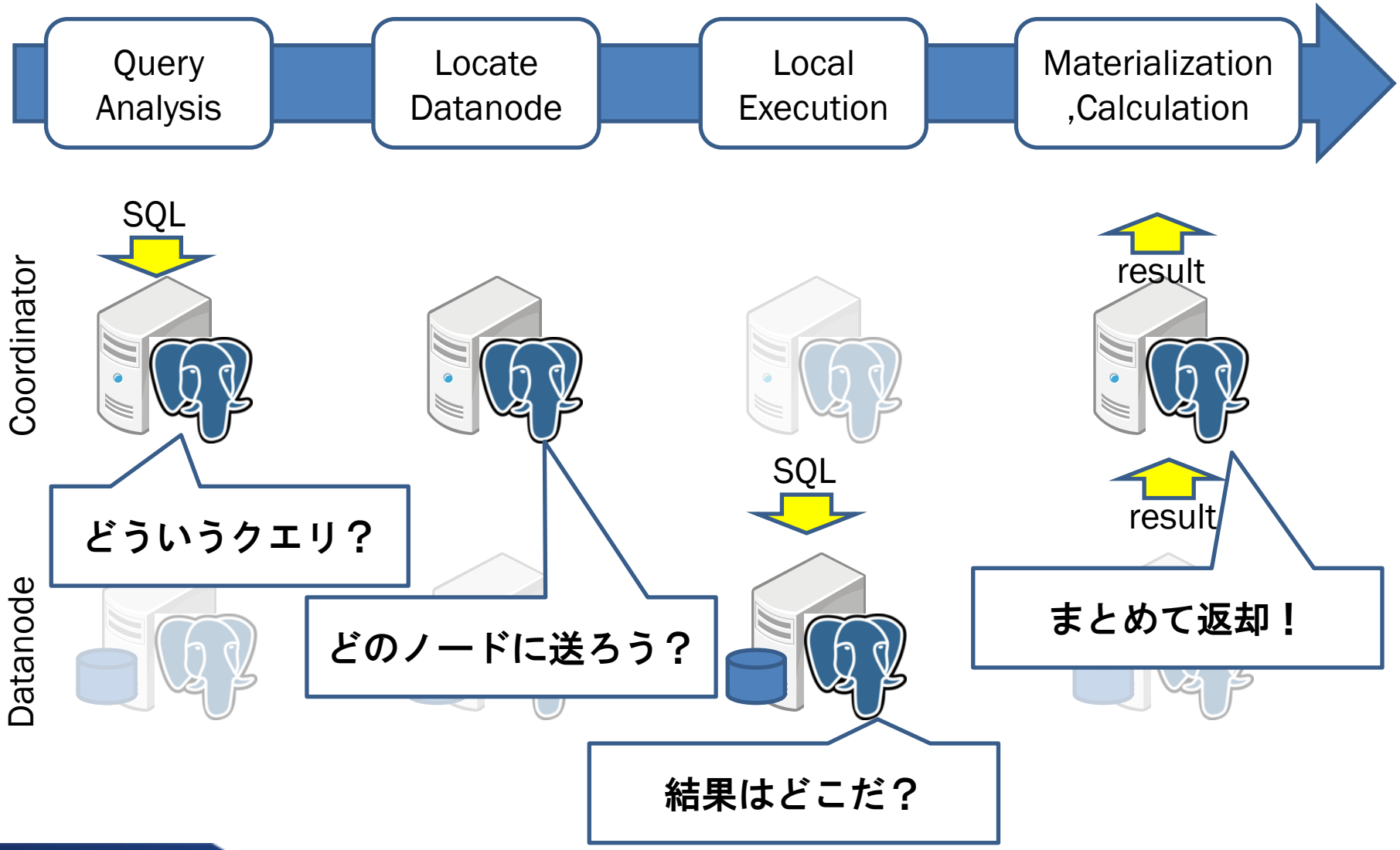
col1	col2
102	26
202	96
302	6

Postgres-XCとは-クエリ処理-(1/2)

PostgreSQLのクエリ処理(パース、リライト、プランニング、エグゼキュート)に加えて、主にCoordinaotrで以下の処理も行う。

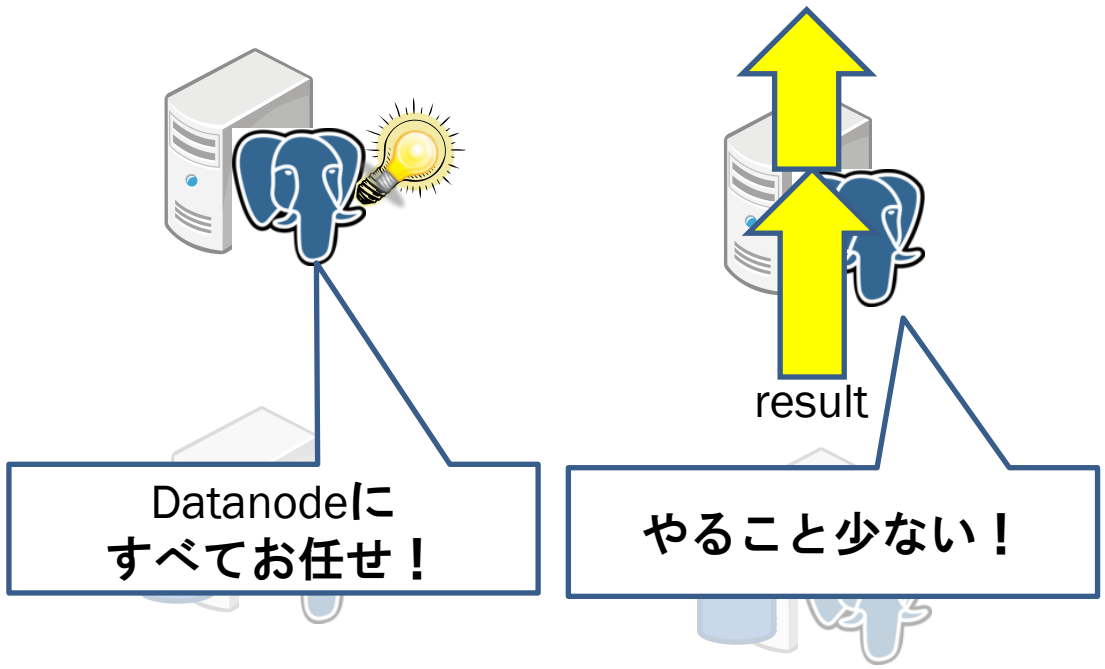
- Query Analysis
  →元のSQLの解析
- Locate Datanode
  →実行すべきDatanodeの特定
- Local Execution
  →Datanodeで(必要な)SQL実行
- Materialization, Calculation
  →結果の集計

Postgres-XCとは-クエリ処理-(2/2)



XCを使いこなす-QueryOptimization-（1/3）

「LocateDatanode」でよいプランニングができれば
DatanodeでJoin等を含めて処理できるので効率的！
これをFQS（FastQueryShipping）という。



どのようなデータが格納されるのか、どのようなクエリパターンでアクセスするのかということから、FQSが実行できるようにテーブルを分散させると性能向上を期待できる

XCを使いこなす-QueryOptimization-（2/3）

FQSが有効な例：
レプリケーションテーブル同士のJOIN

```
=# create table tab1(val int, val2 int)
          distribute by replication to node (datanode1, datanode2);
=# create table tab2(val int, val2 int)
          distribute by replication to node (datanode1, datanode2);
=# insert into tab1 values (generate_series(1,10), generate_series(1,10));
=# insert into tab2 values (generate_series(1,10), generate_series(1,10));
=# explain verbose select * from tab1,tab2 where tab1.val = tab2.val2;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Output: tab1.val, tab1.val2, tab2.val, tab2.val2
   Node/s: datanode1
   Remote query: SELECT tab1.val, tab1.val2, tab2.val, tab2.val2 FROM public.tab
1, public.tab2 WHERE (tab1.val = tab2.val2)
(4 rows)
```

XCを使いこなす-QueryOptimization-（3/3）

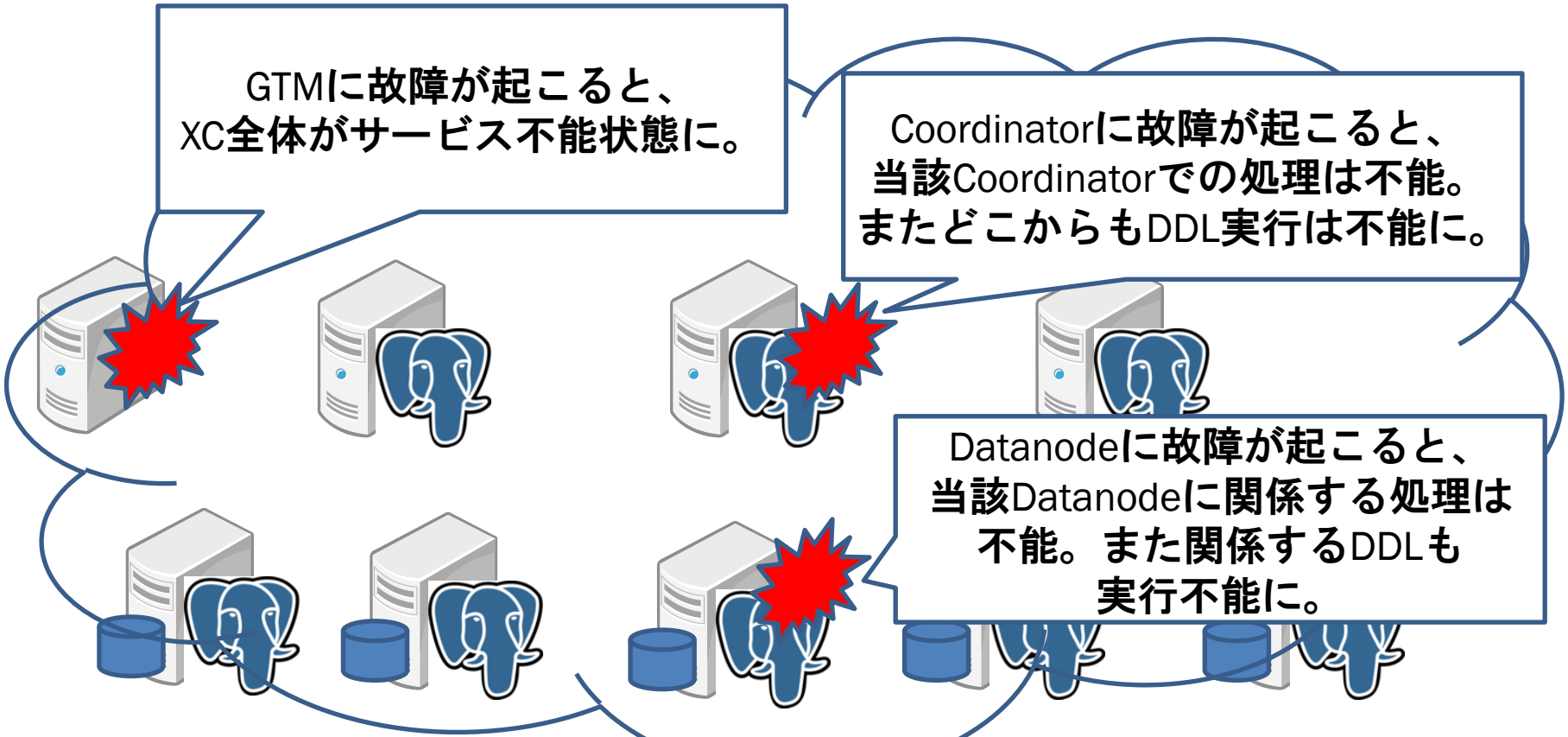
FQSが有効になるJOINの条件

	Hash/Module 分散テーブル	RoundRobin テーブル	Replicated
Hash/Module 分散テーブル	分散keyとなる列のINNER JOIN (同じデータ型、分散方式であること)	×	Replicationテーブルが分散テーブルの振り分けと互換があるINNER JOIN(*)
RoundRobin テーブル	×	×	Replicationテーブルが分散テーブルの振り分けと互換があるINNER JOIN(*)
Replicated	Replicationテーブルが分散テーブルの振り分けと互換があるINNER JOIN(*)	Replicationテーブルが分散テーブルの振り分けと互換があるINNER JOIN(*)	どんなJOINでも可能

(*)Where句で分散テーブルに対する条件がある場合など

XCを使いこなす-HighAvailavility-（1/3）

Postgres-XCにはたくさんのコンポーネントが存在しているため、どのコンポーネントが故障したらどうなるかというのを押さえておくことが重要。



XCを使いこなす-HighAvailavility-（2/3）

- Coordinator/Datanode
  - **これらのコンポーネントが停止すると一部のDML/DDLが実行不能に。**
  - **これらのコンポーネントは、PostgreSQLがベースとなっているため、Synchronous Replicationをそのまま使える。**

必要な設定：

postgresql.conf(on Master)

- wal_level ・・・hot_standby
- archive_mode ・・・on
- archive_command ・・・**適切にアーカイブする**

recovery.conf(on Standby)

- restore_command ・・・**適切にリストアする**
- standby_mode ・・・on
- primarry_conninfo ・・・**マスタの情報**

XCを使いこなす-HighAvailavility-（3/3）

- GTM
  - GTMが故障するとXC全体が停止してしまう。
  - GTMは現時点(v1.1)ではトランザクションログを記録する仕組みがないが、GTM_standbyを用意して組み込むことができる(ただし1台だけ)。

必要な設定：

gtm.conf(on Standby)

- startup ・・・STANDBY
- active_host ・・・GTM(Master)のホスト名
- active_port ・・・GTM(Master)のポート番号

上記パラメータは、起動時オプションとして付与することも可能。

```
$ gtm_ctl start  -Z gtm -o '-s -i actserver -q 6666'
```

また、gtm_ctl**コマンドで**promote**することも可能。**promote**した際には、**Coordinator/Datanode**の再起動**(gtm_proxy**なし**)もしくは**GTM_proxy**の再接続**(gtm_proxy**あり**)が必要**

XCを使いこなす-Backups-(1/3)

たくさんのコンポーネント間で整合性を保ちながらバックアップを取得するにはどうすればよいか？

- 普通のオンラインバックアップではNG

  (各コンポーネントのリカバリポイントがバラバラなので)
- 整合性を保つためには各トランザクションのステータスは以下いずれかでなければならない

  全ノードCommitted/Prepared/Aborted/Running

  (部分的にCommitted/PreparedやRollback/Preparedなのはダメ)

→整合性のあるポイントをつくるのが、、、

BARRIER！

XCを使いこなす-Backups-(2/3)

- BARRIER
  - 整合性のあるポイントですべてのCoordinator、DatanodeのWALにレコード(barrier)を挿入する
    - 2PCで部分的にCommitted/Abortedな状態であれば待つ
    - BARRIER処理実行中は全てのトランザクションのCOMMITも待たされる

    (timeoutがないので要注意)
  - BARRIER処理のコマンド：
    - CREATE BARRIER 'barrier-id'; -- barrier-idは任意の文字列
- GTMは？
  - v1.1からBARRIER処理中にgtmのデータを複製
    - 要gtm_backup_barrierの設定(SET文で可)

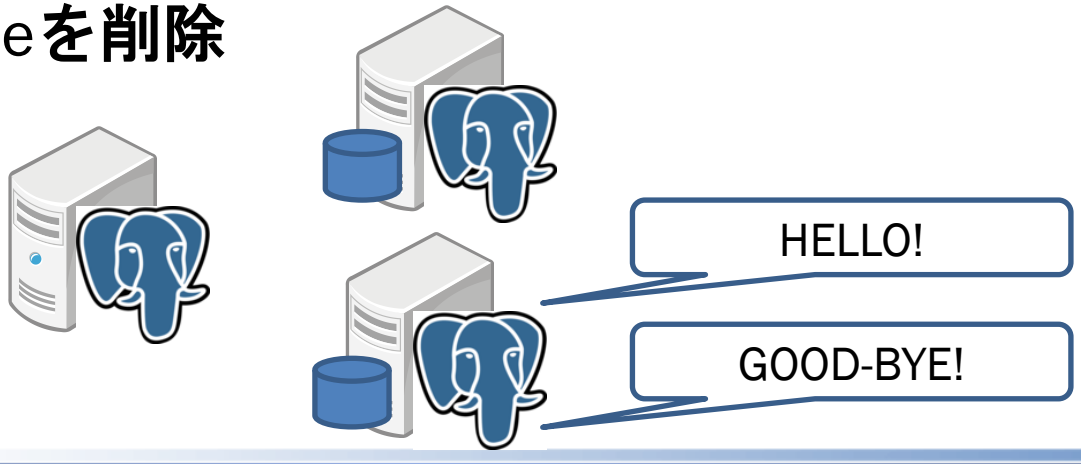
XCを使いこなす-Backups-(3/3)

- リカバリ
  - GTM
    BARRIER中に生成されたgtmのデータを再配置
  - Coordinator/Datanode
    recovery.confで”recovery_target_barrier”の指定を追加。値は”CREATE BARRIER”で指定した文字列

あとはそれぞれのコンポーネントを起動するだけ

デモンストレーション

- Postgres-XC v1.1から備わったpgxc_ctl(contrib)を用いて「ノード追加・削除」のデモンストレーションを行う
- シナリオ
  - ノード追加
    - 1Coordinator/1Datanodeにデータあり(分散テーブル)
    - もう1つDatanodeを追加
    - 追加したノードを含めて再配置
  - ノード削除
    - 追加したノードにデータを配置しないようにする
    - Datanodeを削除



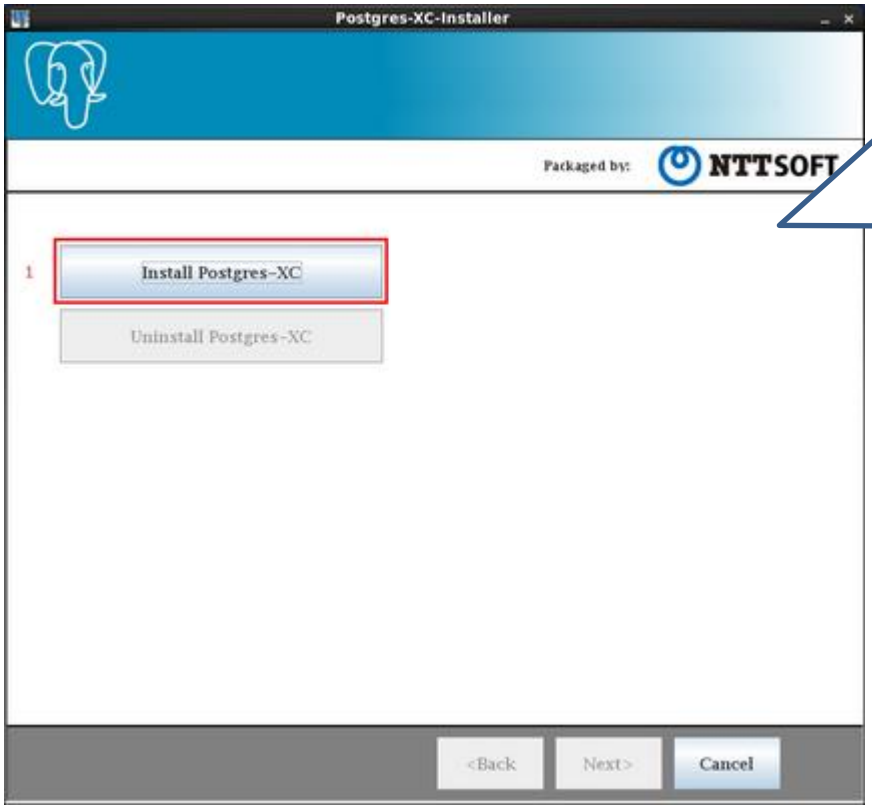
ロードマップ–今後の展開–

- V1.2(2013/12予定)
  - JDBC
  - **プランナの改善**
  - PostgreSQL9.3**ベースへの変更** etc.

- V1.3**以降**
  - **テーブル分散の同時実行性向上**
  - **柔軟なプッシュダウン向け関数**
  - Coordinator/Datanode**の統合**
  - 1**台構成の**XC**対応**
  - **セーブポイント対応**
  - Repeatable READ/SSI**対応**
  - Coordinator**へのGTM_proxy取り込み**
  - **最新版**PostgreSQL**ベースへの変更** and more.

ロードマップ–関連情報–

- インストールを少しでも簡単にするために
  - GUI Installer公開中！
    https://github.com/xctools/postgres-xc-installer



作成できる環境は固定的(＊)
だが、少し触るだけなら
GUIで比較的容易に
インストール可能！

(＊)GTM:1台
GTM_proxy/Coordinator/Datanode:N台

まとめ

本資料では、下記の点について説明した。

- 「Postgres-XCとは」と題して、”開発の歴史”、“アーキテクチャ”、“データ分散処理”、“クエリ処理”の概要を説明した
- 「Postgres-XCを使いこなす」と題して、“QueryOptimization”、“HighAvailavility”、“Backups”の概要を説明した
- 「ロードマップ」にて”今後の展開”と“関連情報”の説明を行い、最後に「デモンストレーション」として”ノード追加・削除”の例を示した

参考

本家サイト
http://sourceforge.net/apps/mediawiki/postgres-xc/index.php?title=Main_Page

本家メーリングリスト(主なもの)
Bugs
https://lists.sourceforge.net/lists/listinfo/postgres-xc-bugs

Developers
https://lists.sourceforge.net/lists/listinfo/postgres-xc-developers

General
https://lists.sourceforge.net/lists/listinfo/postgres-xc-general

Enjoy!