

2018PostgreSQL中国技术大会



Citus在苏宁的大规模应用

陈华军

chjischj@163.com

苏宁易购



目录

□ Citus的引入

□ Citus的工作方式

□ 表的设计

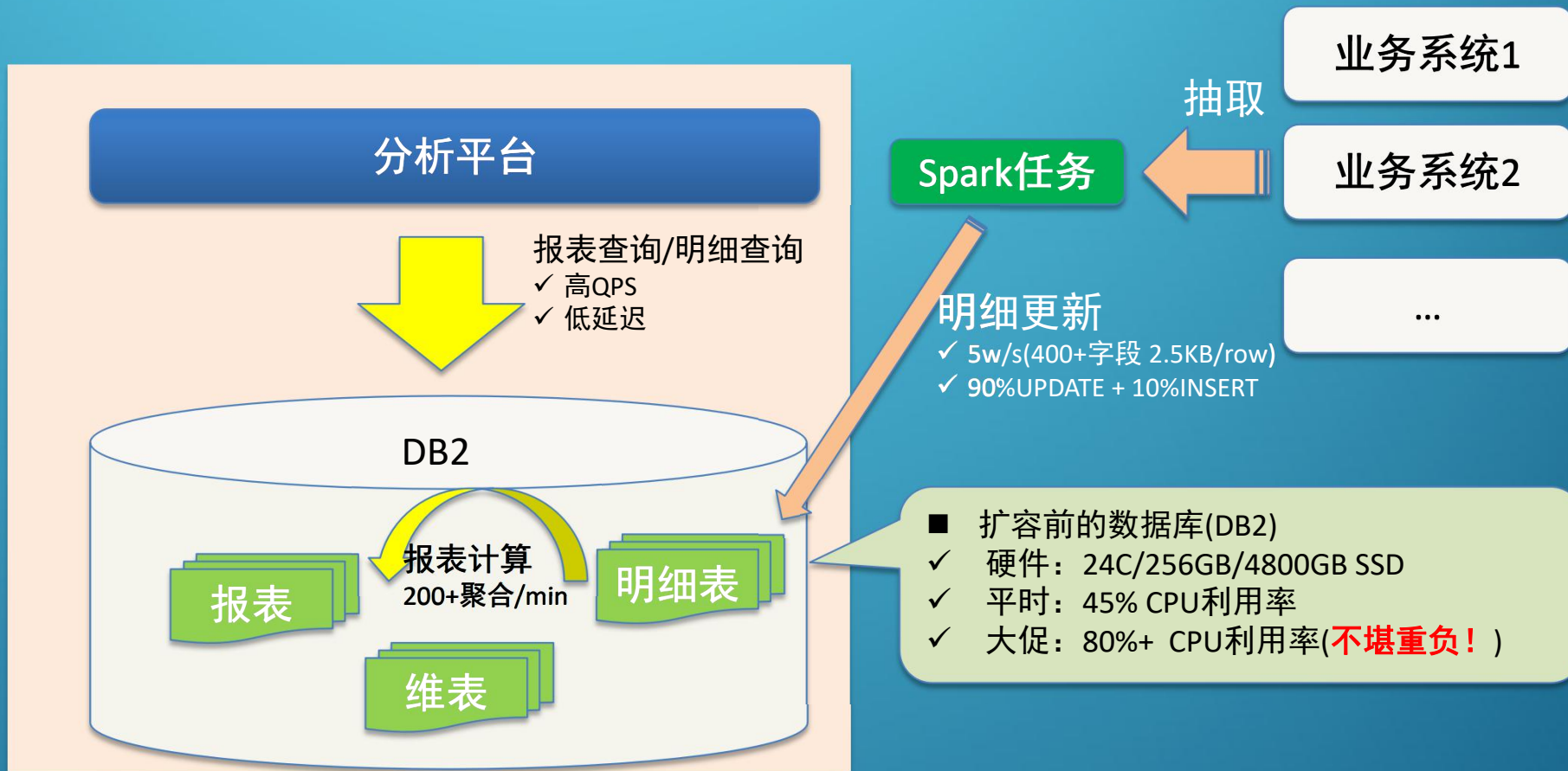
□ 部署与维护

□ 典型案例

□ 常见问题

如何支撑某业务的10倍扩容？

- 性能要求高
- 负载类型多样





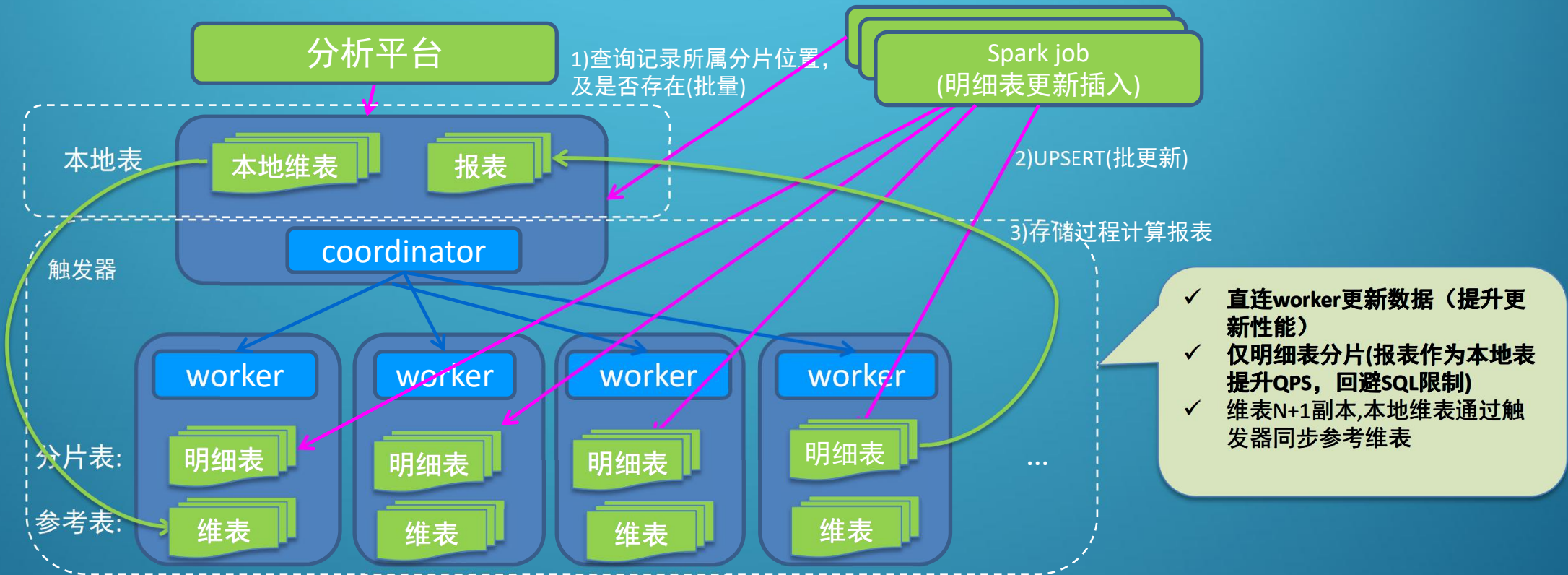
分布式数据库选型

方案	优点	缺点	结论
Greenplum	<ul style="list-style-type: none">✓ 支持列存✓ 支持压缩✓ SQL兼容好	<ul style="list-style-type: none">✓ 更新慢✓ 并发低	不符合明细更新的性能要求
postgres_fdw+ pg_pathman	<ul style="list-style-type: none">✓ SQL兼容好	<ul style="list-style-type: none">✓ 不支持聚合下推(PG10以后支持)✓ 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	<ul style="list-style-type: none">✓ SQL兼容好	<ul style="list-style-type: none">✓ GTM对性能的影响?✓ 稳定性, 维护成本?✓ 社区不活跃✓ 版本更新慢	未深入评估
citus	<ul style="list-style-type: none">✓ Just a extension✓ 分片表管理方便✓ 成功案例较多	<ul style="list-style-type: none">✓ 部分SQL不支持✓ 不支持全局一致性读	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。




基于Citrus的技术方案



Citus的优点一:社区活跃, 发展迅速

- 平均2~3个月一次大版本发布

发布日期	大版本	备注
2017/2/10	6.1	曾经使用
2017/3/17	6.2	
2017/8/28	7.0	
2017/11/16	7.1	曾经使用
2018/1/16	7.2	当前使用
2018/3/15	7.3	
2018/5/15	7.4	当前使用
2018/6/26	7.5	
2018/10/31	8.0	



2017 Postgres大象会

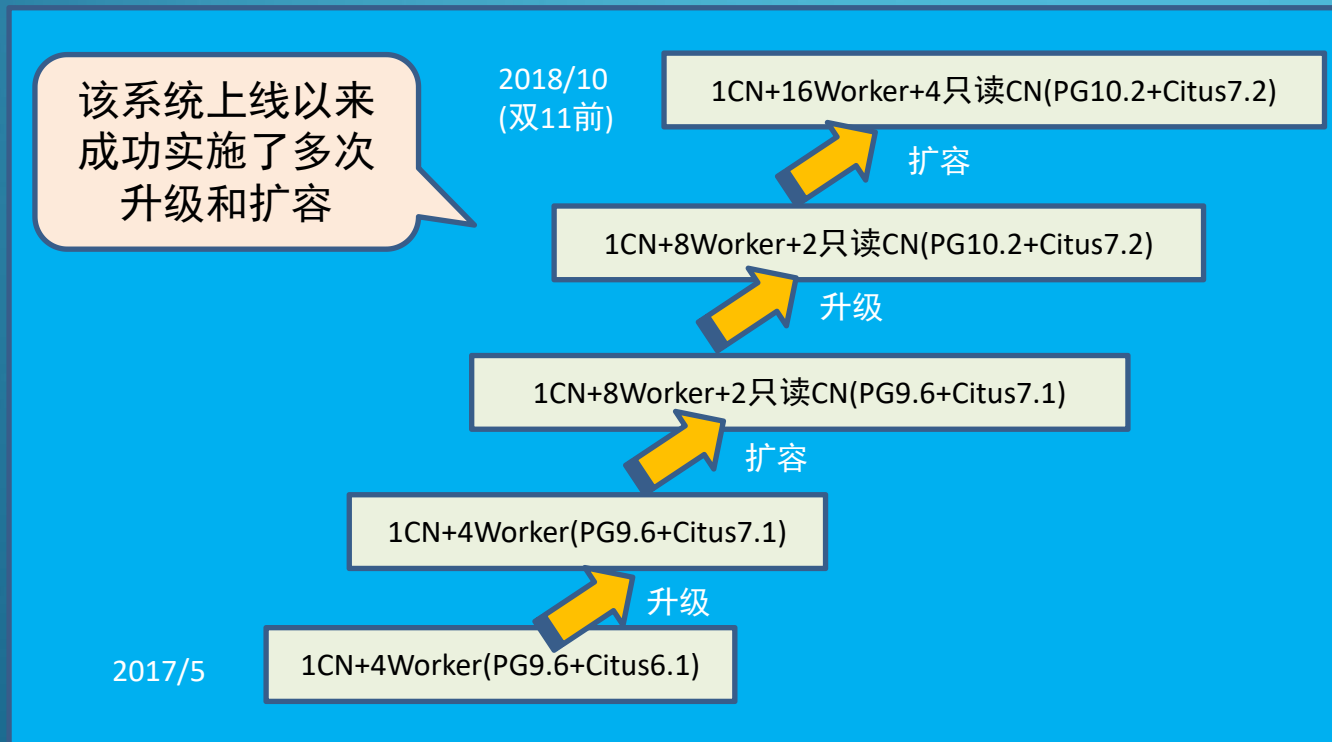
痛点2: SQL限制

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

初始上线时使用的Citus6.1中存在的大部分SQL限制到7.2的时候已经被解除(划线部分)

Citus的优点二:稳定、易维护

- 大部分维护工作和非分布式PostgreSQL相同
- 社区版不支持的扩缩容可通过修改元数据实现

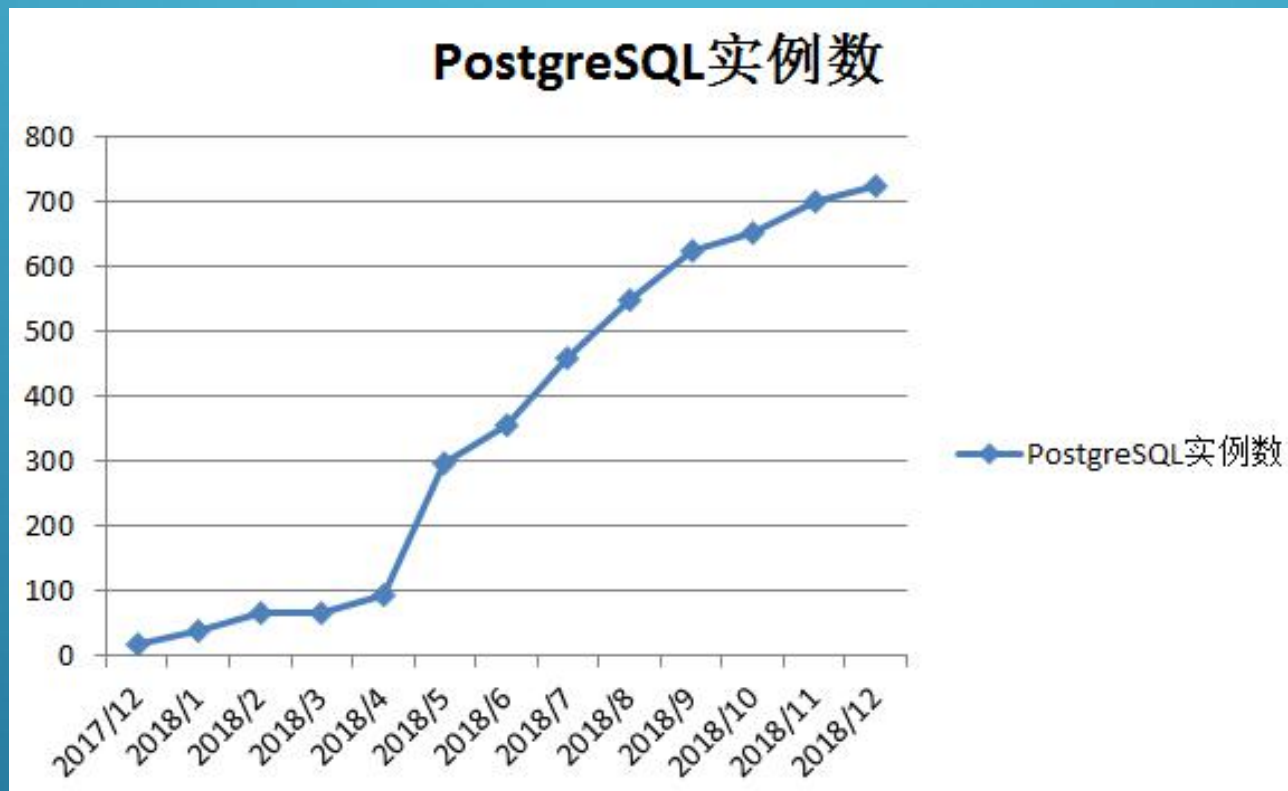


系统表 (元数据)	说明
pg_dist_node	Worker node table
pg_dist_partition	Partition table
pg_dist_shard	Shard table
pg_dist_placement	Shard placement table
pg_dist_colocation	Co-location group table

Citus “推广”

- 2018其他系统开始陆续使用Citus
- 经过1年时间已上线35套Citus集群，近千个PG节点

截止2018年底实际并未主动开展过Citus/PG推广工作，凭借已上线Citus系统的示范效应，逐渐有新的系统选择Citus。





目录

□ Citus的引入

□ Citus的工作方式

□ 表的设计

□ 部署与维护

□ 典型案例

□ 常见问题

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

按分片字段hash值分布数据

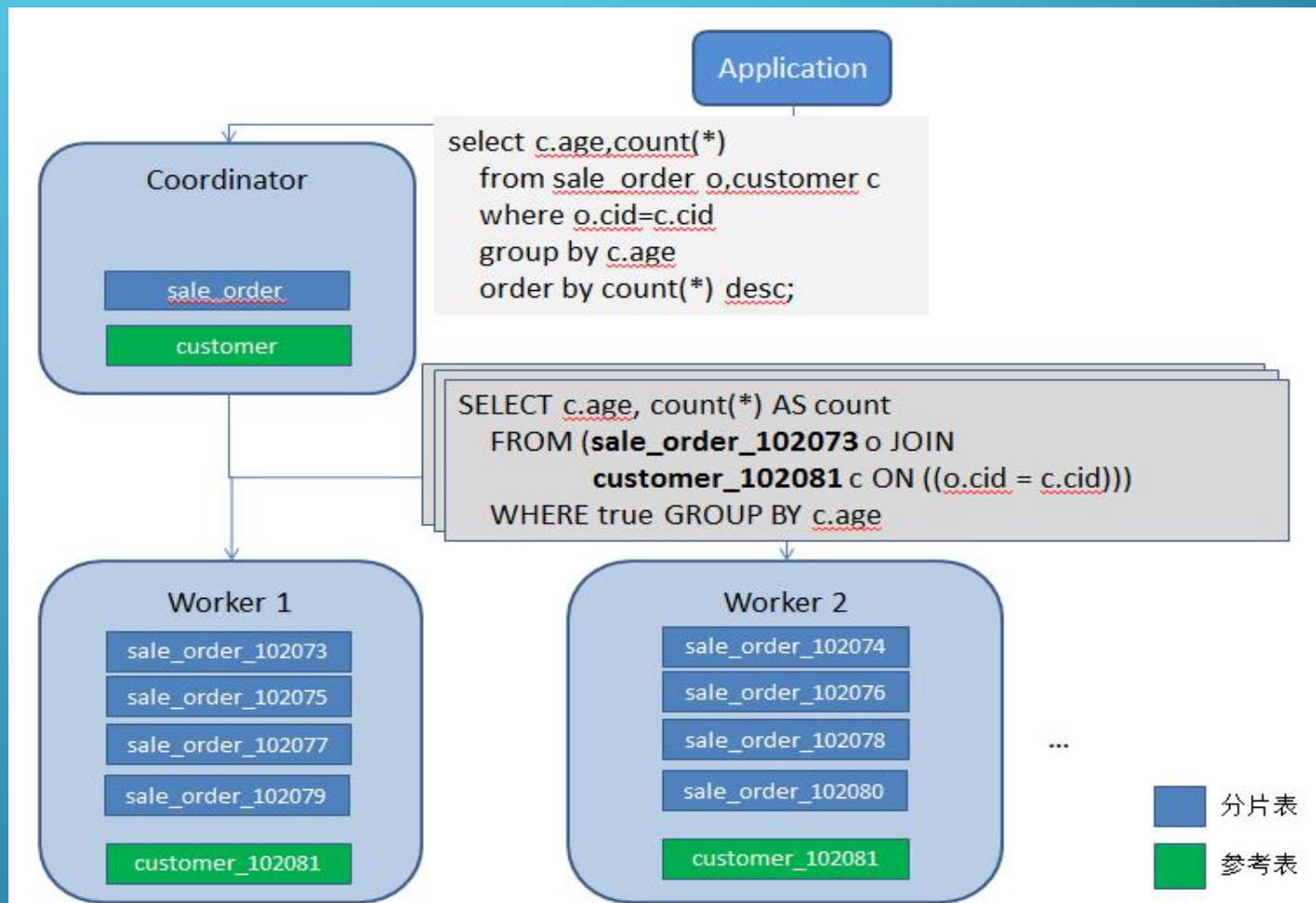
➢ Append

分片size增加到一定量自动创建新的分片

✓ 参考表

仅一个分片，每个worker一个副本，主要用于维度表。

✓ 本地表



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);  
set 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	71
repmode	s

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
```

```
-----+-----+-----+-----+-----
```

logicalrelid	partmethod	partkey	colocationid	repmode
customer	n		60	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  | 151878 | t           | 1610612736    | 2147483647
sale_order  | 151877 | t           | 1073741824    | 1610612735
sale_order  | 151876 | t           | 536870912     | 1073741823
sale_order  | 151875 | t           | 0             | 536870911
sale_order  | 151874 | t           | -536870912   | -1
sale_order  | 151873 | t           | -1073741824   | -536870913
sale_order  | 151872 | t           | -1610612736   | -1073741825
sale_order  | 151871 | 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     | 151879 | 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
151874	1	0	192.168.1.101	6432	49985
151875	1	0	192.168.1.102	6432	49986
151876	1	0	192.168.1.103	6432	49987
151877	1	0	192.168.1.104	6432	49988
151878	1	0	192.168.1.105	6432	49989
151871	1	0	192.168.1.106	6432	49982
151872	1	0	192.168.1.107	6432	49983
151873	1	0	192.168.1.108	6432	49984

(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
151879	1	0	192.168.1.101	6432	49993
151879	1	0	192.168.1.102	6432	49994
151879	1	0	192.168.1.103	6432	49995
151879	1	0	192.168.1.104	6432	49996
151879	1	0	192.168.1.105	6432	49997
151879	1	0	192.168.1.106	6432	49990
151879	1	0	192.168.1.107	6432	49991
151879	1	0	192.168.1.108	6432	49992

(11 rows)

参考表在每个worker上存储1个副本

带分片字段的查询

```
postgres=# explain select o.cid,c.age from sale_order o,customer c where o.cid=c.cid and oid=100;
```

QUERY PLAN

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

Task Count: 1

Tasks Shown: All

-> Task

Node: host=192.168.1.102 port=6432 dbname=sdaspgdb

-> Nested Loop (cost=0.15..40.17 rows=6 width=8)

Join Filter: (o.cid = c.cid)

-> Index Scan using sale_order_pkey_151875 on sale_order_151875 o (cost=0.15..3.17 rows=1 width=4)

Index Cond: (oid = 100)

-> Seq Scan on customer_151879 c (cost=0.00..22.00 rows=1200 width=8)

(10 rows)

CN后端进程分发SQL到对应 shard，对每个worker只创建一个连接，并缓存连接。
适用OLTP场景

不带分片字段的查询

```
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
```

Sort (cost=0.00..0.00 rows=0 width=0)

Sort Key: COALESCE((pg_catalog.sum((COALESCE((pg_catalog.sum(remote_scan.count))::bigint, '0'::bigint))))::bigint, '0'::bigint) DESC

-> HashAggregate (cost=0.00..0.00 rows=0 width=0)

Group Key: remote_scan.age

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

Task Count: 8

Tasks Shown: One of 8

-> Task

Node: host=192.168.1.106 port=6432 dbname=sdaspgdb

-> 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_151871 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_151879 c (cost=0.00..22.00 rows=1200 width=8)

(19 rows)

在coordinator上汇总

在每个worker上预聚合(每分片并行执行)

CN后端进程对所有worker上的所有shard同时发起连接，并行执行SQL，SQL完成后断开连接。
适用OLAP场景



分片表的亲和

- 亲和指2个或2个以上的逻辑表的分片规则，分片位置完全相同可以很好的支持SQL下推
- 亲和可以优化涉及多表的SQL和事务
 - 对一组亲和分片的查询提供完整的SQL支持

SQL场景	亲和分片表	非亲和分片表
Inner join	支持	支持(注1).需要重新分布数据, 性能差
Outter Join	支持	不支持
外键	支持	不支持
INSERT..SELECT	支持	支持,数据经过CN中转, 性能差
其他涉及多表的SQL	支持	部分支持
单个事务中更新多个表的单个分片	单库事务	分布式事务

注1: 需要设置set citus.enable_repartition_joins = on

注2: 参考表在每个worker上都有一个副本, 可以认为和所有分片表都“亲和”

注3: 参考http://docs.citusdata.com/en/v8.0/sharding/data_modeling.html#colocation

亲和性影响示例:Join

表定义

```
create table tb1(userid int,c1 text);
create table tb2(userid int,c2 text);
create table tb3(productid int,userid int, c3 text);
SELECT create_distributed_table('tb1', 'userid', colocate_with=>'none'); --新建一个亲和组1
SELECT create_distributed_table('tb2', 'userid', colocate_with=>'tb1'); --加入到已有的亲和组1
SELECT create_distributed_table('tb3', 'productid', colocate_with=>'none'); --新建一个亲和组2
```

亲和分片表的Join

```
postgres=# explain select a.userid,c1,c2 from tb1 a join tb2 b on(a.userid=b.userid);
          QUERY PLAN
```

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

```
Task Count: 8
```

```
Tasks Shown: One of 8
```

```
-> Task
```

```
Node: host=10.244.202.36 port=6432 dbname=sdaspgdb
```

```
-> Merge Join (cost=176.34..303.67 rows=8064 width=68)
```

```
Merge Cond: (a.userid = b.userid)
```

```
-> Sort (cost=88.17..91.35 rows=1270 width=36)
```

```
Sort Key: a.userid
```

```
-> Seq Scan on tb1_151880 a (cost=0.00..22.70 rows=1270 width=36)
```

```
-> Sort (cost=88.17..91.35 rows=1270 width=36)
```

```
Sort Key: b.userid
```

```
-> Seq Scan on tb2_151888 b (cost=0.00..22.70 rows=1270 width=36)
```

```
(13 rows)
```

非亲和分片表的Join

```
postgres=# select a.userid,c1,c3 from tb1 a join tb3 b on(a.userid=b.userid);
ERROR: the query contains a join that requires repartitioning
HINT: Set citus.enable_repartition_joins to on to enable repartitioning
postgres=# set citus.enable_repartition_joins = on;
SET
postgres=# explain select a.userid,c1,c3 from tb1 a join tb3 b on(a.userid=b.userid);
          QUERY PLAN
```

```
-----
Custom Scan (Citus Task-Tracker) (cost=0.00..0.00 rows=0 width=0)
```

```
Task Count: 44
```

```
Tasks Shown: None, not supported for re-partition queries
```

```
-> MapMergeJob
```

```
Map Task Count: 8
```

```
Merge Task Count: 44
```

```
-> MapMergeJob
```

```
Map Task Count: 8
```

```
Merge Task Count: 44
```

```
(9 rows)
```



分布式事务

- Citus对各种跨库操作采用2PC保障事务一致
 - ✓ DDL
 - ✓ Copy
 - ✓ 参考表的更新
 - ✓ 应用发起的跨库事务
 - ✓ 其他
- “Citus Maintenance Daemon”进程自动检测和处理分布式死锁
- “Citus Maintenance Daemon”进程会根据记录在pg_dist_transaction中的事务信息自动清理未决事务

应用发起的跨库事务举例

```
postgres=# set citus.log_remote_commands=on;
SET
postgres=# set client_min_messages = log;
SET
postgres=# begin;
LOG: statement: begin;
BEGIN
postgres=# update tb1 set c1=c1 where userid=1;
LOG: statement: update tb1 set c1=c1 where userid=1;
LOG: issuing BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT assign_distributed_transaction_id(0, 195, '2018-12-14 03:26:47.130076+08');
DETAIL: on server 192.168.1.105:6432
LOG: issuing UPDATE public.tb1_152303 tb1 SET c1 = c1 WHERE (userid OPERATOR(pg_catalog.=) 1)
DETAIL: on server 192.168.1.105:6432
UPDATE 0
postgres=# update tb1 set c1=c1 where userid=10;
LOG: statement: update tb1 set c1=c1 where userid=10;
LOG: issuing BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;SELECT assign_distributed_transaction_id(0, 195, '2018-12-14 03:26:47.130076+08');
DETAIL: on server 192.168.1.104:6432
LOG: issuing UPDATE public.tb1_152313 tb1 SET c1 = c1 WHERE (userid OPERATOR(pg_catalog.=) 10)
DETAIL: on server 192.168.1.104:6432
UPDATE 0
postgres=# commit;
LOG: statement: commit;
LOG: issuing PREPARE TRANSACTION 'citus_0_19380_195_2'
DETAIL: on server 192.168.1.105:6432
LOG: issuing PREPARE TRANSACTION 'citus_0_19380_195_3'
DETAIL: on server 192.168.1.104:6432
LOG: issuing COMMIT PREPARED 'citus_0_19380_195_2'
DETAIL: on server 192.168.1.105:6432
LOG: issuing COMMIT PREPARED 'citus_0_19380_195_3'
DETAIL: on server 192.168.1.104:6432
COMMIT
```

1. 两次提交的间隔，不能保证全局读一致(比如计算sum)。
2. 中途事务失败，“Citus Maintenance Daemon”进程会自动恢复事务，确保数据一致。



SQL限制

■ 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

■ 其他限制

- ✓ 不支持按非分片字段分组的Window函数
- ✓ 本地表不能和分片表(参考表)混用
- ✓ 'insert into ... select ... from'不支持分片表->本地表

■ 回避方法

- ✓ 通过临时表(或dblink)中转
- ✓ CTE或子查询(会把子查询结果分发到各worker, 注意性能)



目录

- Citus的引入
- Citus的工作方式
- 表的设计
- 部署与维护
- 典型案例
- 常见问题

分片表/参考表/本地表

表类型	适用场景	注意点
分片表	大表，访问频繁的表	SQL限制，分片数，表亲和
参考表	需要和分片表join的表，小表，更新少的表	存储冗余 更新参考表存在写放大，并产生分布式事务。 读参考表，负载固定落在第一个worker。
本地表	需要支持复杂SQL，小表	负载集中在CN节点 部分SQL不支持本地表和分片表/参考表混用 不支持多CN(Citus MX)访问



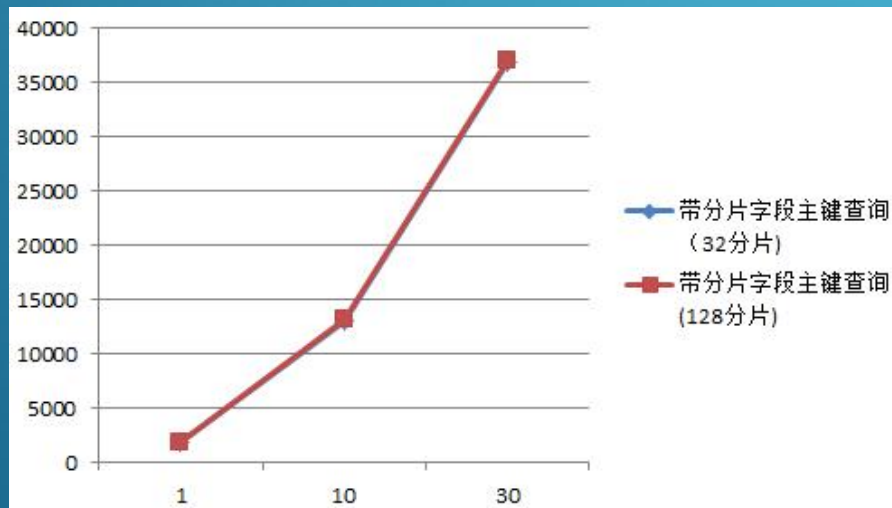
分片字段

- 每个SQL中都带等值条件的字段（比如用户ID）
- Join关联的字段
- 高基数且值分布均匀的字段
- 日期通常不适合作为分片字段，容易形成热点

分片数

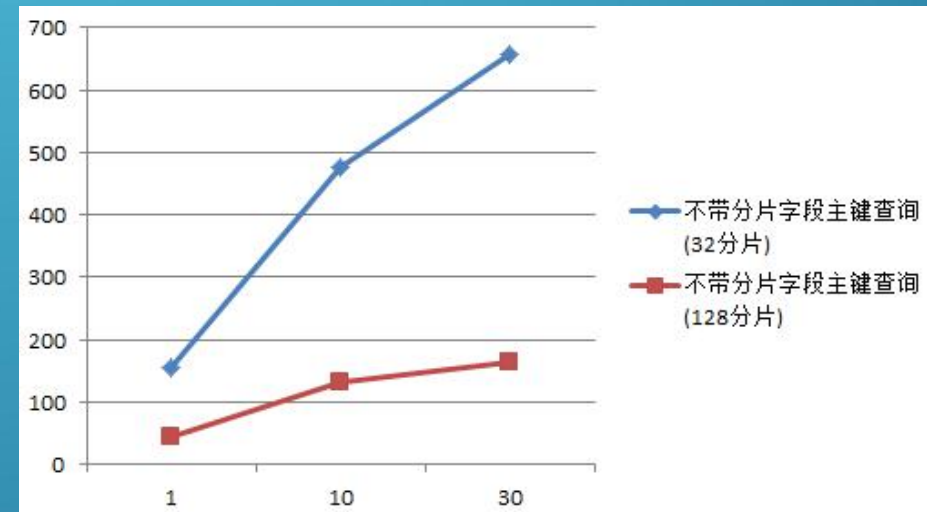
- 官方建议OLTP场景(带分片字段SQL)小负载（小于100GB）32；大负载64或128；
- OLAP场景(不带分片字段SQL)为worker总core数的2倍或4倍
- 不带分片字段SQL，分片数越多CN负载越大；CN是瓶颈时(主键查询，分组结果很多的聚合查询)QPS越低

主键查询QPS



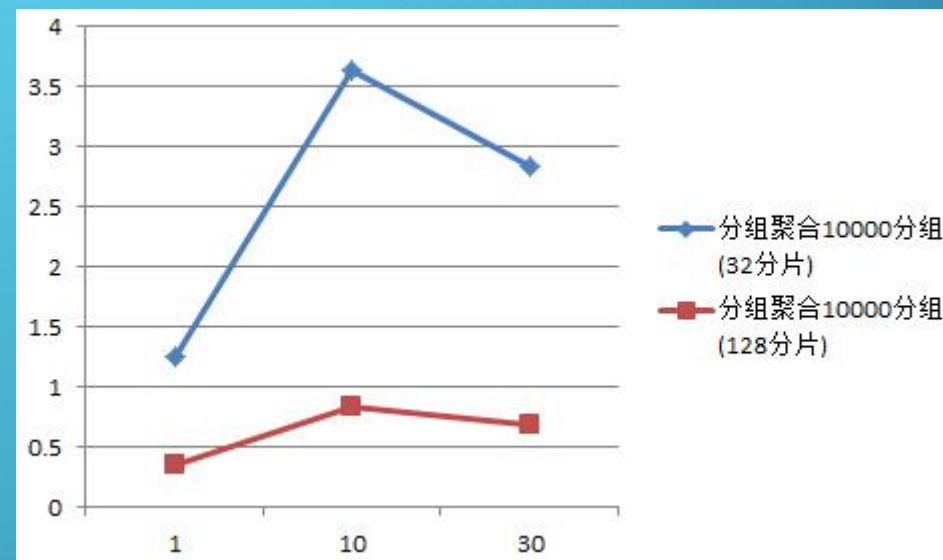
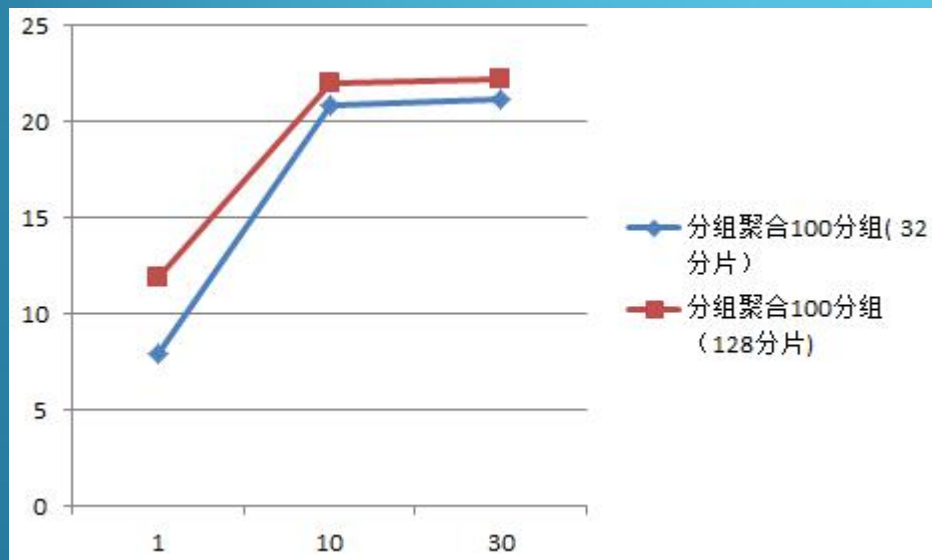
`select * from test_tb32 where id=:num1 and c2=:num2; //c2是分片字段`

备注:16C/32G SSD虚拟机, 1亿记录



`select * from test_tb32 where id=:num1;`

分组聚合查询QPS



```
select c3,count(*) c from test_tb32 where c1 between 5000001 and 10000000 group by c3 order by c limit 10;  
select c3,c4,count(*) c from test_tb32 where c1 between 5000001 and 10000000 group by c3,c4 order by c limit 10;  
//c3和c4的基数都是100
```


表亲和

- 缺省所有分片数，副本数，分片字段类型相同的分片表被归属为同一个亲和组

```
select create_distributed_table('tb','id');
```

- 自定义亲和组可以减小今后分片迁移的粒度

```
select create_distributed_table('tb1','id', colocate_with=>'none'); --生成新的亲和组  
select create_distributed_table('tb2','id', colocate_with=>'tb1'); --加入已有的亲和组
```

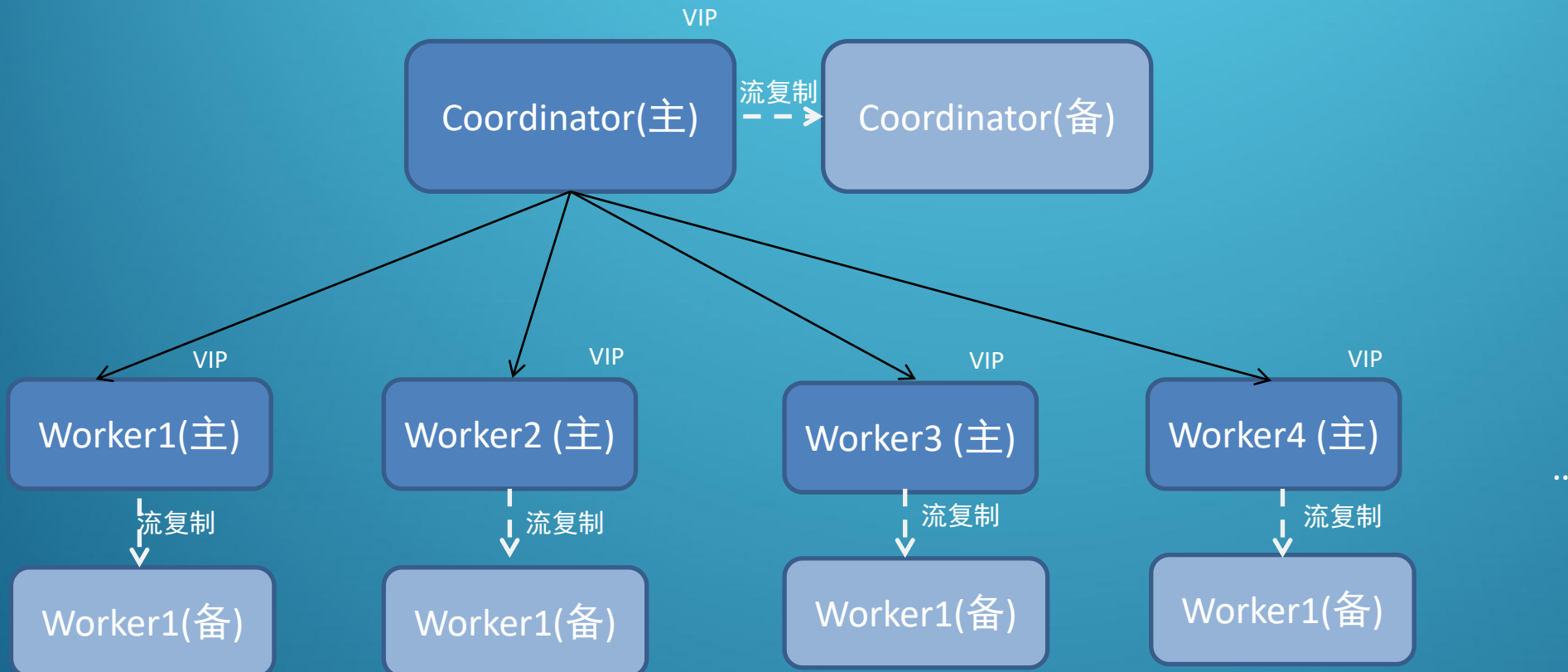


目录

- Citus的引入
- Citus的工作方式
- 表的设计
- 部署与维护**
- 典型案例
- 常见问题

高可用、备份恢复、监控告警

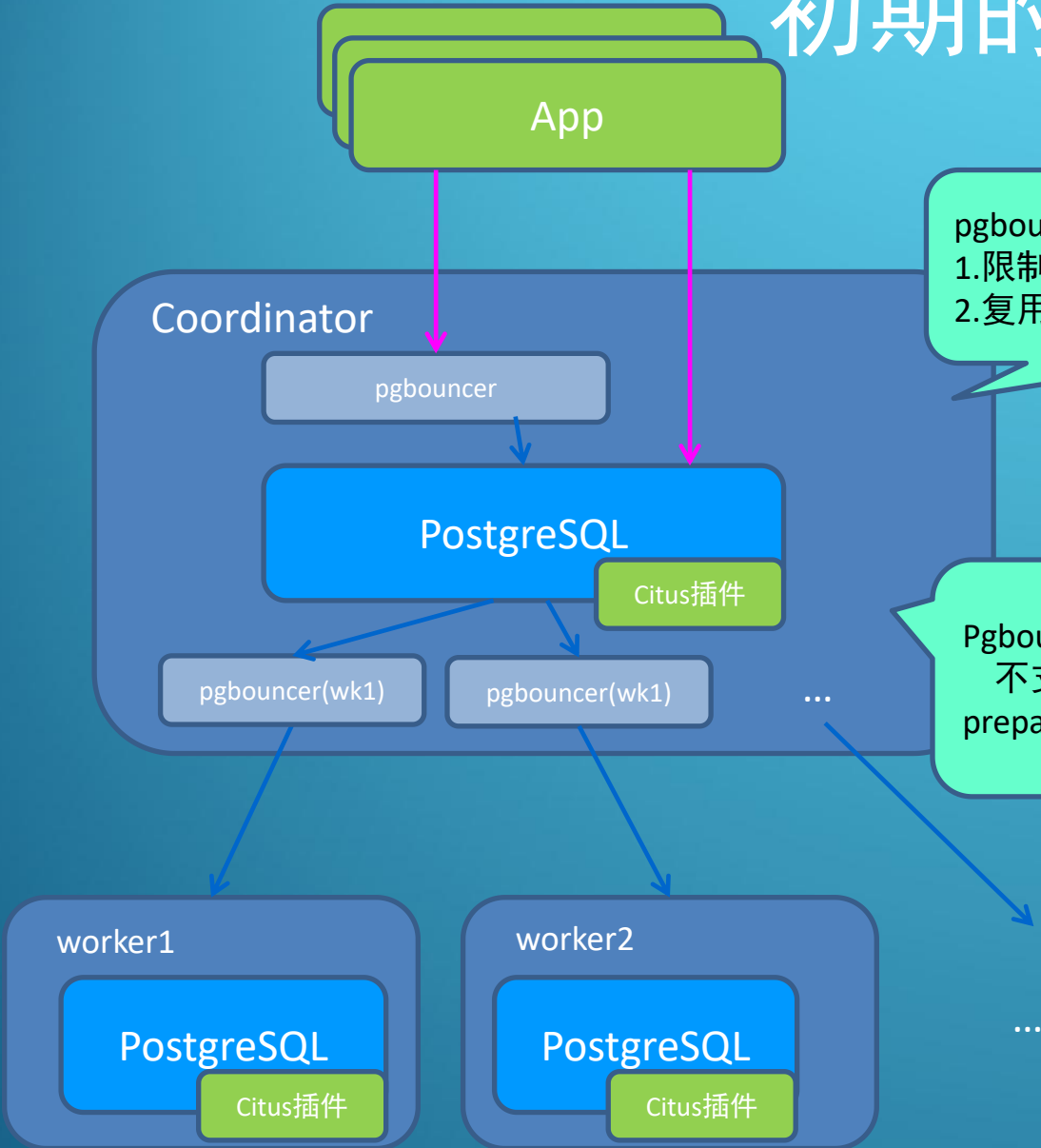
- CN和每个Worker可以作为单独的PG HA集群维护
- 不支持全局一致性备份



注:Citus还支持多副本的HA方案, 但只适合Append-Only场景



初期的部署架构(含连接池)



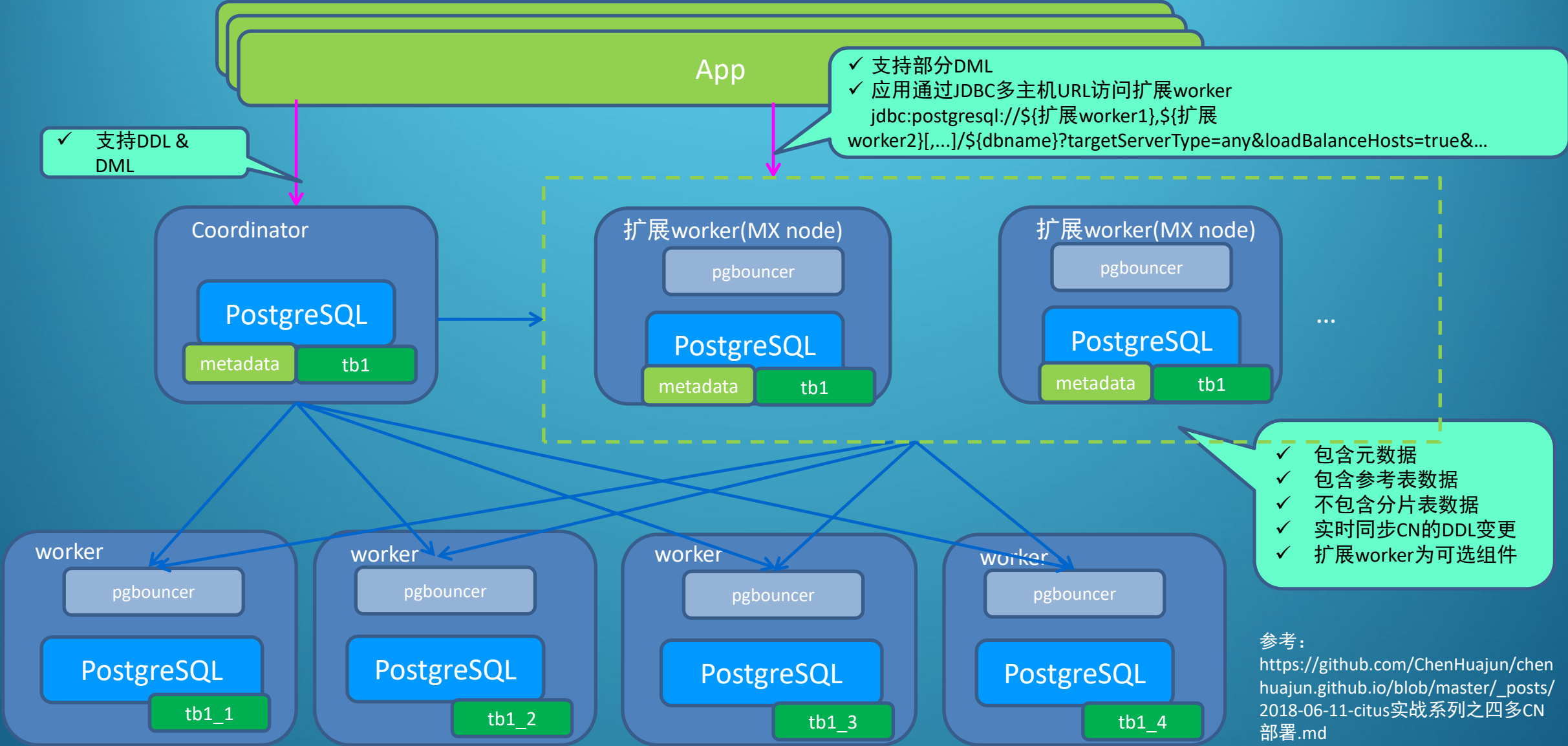
pgbouncer的作用:
 1. 限制CN发起的到worker连接数
 2. 复用worker连接

Pgbouncer(事务池)的限制:
 不支持预编译语句, JDBC中需要设置
 prepareThreshold=0回避

存在问题
 1. 单CN瓶颈
 2. Pgbouncer部署在CN上阻碍分片间通信(分片修复和分片迁移)



新的多CN部署架构



不同连接池部署方式的性能

环境:8C/16G/500G虚拟机, 1CN+2Worker

场景	App->CN	CN->Worker	并发数	QPS
非分片字段查询 (real-time执行器)	直连	直连	8	18
	直连	pgb(CN节点)	30	112
	直连	pgb(Worker节点)	30	97
	pgb	pgb(CN节点)	30	98
	pgb	pgb(Worker节点)	30	94
分片字段查询 (route执行器)	直连	直连	600	24903
	直连	pgb(CN节点)	600	12115
	直连	pgb(Worker节点)	600	16940
	pgb	pgb(CN节点)	100	9051
	pgb	pgb(Worker节点)	100	10738

根据以上测试结果, 连接池部署在Worker上性能更好, 并且应用应尽量直连CN节点。

DDL支持

DDL	是否支持	备注
ALTER TABLE	部分支持(注1)	包含扩展Worker上的逻辑表
DROP TABLE	支持	包含扩展Worker上的逻辑表
CREATE INDEX	支持	包含扩展Worker上的逻辑表
DROP INDEX	支持	包含扩展Worker上的逻辑表
ALTER INDEX	支持	不会操作扩展Worker上的逻辑表。
ANALYZE	支持	不会操作扩展Worker上的逻辑表。
VACUUM	支持	不会操作扩展Worker上的逻辑表。
LOCK	部分支持	仅在CN本地LOCK逻辑表,应用通过扩展Worker仍可以访问表
CLUSTER	不支持	仅在CN本地CLUSTER逻辑表

注1:ALTER TABLE支持的子命令:

ADD|DROP COLUMN, SET|DROP NOT NULL, SET|DROP DEFAULT,
ADD|DROP CONSTRAINT, SET (), RESET (), ATTACH|DETACH PARTITION and TYPE

注2:基于Citus 7.4

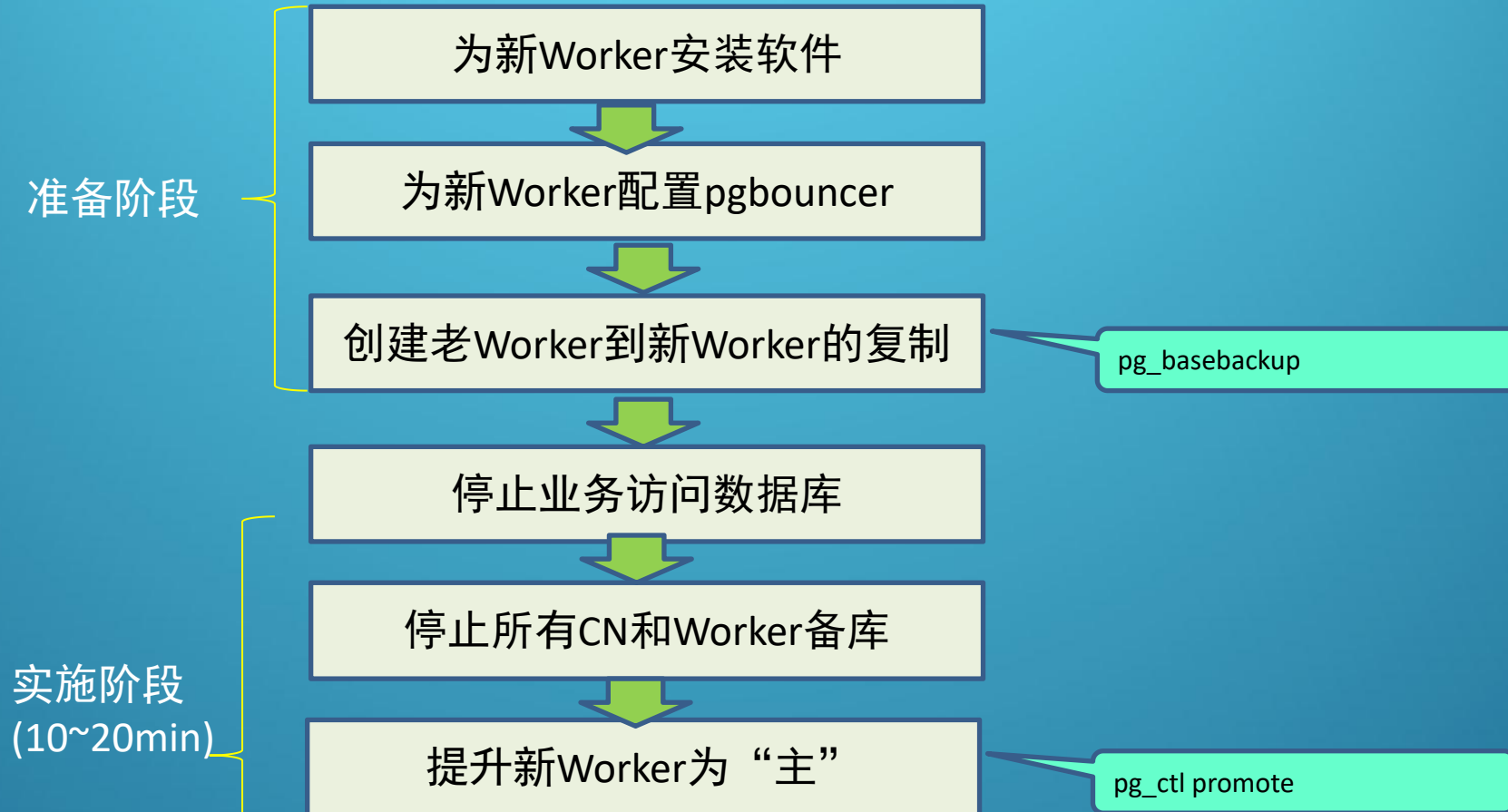
- 不支持子命令的回避方法
- ✓ 创建对等价的唯一索引代替变更主键
- ✓ 通过`run_command_on_placements`函数, 直接在所有分片位置上执行DDL

扩容（基于物理复制）

■ 适用条件

- ✓ 当前集群中WK的数量必须是2的N次方
- ✓ 扩容后的WK数量是扩容前的2倍

■ 扩容步骤





创建测试表

```
create table test_colocation_before(c1 int primary key,c2 int);
create table test_ref(c1 int primary key,c2 int);
select create_distributed_table('test_colocation_before','c1');
select create_reference_table('test_ref');
insert into test_colocation_before select id,id from generate_series(1,10) id;
insert into test_ref select id,id from generate_series(1,10) id;
```

添加新Worker到Citus集群

```
select master_add_node($worker_ip,$port);
select master_add_node($worker_ip,$port);
...
```

修改元数据

```
update pg_dist_shard_placement ... -- 更新分片表已迁移到新worker的shard位置
insert into pg_dist_shard_placement ... -- 插入参考表在新worker上的shard位置
```

测试验证

```
create table test_colocation_after(c1 int primary key,c2 int);
select create_distributed_table('test_colocation_after','c1');
insert into test_colocation_after select id,id from generate_series(1,10) id;
```

删除测试表

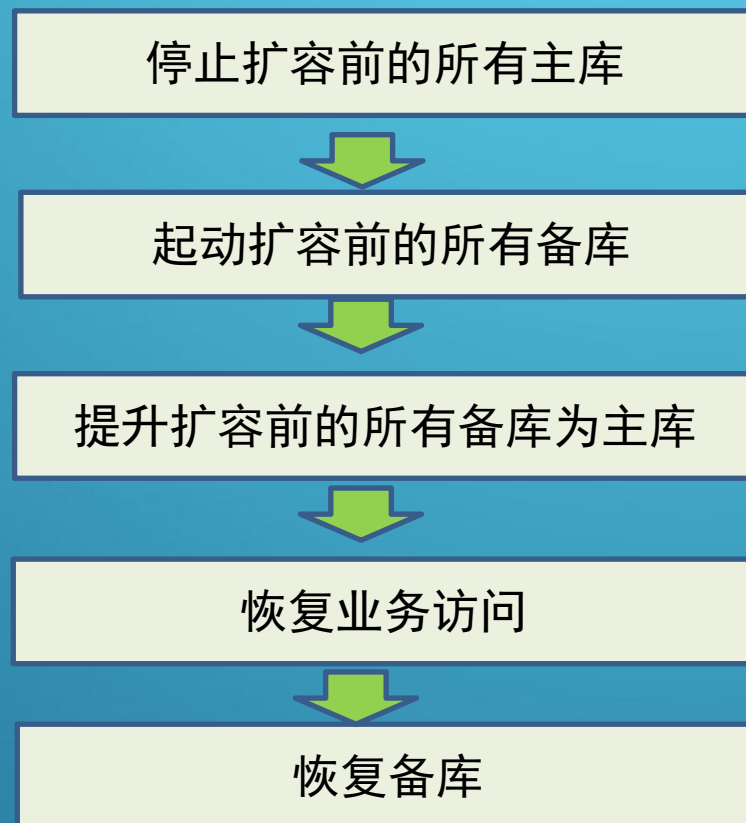
```
#select * from test_colocation_before a,test_colocation_after b where a.c1 = b.c1
and a.c1 = 1;
c1 | c2 | c1 | c2
----+----+----+----
 1 | 1 | 1 | 1
(1 rows)

#select * from test_colocation_after a,test_ref b where a.c1 = b.c1 and a.c1 = 1;
c1 | c2 | c1 | c2
----+----+----+----
 1 | 1 | 1 | 1
(1 rows)
```

恢复业务读写

DROP新老Worker上的冗余分片

■ 回退步骤





扩容（基于逻辑复制）

■ 适用条件

- ✓ 新Citus部署架构（pgbouncer在Worker上）
- ✓ 新Worker账号密码和现有Citus集群一致

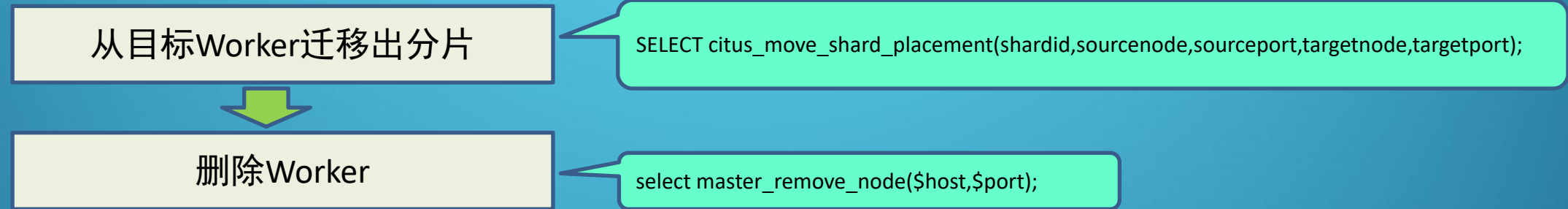
■ 普通Worker扩容步骤



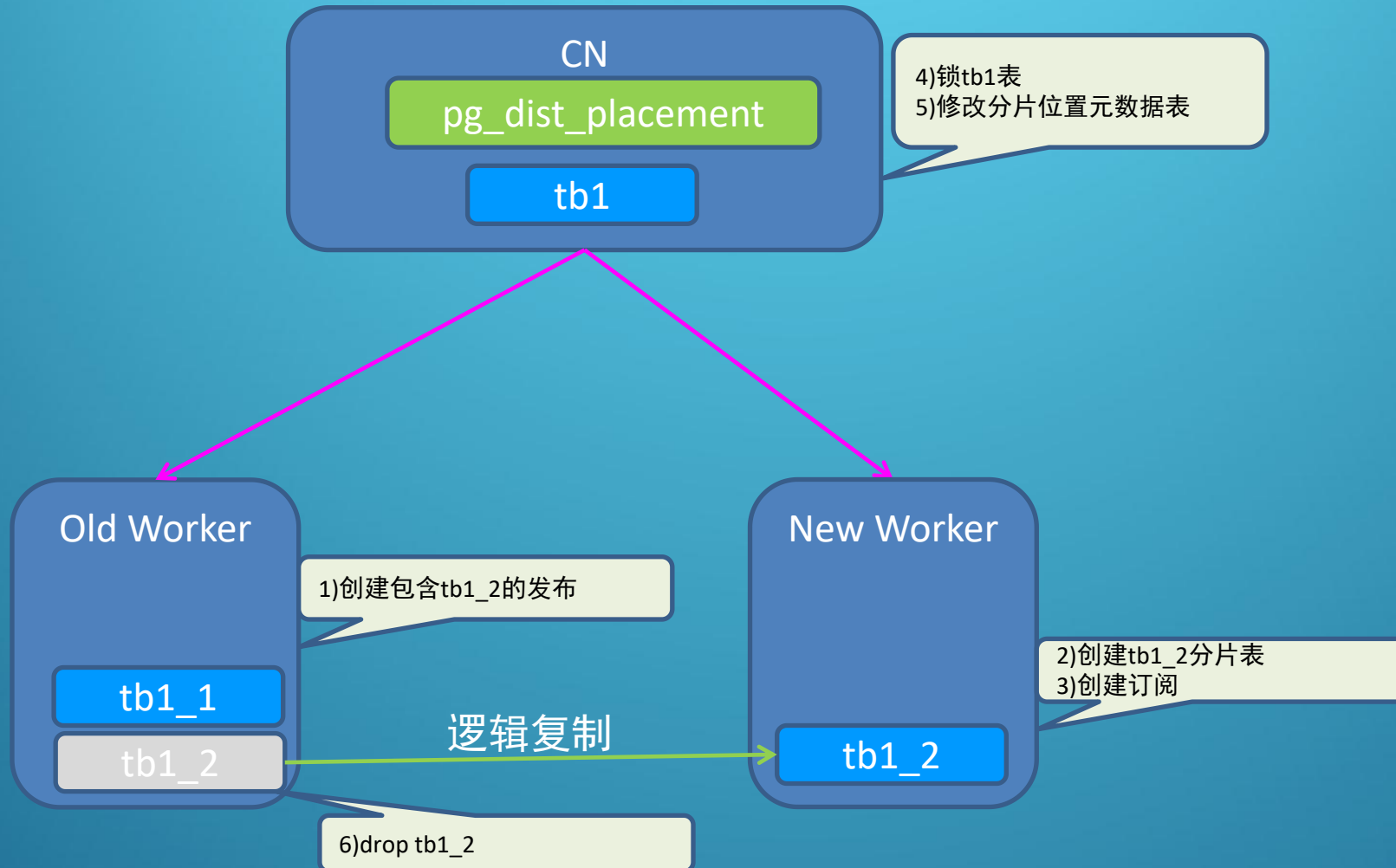
■ 扩展Worker(MX Node)扩容步骤



■ 回退（扩容）步骤



■ 分片迁移





目录

- Citus的引入
- Citus的工作方式
- 表的设计
- 部署与维护
- 典型案例**
- 常见问题



某数据分析系统

■ 业务场景

- ✓ 之前使用50多个DB2分库，每个库大约1TB，需要迁移到开源DB
- ✓ 最大单表200亿记录
- ✓ 每日并发导入大量数据（先大批量删再大批量插）
- ✓ 主要执行带分片字段的分析查询，QPS要求100以上

■ 部署方案

- ✓ 1CN + 32 Worker + 4扩展Worker
- ✓ 分片数512(比128分片qps提升1倍)

某计费结算系统

■ 业务场景

- ✓ 每天处理单据6000w
- ✓ 对单据进行多次业务处理后最终生成1w+对账单
- ✓ 最大单表记录数70亿
- ✓ 主要业务逻辑包含在事务中，且大量使用跨库的分布式事务

■ 部署方案

- ✓ 业务拆分为8套Citus集群(每套1CN + 8Worker)



某经营分析系统

■ 业务场景

- ✓ 按照不同的维度组合(品类、品牌、大区、门店...)分析会员的新老买家类、留存率类、复购类以及沉睡类指标等
- ✓ 每个维度组合的维度值数量多的有几十万
- ✓ 每个维度值涉及的用户数多的有数百万
- ✓ 用户总数数亿
- ✓ 计算纯新买家，次新买家需要和所有历史买家(亿级别)做集合运算

■ 技术方案

- ✓ 使用开源插件pg_roaringbitmap存储用户集合
- ✓ 1CN + 8扩展Worker部署(按计算维度重新分布数据)

■ 使用效果

- ✓ 任意维度组合的客户留存指标计算秒级响应



目录

- Citus的引入
- Citus的工作方式
- 表的设计
- 部署与维护
- 典型案例
- 常见问题

并发控制

- Citus在执行DML时会针对每个分片额外获取advisory锁，objid为分片号，实施附加的并发控制

表类型	操作	获取的锁类型	objsubid	允许的并发操作
分片表	INSERT 带分片字段UPDATE和DELETE	ShareLock	4	INSERT/UPDATE/DELETE
	INSERT ... ON CONFLICT 不带带分片字段UPDATE和DELETE	ShareLock	4	INSERT 带分片字段UPDATE和DELETE
		ShareUpdateExclusiveLock	5	
参考表	INSERT	ShareLock	4	INSERT
		RowExclusiveLock	5	
	INSERT ... ON CONFLICT UPDATE/DELETE	ShareLock	4	无
		ExclusiveLock	5	

注1：SELECT不获取advisory锁，和以上操作都不冲突

注2：DDL会获取objsubid为4的ShareLock类型的advisory锁，和以上操作的advisory锁不冲突，主要靠PG自身的表级锁进行并发控制。

注3：以上结果基于Citus 7.4

分布式事务

- 事务结束前一直占用pgbouncer的连接（事务池），容易耗尽连接

对策：

- ✓ 配置比较大的default_pool_size
- ✓ 把事务拆小
- ✓ 避免不带分片字段的SQL

- 不支持某些事务使用场景

对策：

- ✓ 避免访问多个分片的SQL
- ✓ 把访问多个分片的SQL放到前面，或事务开始时提前做一次跨多分片的查询，预先创建好需要连接

```
begin;  
update tb1 set id=id where id=1; -- id为分片字段。为该shard创建连接1  
update tb1 set id=id where id=3; -- id值为1和3的2条记录位于同一worker的2个不同的shard上。复用连接1  
select * from tb1 where id in(1,3); -- 需要2个连接分别在2个shard上执行SQL，但新建的连接无法看到前面未提交的变更，报错。
```

ERROR: cannot establish a new connection for placement 11440, since DML has been executed on a connection that is in use



数据导入

■ INSERT

- ✓ 批量插入的数据被CN拆成单条非预编译INSERT发送给Worker
- ✓ 单个insert中如有多个value，属于同一分片的值仍以多value的形式发送给worker
- ✓ JDBC的批更新和rewriteBatchedInserts=true优化参数可提升插入性能，但整体上INSERT的性能偏低

■ COPY

- ✓ 典型业务表Copy导入速度：大约10w/s(速度依赖部署环境和表定义)
- ✓ 索引较多的大表导入速度慢。每个Worker上的所有分片同时导入，容易导致缓存不足。可以通过分区进行优化。

■ 官方数据

- ✓ Insert and Update:10-50k/s
- ✓ Bulk Copy:250k/s-2M/s
- ✓ Citus MX:50k/s-500k/s



数据导出

- 大量数据的导出任务，长时间消耗CPU/IO和占用PgBouncer连接资源，可能影响其他业务。
对策：

- ✓ 使用task-tracker执行器，限制每个worker上的最大并发数(默认8)

```
SET citus.task_executer_type = 'task-tracker';
```

real_time执行器

执行不带分片字段SQL时，CN对所有worker上的所有shard同时发起连接，并执行SQL收集结果。

task-tracker执行器

执行不带分片字段SQL时，CN只和worker上的`task-tracker`进程交互，调用worker上的`task_tracker_assign_task()`函数将任务分配给`task-tracker`，然后轮询任务的完成状况，待任务结束后再开一个连接从worker取回结果。

项目	real_time	task-tracker
单个不带分片字段的SQL在每个worker上产生的连接数	该worker上分片数	2个
每个worker上最大并发执行数	应用并发数*worker上分片数	8(citus.max_running_tasks_per_node)
单个SQL的响应时间(2个worker, 4个分片)	40ms	600ms
单个SQL的响应时间(8个worker, 128个分片)	53ms	1627ms

参考表

- 应用读参考表时始终访问第一个分片，造成负载不均衡。

对策：

- ✓ 创建一个空的分片表，和参考表Join，利用分片表分散负载(副作用是增加CN负载)。
- ✓ 升级到8.0.1及以后版本。Citus 8.0.1已支持参考表的轮询，<https://github.com/citusdata/citus/pull/2472>

- 参考表建索引时串行依次对每个副本创建索引，导致大的参考表建索引非常慢

对策：

- ✓ 使用run_command_on_placements()函数并行创建索引

```
SELECT * FROM run_command_on_placements('tb1', 'create index on %s (id,c1...)');
```

注意：

1. CN的逻辑表上没有索引，和worker的表定义不一致。
2. 今后删除索引也要使用run_command_on_placements()函数
3. 如非必要不建议采用这种非标准方式建索引



踩到的Bug

组件	描述	状态	修复版本	参考
Citus	Citus跨库事务导致内存泄漏	已修复	8.0.0 7.5.2	https://github.com/citusdata/citus/issues/2405
	Citus跨库事务失败导致Connection ID泄漏无法执行跨库SQL	未修复		https://github.com/citusdata/citus/issues/2526
	CN节点启用auto_explain有可能导致Citus执行SQL失败	未修复		https://github.com/citusdata/citus/issues/2009
PostgreSQL	Gin索引autovacuum和insert死锁导致连接Hang	已修复	未发布	https://www.postgresql.org/message-id/31a702a.14dd.166c1366ac1.Coremail.c_hjischj%40163.com
	Gin索引delete日志记录的回放和select死锁导致连接Hang	已修复	未发布	同上
	Gin索引和堆文件收缩水位导致访问到不存在的段文件	未修复		https://www.postgresql.org/message-id/15532-bdaabfecf52cdbf1@postgresql.org

Thanks

