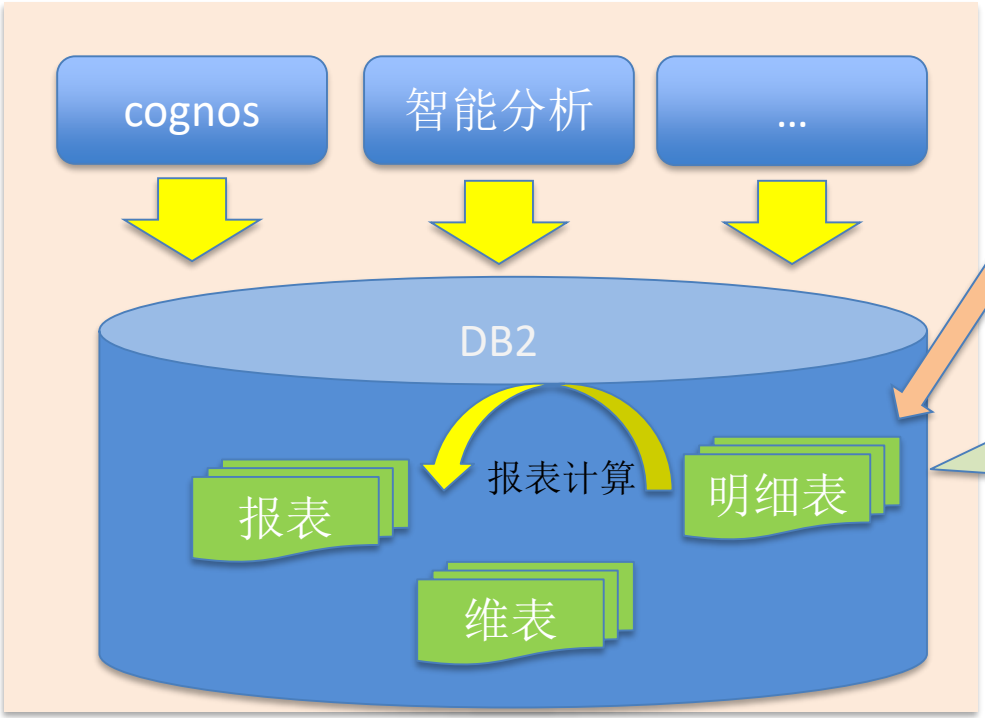
苏宁citus分布式数据库应用实践

嘉宾：陈华军
公司：苏宁云商

目录

- **业务场景**
- Citus介绍
- 部署方案
- 踩过的坑

业务场景



抽取

业务系统1

业务系统2

...

Spark任务

更新

- 当时的数据库(DB2)
  - ✓ 硬件：24C/256GB/4800GB SSD
  - ✓ 平时：45% CPU利用率
  - ✓ 大促：80%+ CPU利用率(不堪重负！)

DB负载在哪里？

■ 明细更新

- ✓ 每5分钟更新约10张明细表(要求在1分钟内完成)
- ✓ 最宽的表400字段,2.5KB/行
- ✓ 最宽的表每次更新约10w记录，总体约30w
- ✓ 保留最近数天数据(最宽的表约1000w数据，总体约3000w)

✓ 单表1000w记录
✓ 单行400字段，2.5KB
✓ **5k/s**随机更新

■ 报表计算

- ✓ 每5分钟计算30+报表(要求在2分钟内完成)
- ✓ 每个报表平均执行14次明细表集合查询

✓ **200次/min**明细表聚合运算

■ 报表查询/明细查询

- ✓ 要求并发度>30
- ✓ 要求RT<3秒

年内负载还会增加10倍

- 5k/s明细表随机更新
- 3000w明细表数据

- 5w+/s明细表随机更新
- 3亿明细表数据

Scale Out + 去DB2

方案选型

方案	优点	缺点	结论
Greenplum	✓ 支持列存 ✓ 支持压缩 ✓ SQL兼容好	✓ 更新慢 ✓ 并发低	不符合明细更新的性能要求
postgres_fdw+ pg_pathman	✓ SQL兼容好	✓ 不支持聚合下推 ✓ 不支持并行查询 ✓ 分片表管理不便	不符合明细表查询性能要求
PG-XL	✓ SQL兼容好	✓ GTM对性能的影响？ ✓ 稳定性，维护成本？	未深入评估
citus	✓ Just a extension ✓ 分片表管理方便 ✓ 成功案例较多	✓ 部分SQL不支持	**基本匹配业务场景**

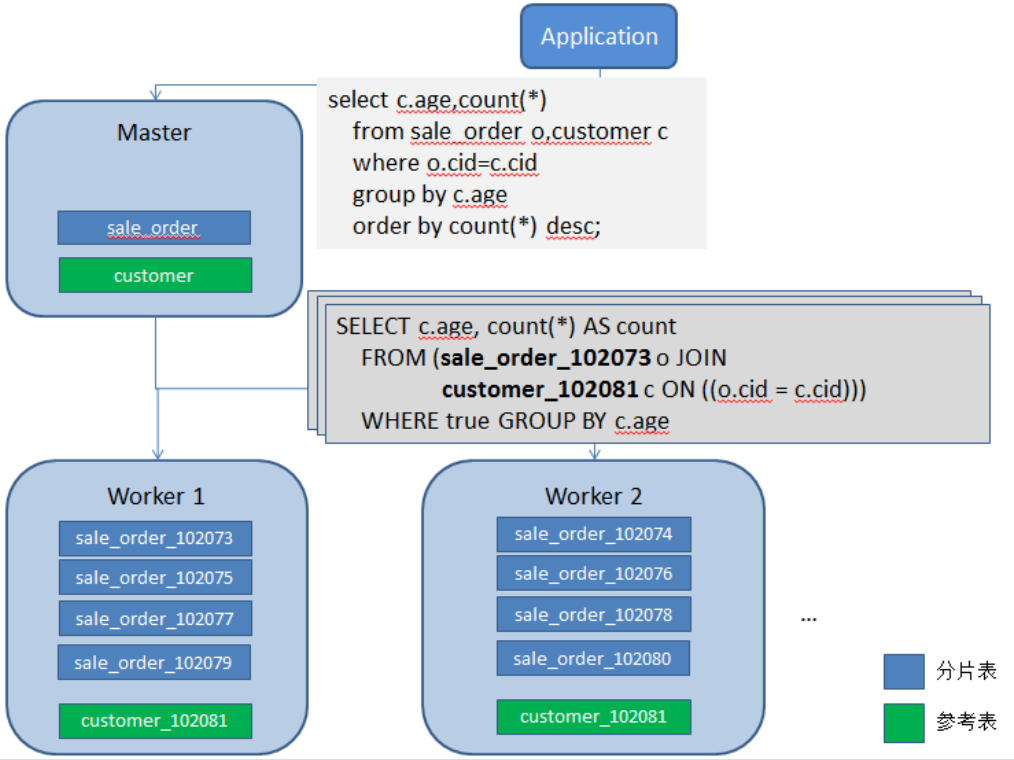
注:hadoop,spark等大数据解决方案无法适配该系统复杂的负载，首先被业务方否定。

目录

- 业务场景
- **Citus介绍**
- 部署方案
- 踩过的坑

Citus架构与原理

- 适用场景
  - ✓ 实时数据分析
  - ✓ 多租户应用
- 数据分布
  - ✓ 分片表
    - ➢ hash
    - ➢ append
    - ➢ range(*)
  - ✓ 参考表
  - ✓ 本地表



注:手册中未公开range分片方法

案例演示-建表

创建分片表sale_order

```
create table sale_order(oid int PRIMARY KEY,cid int,other text);
citus.shard_replication_factor=1;//可省，默认为1
set citus.shard_count =8;

//oid作为分片列,默认分片方法为hash
select create_distributed_table('sale_order', 'oid');
```

创建参考表customer

```
create table customer(cid int,age int,other text);
select create_reference_table('customer');
```

元数据

- pg_dist_partition
- pg_dist_shard
- pg_dist_shard_placement

1. 在每worker上创建分片
2. 更新元数据

元数据- pg_dist_partition

分片表sale_order

```
postgres=# select * from pg_dist_partition where logicalrelid='sale_order'::regclass;
-[ RECORD 1 ]+----------------------------------------------------------------------------------------------------------------------------------
-
logicalrelid | sale_order
partmethod   | h
partkey      | {VAR :varno 1 :varattno 1 :vartype 23 :vartypmod -1 :varcollid 0 :varlevelsup 0 :varnoold
1 :varoattno 1 :location -1}
colocationid | 24
repmodel     | c
```

Hash分片

分片列为第1字段

参考表customer

```
postgres=# select * from pg_dist_partition where logicalrelid='customer'::regclass;
 logicalrelid | partmethod | partkey | colocationid | repmodel
--------------+------------+---------+--------------+----------
 customer     | n          |         |            9 | t
(1 row)
```

none,代表参考表

元数据- pg_dist_shard

分片表sale_order

```
postgres=# select * from pg_dist_shard where logicalrelid = 'sale_order'::regclass;
 logicalrelid | shardid | shardstorage | shardminvalue | shardmaxvalue
--------------+---------+--------------+---------------+---------------
 sale_order   | 105007 | t           | 1610612736    | 2147483647
 sale_order   | 105006 | t           | 1073741824    | 1610612735
 sale_order   | 105005 | t           | 536870912     | 1073741823
 sale_order   | 105004 | t           | 0             | 536870911
 sale_order   | 105003 | t           | -536870912    | -1
 sale_order   | 105002 | t           | -1073741824   | -536870913
 sale_order   | 105001 | t           | -1610612736   | -1073741825
 sale_order   | 105000 | t           | -2147483648   | -1610612737
(8 rows)
```

定义8个分片对应的hash范围

参考表customer

```
postgres=# select * from pg_dist_shard where logicalrelid = 'customer'::regclass;
 logicalrelid | shardid | shardstorage | shardminvalue | shardmaxvalue
--------------+---------+--------------+---------------+---------------
 customer     | 104999 | t           |               |
(1 row)
```

参考表只有1个分片

元数据-pg_dist_shard_placement

分片表sale_order

```
postgres=# select * from pg_dist_shard_placement where shardid in (select shardid from
pg_dist_shard where logicalrelid = 'sale_order'::regclass);
 shardid | shardstate | shardlength |      nodename       | nodeport | placementid
---------+------------+-------------+---------------------+----------+-------------
  105007 |          1 |           0 | /var/run/postgresql |    60002 |        3010
  105006 |          1 |           0 | /var/run/postgresql |    60001 |        3009
  105005 |          1 |           0 | /var/run/postgresql |    60002 |        3008
  105004 |          1 |           0 | /var/run/postgresql |    60001 |        3007
  105003 |          1 |           0 | /var/run/postgresql |    60002 |        3006
  105002 |          1 |           0 | /var/run/postgresql |    60001 |        3005
  105001 |          1 |           0 | /var/run/postgresql |    60002 |        3004
  105000 |          1 |           0 | /var/run/postgresql |    60001 |        3003
(8 rows)
```

定义8个分片在 worker上的分布

参考表customer

```
postgres=# select * from pg_dist_shard_placement where shardid in (select shardid from
pg_dist_shard where logicalrelid = 'customer'::regclass);
 shardid | shardstate | shardlength |      nodename       | nodeport | placementid
---------+------------+-------------+---------------------+----------+-------------
  104999 |          1 |           0 | /var/run/postgresql |    60002 |        3002
  104999 |          1 |           0 | /var/run/postgresql |    60001 |        3001
(2 rows)
```

参考表在每个worker上存1个副本

分布式执行计划

```
postgres=# explain select c.age,count(*) from sale_order o,customer c where o.cid=c.cid group by c.age order by count(*) desc;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Distributed Query into pg_merge_job_8279326
   Executor: Real-Time
   Task Count: 8
   Tasks Shown: One of 8
   ->  Task
         Node: host=/var/run/postgresql port=60001 dbname=lma
         ->  HashAggregate  (cost=316.75..318.75 rows=200 width=12)
               Group Key: c.age
               ->  Merge Join  (cost=166.75..280.75 rows=7200 width=4)
                     Merge Cond: (o.cid = c.cid)
                     ->  Sort  (cost=83.37..86.37 rows=1200 width=4)
                           Sort Key: o.cid
                           ->  Seq Scan on sale_order_105000 o  (cost=0.00..22.00 rows=1200 width=4)
                     ->  Sort  (cost=83.37..86.37 rows=1200 width=8)
                           Sort Key: c.cid
                           ->  Seq Scan on customer_104999 c  (cost=0.00..22.00 rows=1200 width=8)
 Master Query
   ->  Sort  (cost=0.00..0.00 rows=0 width=0)
         Sort Key: COALESCE((pg_catalog.sum((COALESCE((pg_catalog.sum(intermediate_column_8279326_1))::bigint, '0'::bigint))))::bigint, '0'::bigint) DESC
         ->  HashAggregate  (cost=0.00..0.00 rows=0 width=0)
               Group Key: intermediate_column_8279326_0
               ->  Seq Scan on pg_merge_job_8279326  (cost=0.00..0.00 rows=0 width=0)
(22 rows)
```

在每个worker上预聚合(每分片并行执行)

在master上汇总

SQL限制-查询

■ Join的限制
✓ 不支持2个非亲和分片表的outer join
✓ 仅task-tracker执行器支持2个非亲和分片表的inner join
✓ 对分片表和参考表的outer join，参考表只能出现在left join的右边或right join的左边

■ 子查询的限制
✓ 子查询不能参与join
✓ 不能出现order by，limit和offset

■ 回避方法
✓ 通过Hll(HyperLogLog)插件支持count(distinct)
✓ 通过临时表(或dblink)中转

■ 不支持的SQL特性
✓ CTE
✓ Window函数
✓ 集合操作，比如 union
✓ 非分片列的count(distinct)

■ 其它限制
✓ 本地表不能和分片表(参考表)混用
✓ …

参考: http://blog.chinaunix.net/xmlrpc.php?r=blog/article&uid=20726500&id=5761942

SQL限制-更新

- ■ 不支持跨分片的更新SQL
- ■ 不支持跨分片的事务
- ■ ‘insert into … select … from …’的支持存在部分限制
- ✓ 插入源表和目的表必须是具有亲和性的分片表
- ✓ 不允许出现Stable and volatile函数
- ✓ 不支持LIMIT，OFFSET，窗口函数，集合操作，Grouping sets，DISTINCT

Citus V7已经改进了insert … select

- ■ 回避方法
- ✓ copy代替insert
- ✓ SELECT master_modify_multiple_shards(‘…’)实现扩分片更新
- ✓ Dblink()拆分事务
- ✓ 临时表(分片表->本地表)
- ✓ SELECT create_insert_proxy_for_table(‘…’)(本地表->分片表)

SQL限制-DDL(1)

DDL	是否支持	备注
DROP TABLE	支持	
CREATE INDEX	支持	
DROP INDEX	支持	
ANALYZE	支持	
VACUUM	支持	
CREATE VIEW	支持	在master本地创建视图
LOCK	支持	在master本地LOCK表
ALTER INDEX	不支持	
CLUSTER	不支持	不报错,在master本地CLUSTER表
ALTER TABLE	部分支持	见备注1

SQL限制-DDL(2)

备注1：ALTER TABLE只支持以下子命令
Only ADD|DROP COLUMN, SET|DROP NOT NULL, SET|DROP DEFAULT, ADD|DROP CONSTRAINT FOREIGN KEY and TYPE subcommands are supported.

■ 回避方法
✓ DROP INDEX + CREATE INDEX代替ALTER INDEX
✓ 创建对等的唯一索引代替变更主键
✓ 通过`run_command_on_placements`函数，直接在所有分片位置上执行DDL

例:

```
SELECT * FROM run_command_on_placements('big_tb', 'alter table %s set(fillfactor=65)');
```

两种执行器（1）

```
set citus.task_executor_type='task-tracker'|'real-time'
```

执行器
- real-time (默认)
  - router
    - 适用条件：只需在一个shard上执行的SQL
    - 连接管理： 1个master后端进程对每个worker只创建一个连接，并缓存连接
    - 示例: select ... from tb1 where 分片列=...

    对应OLTP场景
  - 非router

    master后端进程对所有worker上的所有shard同时发起连接，并执行SQL，SQL完成后断开连接。
- task-tracker

  master只和worker上的task-tracker进程交互，task-tracker进程负责worker上的任务调度，任务结束后master从worker上取回结果。worker上总的并发任务数可以通过参数控制。

对应OLAP场景

两种执行器（2）

执行器	优点	缺点
real-time (默认)	RT小	Worker上并发放大，可能导致瞬间资源消耗(连接数)暴增。放大倍数为该分片表分配在单个worker上shard数
task-tracker	✓支持数据重分布，SQL支持比real-time略好。 ✓并发数，资源消耗可控	✓ RT大 ✓ 要通过临时表暂存结果，额外消耗略高于real-time

目录

- 业务场景
- Citus介绍
- **部署方案**
- 踩过的坑

痛点1：随机更新速度

性能目标是5w+/s。看似比较接近性能要求，实际上性能远远不够 (宽表，其它负载，性能余量)

■ 官方数据

✓ Real-time Inserts:0-50k/s
✓ **Real-time Updates:0-50k/s**
✓ Bulk Copy:100-200k/s
✓ Masterless Citus:50k/s-500k/s

■ 插入优化(citus 8worker)

部署方式	TPS	备注
单机PG	134030	
Citus	55717	Master是性能瓶颈
Citus(不解析SQL)	75973	改citus代码
Citus(masterless)	20w+	结果仍不理想(8worker只比单机略好)

参考:

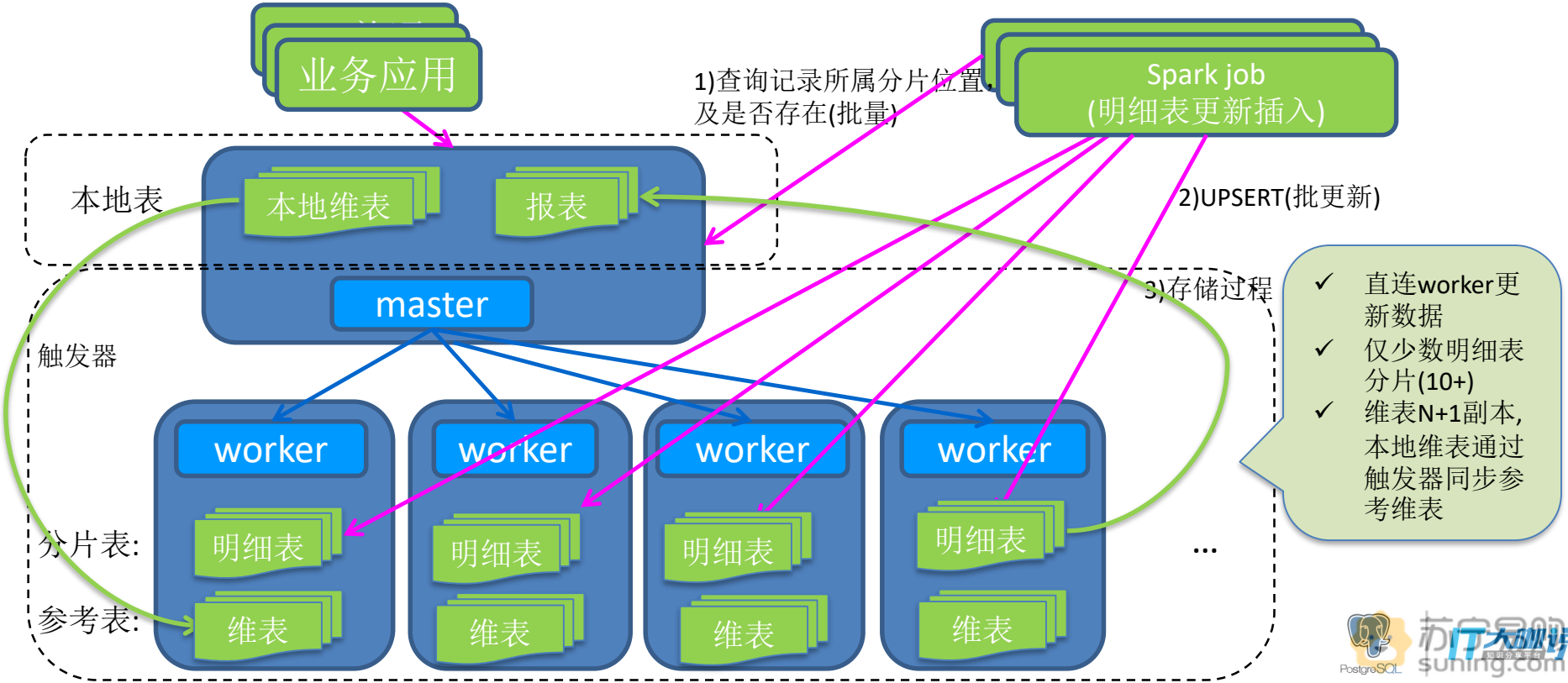
✓ https://docs.citusdata.com/en/v6.1/performance/scaling_data_ingestion.html
✓ http://blog.chinaunix.net/xmlrpc.php?r=blog/article&uid=20726500&id=5761214
✓ http://blog.chinaunix.net/xmlrpc.php?r=blog/article&uid=20726500&id=5761937

痛点2：SQL限制

- 应用需要大量使用citus不支持的SQL
  - ✓ 多表left join
  - ✓ 子查询参与join
  - ✓ union all
  - ✓ count(distinct)
  - ✓ 多记录DML
  - ✓ …

应用改造量大

解决方案


业务应用
1)查询记录所属分片位置，及是否存在(批量)
Spark job
(明细表更新插入)
本地表
本地维表
报表
2)UPSERT(批更新)
master
3)存储过程
触发器
worker
worker
worker
worker
分片表:
明细表
明细表
明细表
明细表
...
参考表:
维表
维表
维表
维表
✓ 直连worker更新数据
✓ 仅少数明细表分片(10+)
✓ 维表N+1副本，本地维表通过触发器同步参考维表

辅助工具函数开发

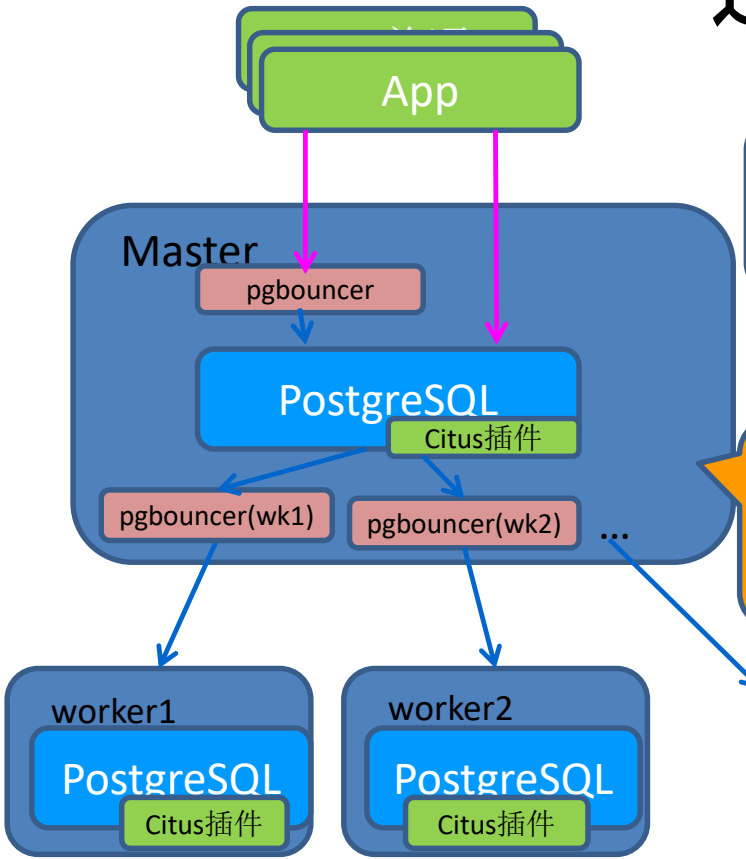
- 批量获取记录所在分片位置函数

pg_get_dist_shard_placement()

- 自动生成本地维表->参考维表同步触发器的函数

create_sync_trigger_for_table()

连接池



pgbounce的作用:
1.限制发往worker的并发连接数
2.复用worker连接
3. 避免大量创建删除TCP套接字

pgbounce的限制:
1. 事务池不支持预编译语句
JDBC中可以设置prepareThreshold=0
2. 会话池不适合客户端长连接

效果

■ POC压测

1master+8worker: 16C/128GB/3000GB SSD

测试场景	测试内容	批量	并发数	性能实绩	性能目标
明细更新	分片位置查询	1000条/次	32	16w/s	10w/s
	更新worker	100条/次	32	2.6w/s * 8	10w/s
报表计算	报表计算		16	33/min	17/min

■ 上线运行

✓ 1master+4worker: 16C/128GB/3000GB SSD

✓ 上线半年来运行平稳

✓ Worker/master的平均CPU利用率均在5%左右

日常维护 - DDL

- 在每个worker上批量执行DDL - run_command_on_workers ()

```
SELECT * FROM run_command_on_workers('alter system set log_statement=$$mod$$')
```

- 在每个分片上批量执行DDL - run_command_on_placements()

```
SELECT * FROM run_command_on_placements('big_tb', 'alter table %s set(fillfactor=65)')
```

- 在每个分片上批量执行DML - master_modify_multiple_shards()

```
SELECT master_modify_multiple_shards(
        'DELETE FROM customer_delete_protocol WHERE c_custkey > 500 AND c_custkey < 500')
```

日常维护 -失效副本的修复

更新多副本分片表的途中worker发生故障，可能导致该worker上的副本没有被正确更新。此时citus会在系统表pg_dist_shard_placement 中将其标识为“失效”状态。

- 多副本分片表中部分副本失效的检测

```
select * from pg_dist_shard_placement where shardstate <> 1
```

- 多副本分片表中部分副本失效的修复

master_copy_shard_placement()

```
select master_copy_shard_placement(102178,'/tmp',60002,'/tmp',60003)
```

我们的分片表使用单副本，通过主备流复制做HA

日常维护 – 2PC未决事务的处理

- Citus对以下跨库操作采用2PC保障事务一致，2PC中途发生故障会产生未决事务
  - ✓ DDL
  - ✓ copy
  - ✓ 参考表的更新
  - ✓ master_modify_multiple_shards()等
- 未决事务的检测

```
WARNING:  terminating connection due to administrator command
WARNING:  failed to commit transaction on 10.37.2.181:54323
WARNING:  failed to roll back prepared transaction 'citus_0_23473_4'
HINT:  Run "COMMIT PREPARED 'citus_0_23473_4'" on 10.37.2.181:54323
```

- 未决事务的处理

```
select recover_prepared_transactions();
```

日常维护 – 2PC事务清理

- Citus对每个2PC事务中的操作都记录到系统表pg_dist_transaction,对频繁发起2PC的系统，容易导致该系统表膨胀。需要定期执行以下SQL清理pg_dist_transaction。

```
select recover_prepared_transactions()
```

- 6.1.x以前版本，存在以下bug，务必使用高版本
  - ✓ https://github.com/citusdata/citus_docs/issues/417
  - ✓ https://github.com/citusdata/citus/pull/1505
  - ✓ https://github.com/citusdata/citus/issues/1614

日常维护 – 扩容/缩容/迁移

- 1. 增加删除worker
  - ✓ master_add_node()
  - ✓ master_remove_node()
- 2. 迁移数据
  - ✓ copy
- 3. 手动修改分片位置元数据
  - ✓ pg_dist_placement

目录

- 业务场景
- Citus介绍
- 部署方案
- **踩过的坑**

踩过的坑

- master(real-time)到worker用的短连接，pgbouncer默认记录连接和断连接事件，导致日志文件增长太快。后将其关闭

```
log_connections = 0
log_disconnections = 0
```

- master(real-time)会瞬间创建大量到worker 的并发连接，而默认的unix套接字的backlog连接数偏低， master节点的 postgres日志中经常发现大量连接出错的告警

```
could not connect to server: Resource temporarily unavailable
```

解决办法:

1. echo 2048 > /proc/sys/net/core/somaxconn
2. 修改pgbouncer的listen_backlog
3. 硬重启pgbouncer(通过-R参数的在线重启无效)

Thanks!