

# Citusを使ってPostgreSQLを スケールアウトしてみよう

Noriyoshi Shinoda

November 22, 2018



# 自己紹介

篠田典良(しのだのりよし)



## ✓所属

- 日本ヒューレット・パカード株式会社 Pointnext 事業統括

## ✓現在の業務など

- PostgreSQLをはじめ Oracle Database , Microsoft SQL Server, Vertica, Sybase ASE 等 RDBMS 全般に関するシステムの設計、チューニング、コンサルティング
- Oracle ACE
- Oracle Database 関連書籍 15 冊の執筆
- オープンソース製品に関する調査、検証

## ✓関連する URL

- 「PostgreSQL 虎の巻」シリーズ

<http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802>

- Oracle ACE ってどんな人？

<http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html>

---

# Agenda

- ✓ Citus とは？
- ✓ 試してみよう
- ✓ アーキテクチャー
- ✓ 制約
- ✓ トラブル発生時の動作

本資料は **Citus Community Edition 8.0-8** を元に作成しています





# Citus とは？

---

# Citus とは？

## Citus とは？

- ✓ PostgreSQL でスケールアウト環境を実現
  - 複数ノードにまたがったパラレル・クエリーとパーティショニング機能
- ✓ PostgreSQL の拡張 (EXTENSION) として実装
- ✓ Citusdata 社が開発
  - Community Edition はオープンソース
  - オンライン・リバランス等の機能を使うためには Enterprise Edition が必要
- ✓ 以下の機能は含まない
  - 自動フェイルオーバー
  - 自動データ・リバランス
  - バックアップ等の運用機能



# Citus とは？

## インスタンス構成

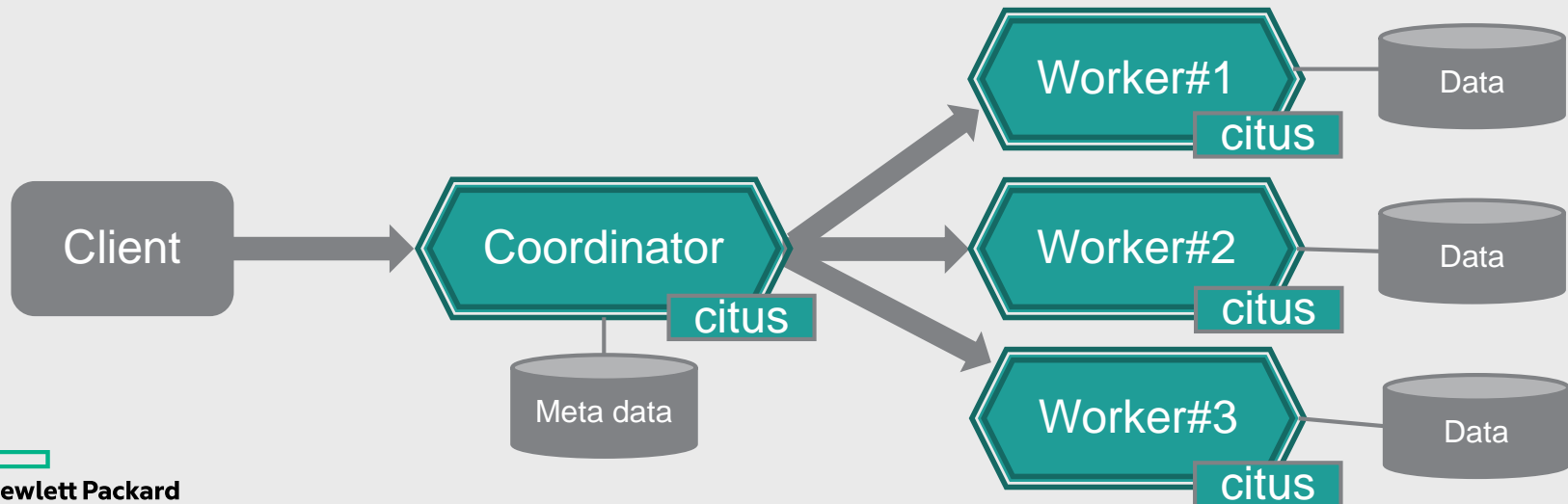
### ✓Coordinator Node

- クライアントからの接続を受け付ける PostgreSQL インスタンス
- メタ・データを管理

### ✓Worker Node

- 実際にデータを保存する PostgreSQL インスタンス
- Worker Node 間には通信を行わない

### ✓すべてのノードに citus EXTENSION をインストール



---

# Citus とは？

## citus EXTENSION

### ✓インストール先

- Coordinator Node と、Worker Node にインストール
- テーブルを作成する全データベースで CREATE EXTENSION 文を実行
- インストール・バイナリは全ノードで同一
- アプリケーションに必要なエクステンション (pgcrypto など) も全ノードにインストール



# Citus とは？

## テーブル構成

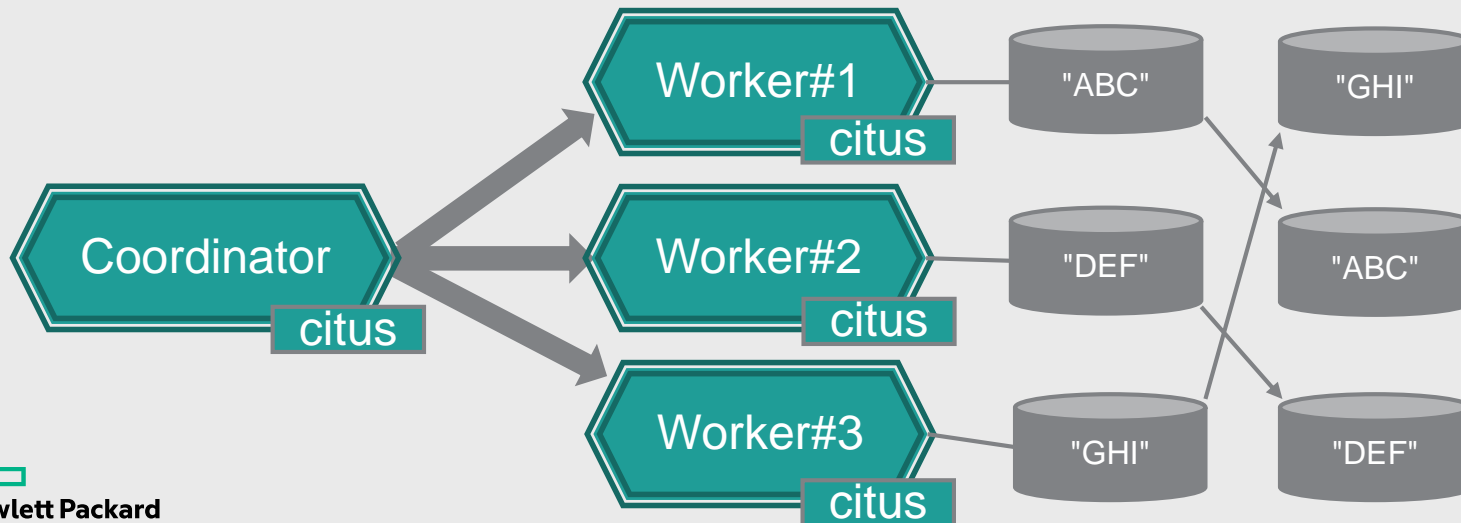
### ✓ Distributed Table

- データを分散して保存するテーブル
- ファクトテーブル向き
- 分散キーとして列を指定 (ハッシュ値の範囲によって分散先テーブルを決定)
- 分割数を指定可能

`citus.shard_count` (デフォルト値 32)

- 異なる Worker Node にレプリカを作成可能

`citus.shard_replication_factor` (デフォルト値 1 = レプリカを作らない)



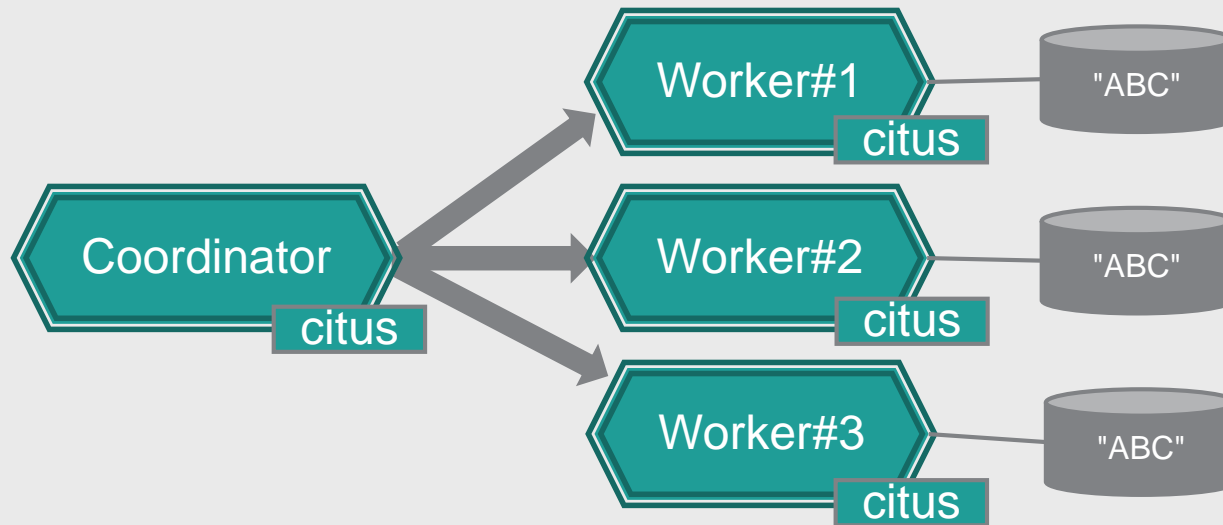


# Citus とは？

## テーブル構成

### ✓Reference Table

- 全ノードに同一データを保存するテーブル
- ディメンジョンテーブル向き





試してみよう

# 試してみよう

## インストール

### ✓ビルド

```
$ ./configure  
$ make  
# make install
```

### ✓EXTENSION の確認

```
postgres=# SELECT * FROM pg_available_extensions WHERE  
          name='citus' ;
```

```
-[ RECORD 1 ]-----+-----  
name          | citus  
default_version | 8.0-8  
Installed_version |  
comment       | Citus distributed database
```



# 試してみよう

## 全インスタンス上の操作

- ✓パラメーター `shared_preload_libraries` に '`citus`' を指定

```
postgres=# SHOW shared_preload_libraries ;
shared_preload_libraries
-----
citus
(1 row)
```

- ✓EXTENSION のロード

```
postgres=# CREATE EXTENSION citus ;
CREATE EXTENSION
```



# 試してみよう

## 認証設定

- ✓Coordinator Node から Worker Node に libpq 接続を行う
- ✓Coordinator Node と Worker Node 間の通信にパスワード認証の設定が無い
- ✓パスワード不要の接続設定が必要
- ✓pg\_hba.conf ファイルの設定例 (Worker Node)

host	all	all	coordhost1/32	trust
------	-----	-----	---------------	-------

- ✓.pgpass ファイルの作成でも可 (Coordinator Node)
- ✓パラメーター `citusb.node_conninfo` にSSL の設定などの情報を指定可能

# 試してみよう

## Coordinator Node の操作

- ✓ Coordinator Node に Worker Node を登録(データベース単位に実施)
- ✓ **master\_add\_node** 関数にホスト名とポート番号を指定

```
postgres=# SELECT master_add_node('wrkhost1', 5432) ;
master_add_node
-----
(1, 1, wrkhost1, 5432, default, f, t, primary, default)
(1 row)
```



# 試してみよう

## Coordinator Node の操作

✓確認

```
postgres=> SELECT nodeid, nodename, isactive FROM pg_dist_node ;
```

nodeid		nodename		isactive
--------	--	----------	--	----------

1		wrkhost1		t
2		wrkhost2		t
3		wrkhost3		t

(3 rows)

# 試してみよう

## Distributed Table の作成

### ✓分散テーブル数とレプリカ数を指定

```
postgres=> SET citus.shard_count = 6 ;  
SET  
postgres=> SET citus.shard_replication_factor = 2 ;  
SET
```

### ✓テーブルの作成例

```
postgres=> CREATE TABLE dist1(key1 NUMERIC, val1 VARCHAR(10)) ;  
CREATE TABLE  
postgres=> SELECT create_distributed_table('dist1', 'key1') ;  
create_distributed_table  
-----  
  
(1 row)
```





# 試してみよう

## Distributed Table の作成

✓Worker Node に同一構成のテーブルが自動作成される

- テーブル名は「{元テーブル名}\_{ShardID}」
- TABLESPACE 句は伝播しない

✓Worker Node で作成されるテーブル数

- 前スライドの例では分散数 6 × コピー数 2 ÷ Worker Node 数 3 = 4 テーブル作成される



# 試してみよう

## Distributed Table の作成

✓レプリカを指定すると、同一名称のテーブルが異なる Worker Node に作成

- 名前が同じテーブルには同一データが格納

Coordinator	Worker#1	Worker#2	Worker#3
dist1	dist1_102046	dist1_102046	
		dist1_102047	dist1_102047
	dist1_102048		dist1_102048
	dist1_102049	dist1_102049	
		dist1_102050	dist1_102050
	dist1_102051		dist1_102051

# 試してみよう

## Reference Table の作成

### ✓テーブルの作成例

```
postgres=> CREATE TABLE ref1(key1 NUMERIC, val1 VARCHAR(10)) ;
CREATE TABLE
postgres=> SELECT create_reference_table('ref1') ;
create_reference_table
-----
(1 row)
```

### ✓Worker Node のテーブル確認例

```
postgres=> \d
List of relations
Schema |      Name      | Type  | Owner
-----+-----+-----+-----
public | ref1_102084    | table | demo
(1 row)
```



# アーキテクチャー

# アーキテクチャー プロセス

## ✓bgworker: task tracker

- Coordinator Node / Worker Node で起動

## ✓bgworker: Citus Maintenance Daemon

- Coordinator Node / Worker Node で起動
- 24時間ごとに <https://reports.citusdata.com> に使用状況などを送信

## ✓バックエンド・プロセス

- Coordinator Node からの接続により Worker Node で起動
- Distributed Table または Reference Table に対して SQL 文を実行
- 2秒ごとに dump\_local\_wait\_edges 関数実行
- 1分ごとに pg\_prepared\_xacts ビューを検索

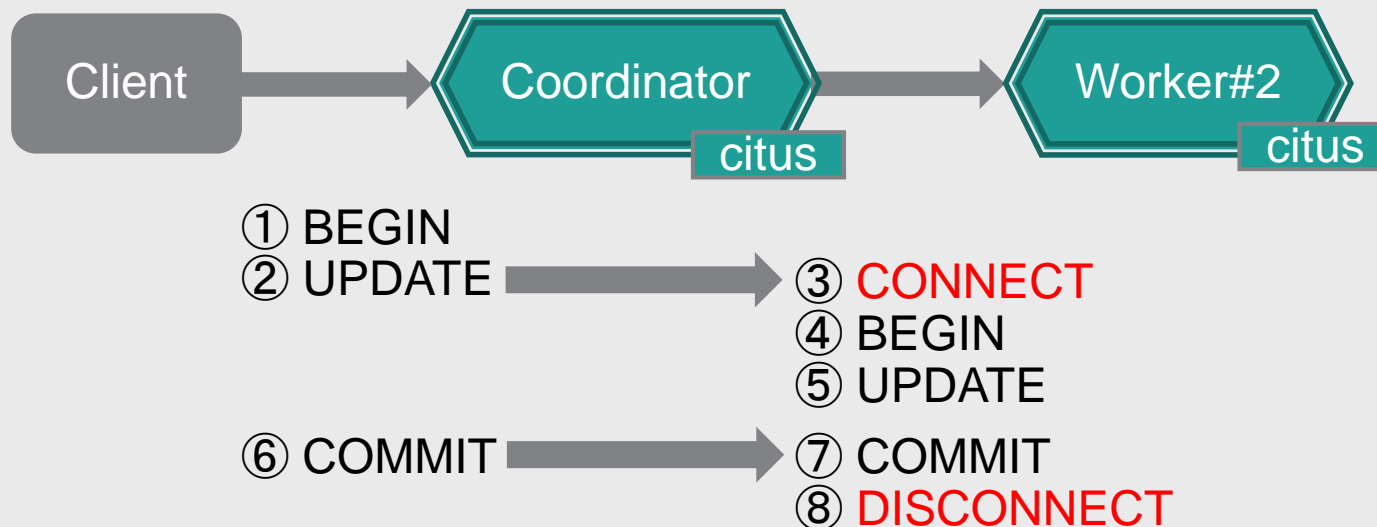


# アーキテクチャ

## セッション管理

### ✓Coordinator Node と Worker Node 間のセッション

- トランザクション内の最初の SQL 文実行時にセッション確立
- トランザクション完了と同時に切断
- **コネクション・プールを使っていない** (Enterprise Editionで実現)



# アーキテクチャー

## 実行される SQL

### ✓Coordinator Node で実行される処理

- 最終的なソート (ORDER BY)
- シーケンスの処理 (SERIAL 列、GENERATED AS IDENTITY 列含む)
- テーブルとインデックス以外のオブジェクトの処理

### ✓Worker Node にそのまま伝播する処理

- VACUUM 文
- ANALYZE 文
- CREATE INDEX 文
- ALTER TABLE 文 (一部制約あり)

### ✓その他の DML

- 分散キー列の指定により SQL 文を発行する Worker Node 上のテーブルを選択



---

# アーキテクチャー

## 実行される SQL

- ✓ Worker Node に SQL を投入する PostgreSQL API
  - PQsendQuery API
  - PQsendQueryParams API
  - 非同期 API を使用





# アーキテクチャー

## 実行される SQL


- ✓分散キー列を特定できる場合
- ✓アプリケーションが実行する SQL⇒ Worker Nodeで実行されるSQL

```
SELECT * FROM dist1 WHERE key1 = 20
```



```
SELECT key1, val1 FROM public.dist1_102131 dist1 WHERE  
      (key1 OPERATOR(pg_catalog.=) 20)
```

```
UPDATE dist1 SET val1 = 'update' WHERE key1 = 20
```



```
UPDATE public.dist1_102131 dist1 SET val1 = 'update'::character  
varying WHERE (key1 OPERATOR(pg_catalog.=) 20)
```

# アーキテクチャー

## 実行される SQL

✓分散キーを特定できる場合は特定の Worker Node のみにアクセス

```
postgres=> EXPLAIN SELECT * FROM dist1 WHERE key1 = 1000 ;  
          QUERY PLAN
```

---

Custom Scan (Citus Router) (cost=0.00..0.00 rows=0 width=0)

Task Count: 1

Tasks Shown: All

-> Task

Node: host=wrkhost1 port=5432 dbname=postgres

-> Seq Scan on dist1\_102078 dist1 (cost=0.00..2973.04  
rows=1 width=12)

Filter: (key1 = '1000'::numeric)

(7 rows)

# アーキテクチャー

## 実行される SQL

- ✓分散キー列を特定できない場合
- ✓アプリケーションが実行する SQL ⇒ Worker Nodeで実行されるSQL

```
SELECT * FROM dist1 WHERE key1 != 20
```



```
COPY (SELECT key1, val1 FROM dist1_102146 dist1 WHERE  
      (key1 OPERATOR(pg_catalog.<>) 20)) TO STDOUT
```

```
COPY (SELECT key1, val1 FROM dist1_102147 dist1 WHERE  
      (key1 OPERATOR(pg_catalog.<>) 20)) TO STDOUT
```

```
...
```

- テーブル名を変えながら全 Worker Node に投入

# アーキテクチャー

## 実行される SQL

✓分散キーを特定できない場合の実行計画

```
postgres=> SET citus.explain_all_tasks = on ;
```

```
SET
```

```
postgres=> EXPLAIN SELECT * FROM dist1 ;
```

```
QUERY PLAN
```

---

```
Aggregate (cost=0.00..0.00 rows=0 width=0)
```

```
  -> Custom Scan (Citus Real-Time) (cost=0.00..0.00 rows=0  
width=0)
```

```
    Task Count: 6
```

```
    Tasks Shown: All
```

```
    -> Task
```

```
        Node: host=wrkhost1 port=5001 dbname=demodb
```

```
        -> Aggregate (cost=2973.04..2973.05 rows=1 width=8)
```

```
...
```

# アーキテクチャー

## 実行される SQL

✓ Distributed Table と Reference Table の結合

- Worker Node 内で結合

✓ アプリケーションが発行する SQL ⇒ Worker Node で実行される SQL

```
SELECT * FROM dist1 d1 INNER JOIN ref1 r1 ON d1.key1=r1.key1 WHERE  
d1.key1=2
```



```
SELECT d1.key1, d1.val1, r1.key1, r1.val1 FROM (public.dist1_102221  
d1 JOIN public.ref1_102084 r1 ON ((d1.key1 OPERATOR(pg_catalog.=)  
r1.key1))) WHERE (d1.key1 OPERATOR(pg_catalog.=) (2)::numeric)
```

# アーキテクチャー

## 実行される SQL

### ✓ Distributed Table 同士の結合

- デフォルトではエラーになる場合がある
- パラメーター `citus.enable_repartition_joins` を on に指定することで実行可能になる

### ✓ 同一の列値を持つテーブルを同じノードに配置することを推奨

```
CREATE TABLE event(tenant_id INT, event_id BIGINT, ... ) ;
SELECT create_distributed_table(' event', ' tenant_id') ;

CREATE TABLE page(tenant_id INT, page_id INT, ... ) ;
SELECT create_distributed_table(' page', ' tenant_id',
                                colocate_with => ' event') ;
```



# アーキテクチャー

## パラメーター設定

### ✓ 主なパラメーター (全部で38種類)

- citus.node\_connection\_timeout
- citus.partition\_buffer\_size
- citus.recover\_2pc\_interval
- citus.remote\_task\_check\_interval
- citus.shard\_count
- citus.shard\_max\_size
- citus.shard\_placement\_policy
- citus.shard\_replication\_factor
- citus.subquery\_pushdown
- citus.task\_assignment\_policy
- citus.task\_executor\_type
- citus.task\_tracker\_delay
- citus.use\_secondary\_nodes
- ...



# アーキテクチャー カタログ

✓主なカタログ(全部で11個)

カタログ名	説明
pg_dist_authinfo	パスワード等の接続情報を格納(Enterprise Edition)
pg_dist_colocation	co-location 情報
pg_dist_node	Worker Node の一覧
pg_dist_node_metadata	Server ID の情報
pg_dist_partition	テーブル列の分散方法
pg_dist_placement	ShardID と Worker Node の対応
pg_dist_poolinfo	接続プーリング機能(Enterprise Edition)
pg_dist_shard	テーブルの分散に関する情報







# 制約



# 制約

## 実行できないSQL

✓以下の構文は Distributed Table に対して実行できない

- 分散キ一列の更新 (UPDATE / INSERT ON CONFLICT)
- **SELECT FOR UPDATE / SHARE 文** (レプリカを作成している場合)
- TABLESAMPLE 句
- WITH RECURSIVE 句
- INSERT VALUES 文に対する generate\_series 関数等

✓マニュアルに記載あり

```
postgres=> BEGIN ;
BEGIN
postgres=> SELECT * FROM dist1 WHERE key1=100 FOR UPDATE ;
ERROR:  could not run distributed query with FOR UPDATE/SHARE
commands
HINT:  Consider using an equality filter on the distributed table's
partition column.
```

# 制約

## 制限があるSQL

✓以下の構文は実行に対して制限あり

- 相関サブクエリー
- GROUPING SETS 句
- PARTITION BY 句
- Local Table と Distributed Table の結合
- Coordinator Node 上のテーブルに作成されたトリガー
- INSERT SELECT ON CONFLICT 文

```
postgres=> SELECT COUNT(*) FROM dist1 d INNER JOIN local1 l ON  
           d.key1 = l.key1 ;
```

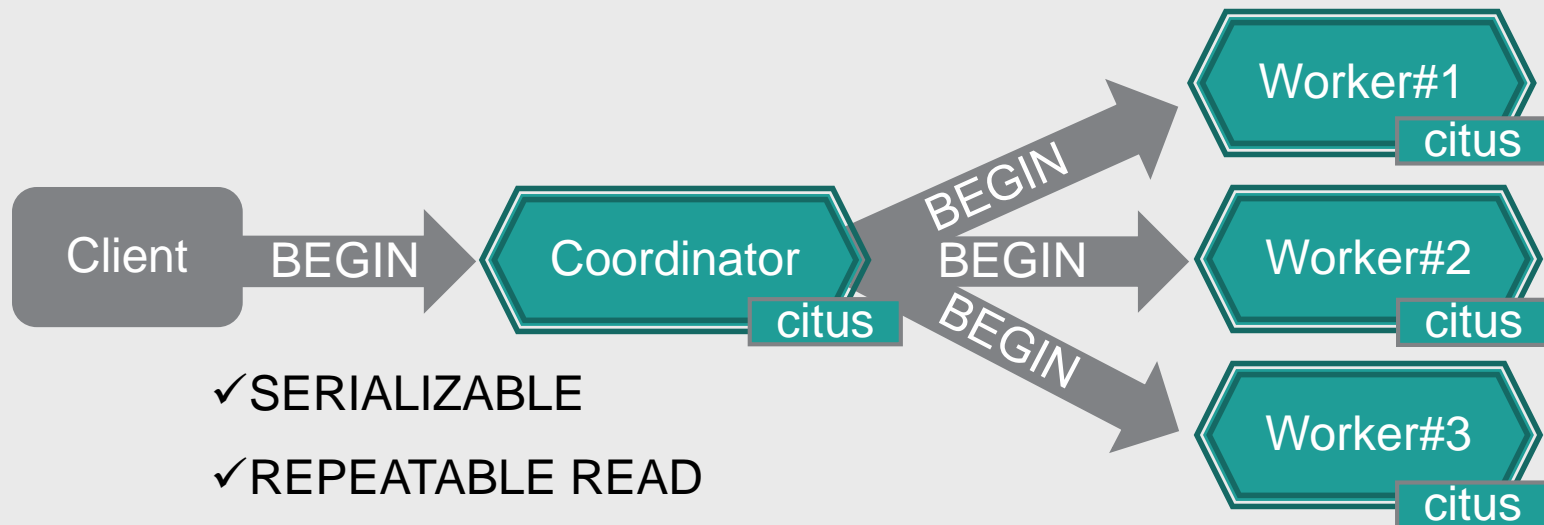
```
ERROR:  relation local1 is not distributed
```

```
postgres=> SELECT COUNT(*) FROM dist1 d INNER JOIN  
           (SELECT * FROM local1) l ON d.key1 = l.key1 ;
```

# 制約

## ISOLATION LEVEL

- ✓クライアントから Coordinator Node への接続
  - 制約なし
- ✓Coordinator Node から Worker Node への接続
  - **READ COMMITTED 固定** (ハードコード)



- ✓SERIALIZABLE
- ✓REPEATABLE READ
- ✓READ COMMITTED
- ✓READ UNCOMMITTED

✓READ COMMITTED

# 制約

## テーブルの変更

✓ALTER TABLE 文は下記しか実行できない

- 列の追加／削除
- 制約の設定
- パーティションの管理
- データ型の変更

✓DDL の自動伝播を制御可能

- パラメーター `citus.enable_ddl_propagation` (デフォルト値 on)

```
postgres=> ALTER TABLE dist1 SET UNLOGGED ;  
ERROR: alter table command is currently unsupported  
DETAIL: Only ADD|DROP COLUMN, SET|DROP NOT NULL, SET|DROP DEFAULT,  
ADD|DROP CONSTRAINT, SET (), RESET (), ATTACH|DETACH PARTITION and  
TYPE subcommands are supported.
```




# 制約

## 伝播しないSQL

✓ DATABASE や USER 情報は同一構成にすべき

- CREATE USER / CREATE DATABASE 文は伝播しない
- 警告が出力される
- Worker Node で SQL 文を実行する関数 `run_command_on_workers` を提供

```
postgres=# CREATE DATABASE demodb ;  
NOTICE: Citus partially supports CREATE DATABASE for distributed  
databases  
DETAIL: Citus does not propagate CREATE DATABASE command to workers  
HINT: You can manually create a database and its extensions on  
workers.  
CREATE DATABASE
```



# トラブル発生時の動作

---

# トラブル発生時の動作

## Coordinator Node の停止

- ✓ Coordinator Node が停止するとクライアント接続不可
- ✓ 自動フェイル・オーバーの機能は提供されていない
- ✓ Streaming Replication + Clusterware 等に対応





# トラブル発生時の動作

## Worker Node の停止

- ✓ Worker Node が停止すると停止ノードを含むデータ全体を更新する SQL は実行不可

```
postgres=> DELETE FROM dist1 ;  
ERROR:  connection error: wrkhost1:5432  
DETAIL:  could not connect to server: Connection refused  
          Is the server running on host "workhost1" (192.168.1.101)  
and accepting  
          TCP/IP connections on port 5432?
```

- ✓ 停止したノードのデータ以外を操作する SQL は警告が出力されるが実行可能

```
postgres=> SELECT COUNT(*) FROM dist1 WHERE key1 = 100 ;  
WARNING:  connection error: wrkhost1:5432  
DETAIL:  could not send data to server: Connection refused  
Could not send startup packet: Connection refused  
...
```

# トラブル発生時の動作

## Worker Node の停止

✓レプリカが存在するレコードは警告が出力されるが、更新可能

- 停止していた Worker Node のレプリカのメンテナンスが維持されなくなる

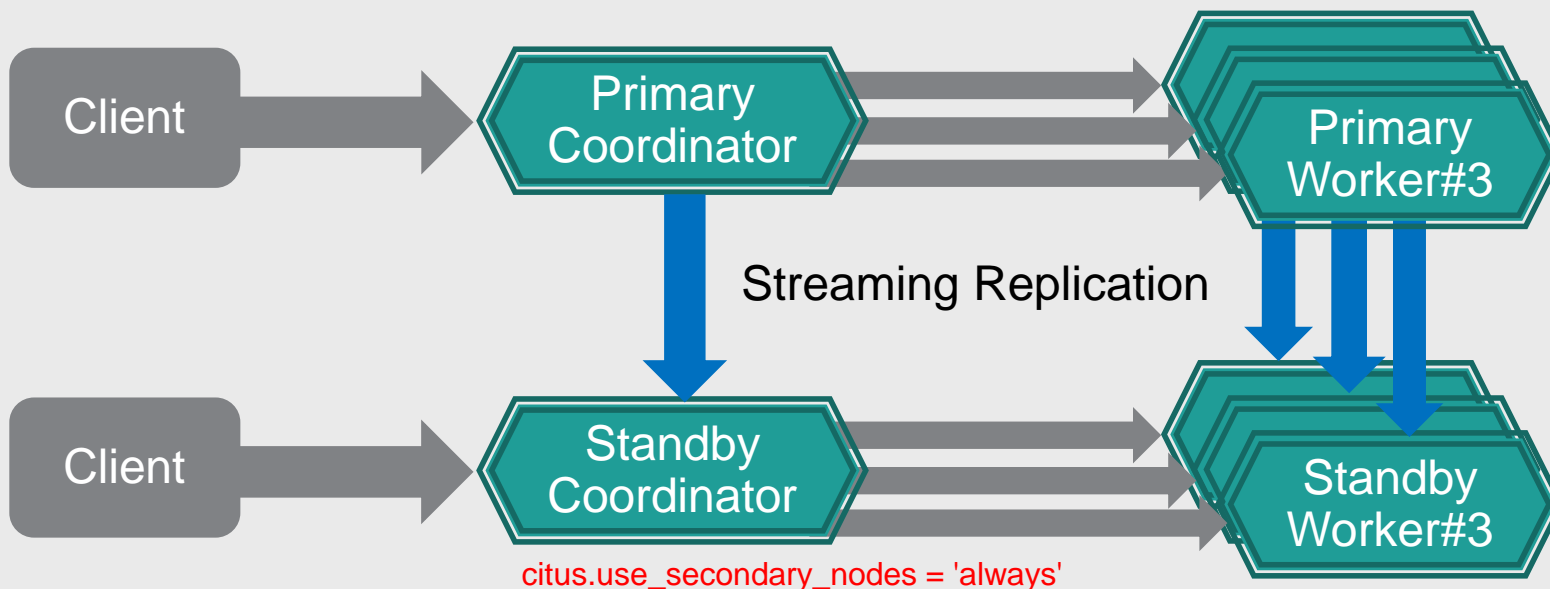
```
postgres=> DELETE FROM dist1 WHERE key1 = 1 ;  
WARNING:  connection error: wrkhost1:5432  
DETAIL:  could not send data to server: Connection refused  
could not send SSL negotiation packet: Connection refused  
DELETE 1
```

# トラブル発生時の動作

## ストリーミング・レプリケーションとの組み合わせ

✓ 負荷分散用にストリーミング・レプリケーション環境を利用可能

- 全てのノードに対してストリーミング・レプリケーション環境を作成
- Coordinator Node のスタンバイに `citpus.use_secondary_nodes = 'always'` を設定
- SELECT 文は Worker Node のスタンバイ・インスタンスで実行される
- スタンバイ・インスタンスは `master_add_secondary_node` 関数で登録しておく





まとめ



---

# まとめ

## 制約と運用に注意すれば気軽にスケールアウトが可能

- ✓比較的簡単にスケールアウト環境を構築
- ✓ノードをまたいだパラレル・クエリー＋パーティショニングによる性能改善の可能性
- ✓障害対策やバックアップは独自に実装が必要
- ✓SQL 文の実行制約があるので事前のアプリケーション検証を推奨
- ✓Enterprise Edition との違いに注意

---

# まとめ

## 参考になる情報URL

- ✓製品マニュアル

<https://docs.citusdata.com/en/v8.0/>

- ✓性能比較ビデオ

<https://www.youtube.com/watch?v=g3H4nGsJsl0>

- ✓Getting Started

[https://docs.citusdata.com/en/v8.0/portals/getting\\_started.html](https://docs.citusdata.com/en/v8.0/portals/getting_started.html)

- ✓Use Cases

[https://docs.citusdata.com/en/v8.0/portals/use\\_cases.html](https://docs.citusdata.com/en/v8.0/portals/use_cases.html)

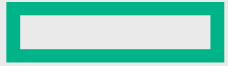
- ✓API / Reference

<https://docs.citusdata.com/en/v8.0/portals/reference.html>

- ✓GitHub

<https://github.com/citusdata/citus>





**Hewlett Packard**  
Enterprise

**Thank you**

noriyoshi.shinoda@hpe.com