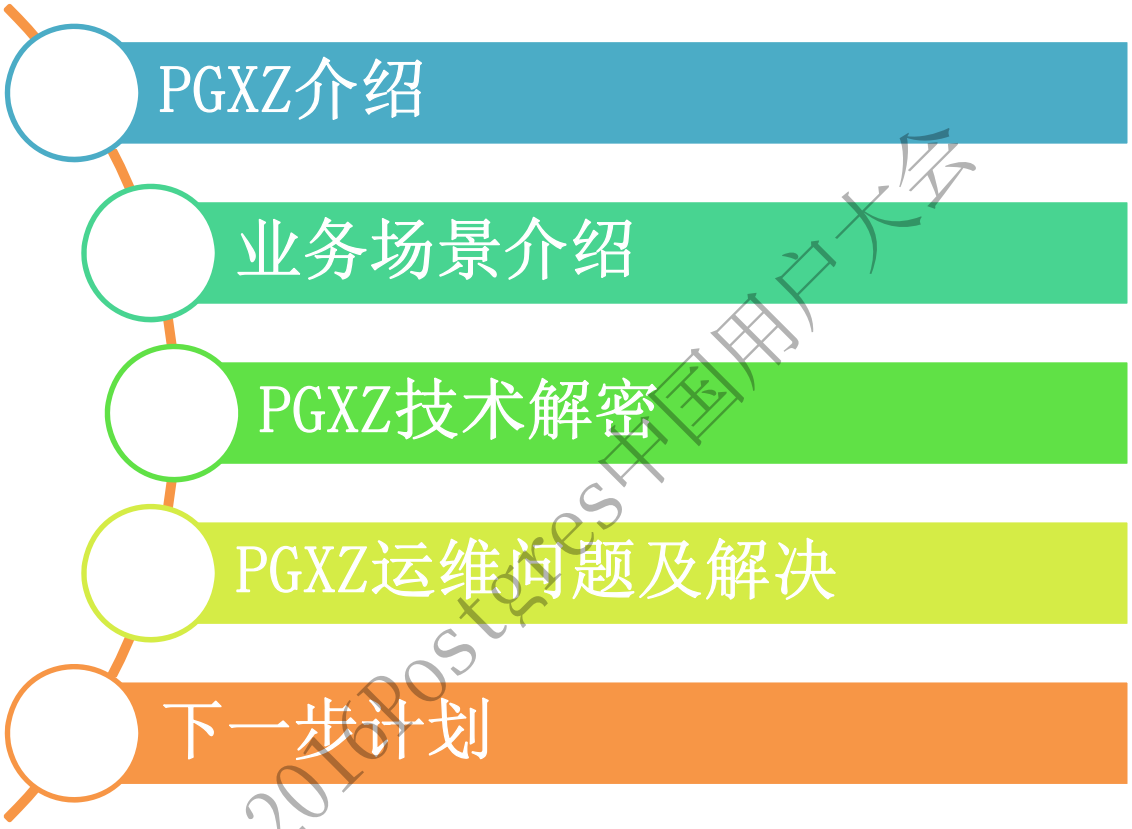
PGXZ在微信支付中的应用

李跃森

腾讯科技



PGXZ溯源—PGXC介绍

- 由日本NTT和美国EnterpriseDB公司联合发起的开源分布式数据库项目，最新版本为1.2.2。
- PGXC特点:
  - ✓ 基于postgresql的share nothing的MPP集群解决方案，对外提供一个统一的数据库视图
  - ✓ 客户端完全兼容单机PG
  - ✓ 支持数据replicated、hash、round robin分布方式
  - ✓ 处理能力和集群节点数正比线性提升
  - ✓ 支持hot standby，故障切换零数据丢失，一主多备高可靠性保护数据防丢失
  - ✓ 单节点点查询TPS达到20000，并可线性扩展

PGXZ技术愿景：

- 高效
- 易用
- 稳定
- 可靠

PGXZ VS PGXC

性能

- 开发集群分区表支持，大幅提升系统的性能
- 开发并行执行引擎，同一个SQL可以同时多个CORE并行，大幅提高执行效率
- 优化网络缓冲区，提升网络处理效率

功能

- 提供PGXZ数据库集群数据倾斜解决方案，帮助业务轻松应对数据倾斜问题
- 提供冷热数据分离的解决方案，帮助业务减轻成本压力

运维能力

- 开发PGXZ的分布式运维管理系统，提升运维管理的效率
- 开发PGXZ的两地三中心功能支持，满足数据安全和数据一致性要求
- 开发系统在线扩容功能，新增shard分布表，在不影响业务的情况下在线扩容

PGXZ在微信支付中的位置

Payment bills
Funds flow
Billing flow

UPDATE/INSERT

POSTGRES XC

SELECT

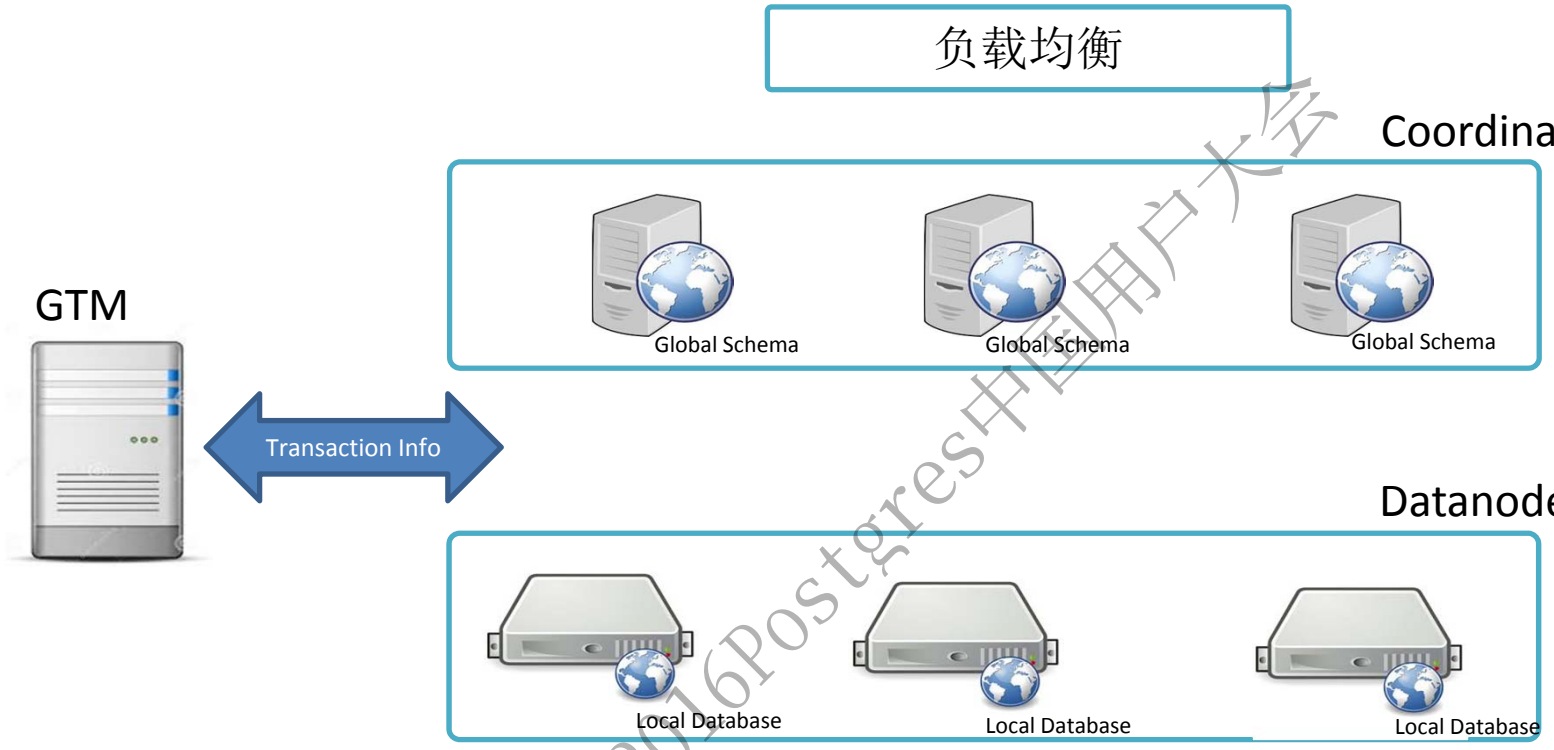
JD.COM 京东

美团 meituan.com

TAXI

Other

PGXZ在微信中的部署方式



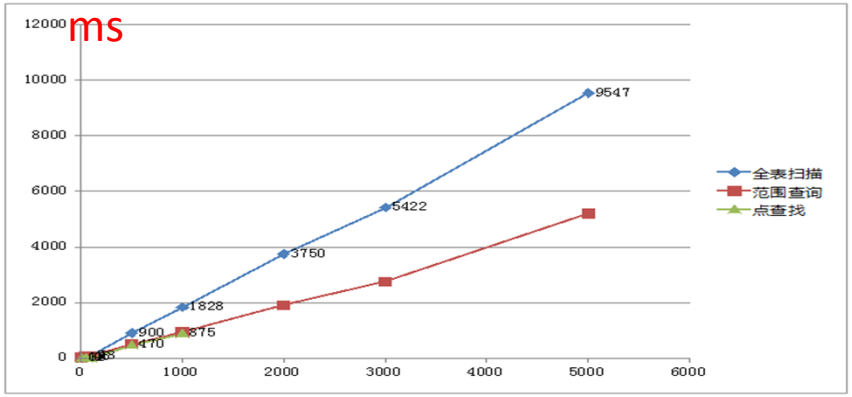
5台
8CORE，16G内存
SATA硬盘

24组(每组主备两台) 32CORE，64G内存
FUSION IO CARD

PGXZ技术解密—集群分区表，为什么不用PG原生的分区表？

SELECT

SELECT生成执行计划的时间 与 子表数量成线性关系

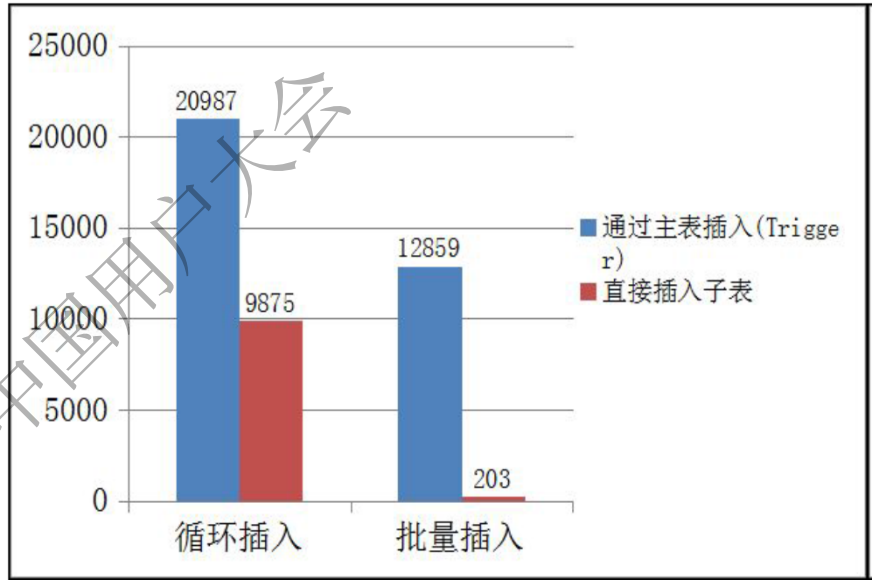


子表数量	5	50	100	500	1k	2k	3k	5k
全表扫描	16	46	78	900	1,828	3,750	5,422	9,547
范围查询	16	47	78	500	937	1,900	2,750	5,203
点查询				470	875	1,875	2,687	4,797

原因：
❶对主表的查询会展开成N+1条类似的查询语句。(N为主表的子表数量)
❷相当于要把同一条SQL语句(主表替换成子表)优化N+1遍

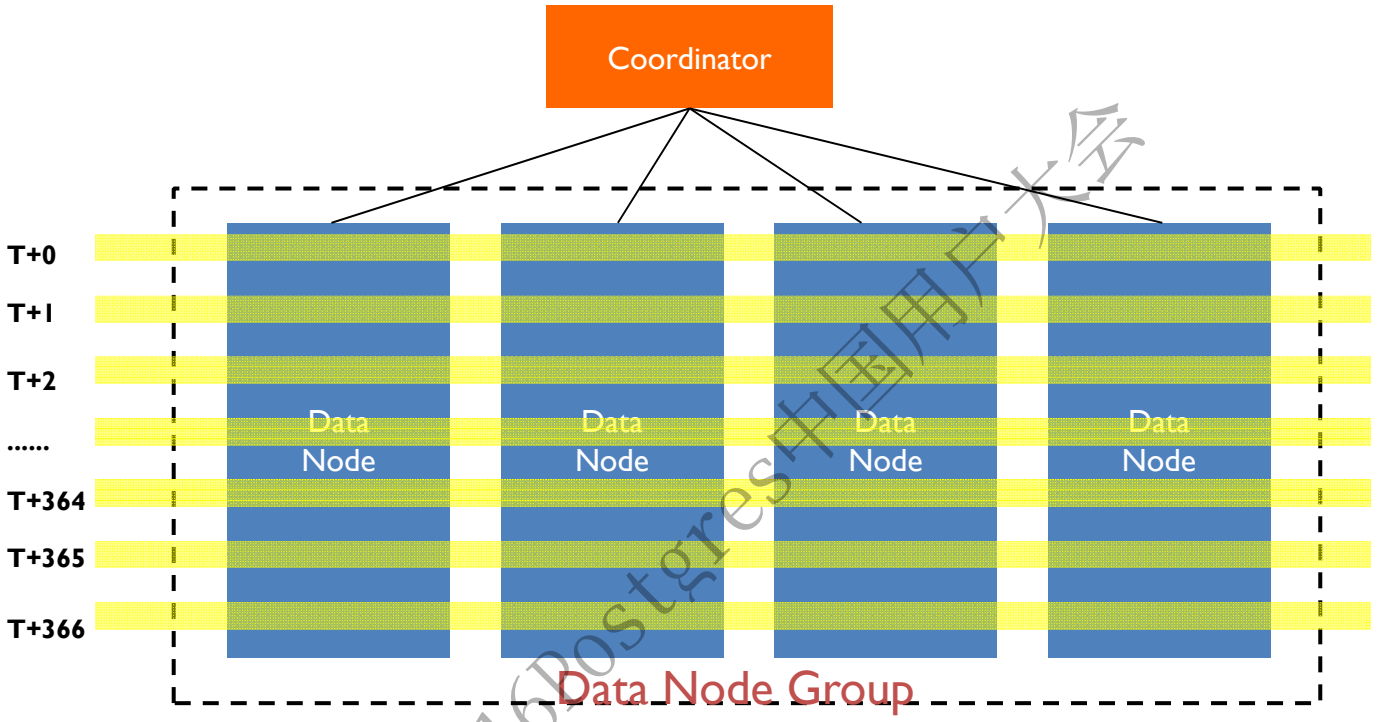
INSERT

在批量插入的场景下，trigger对性能影响很大



❶测试环境：5个子表，插入10K条数据，插入的所有数据都在一个分区内每个tuple的大小16 byte

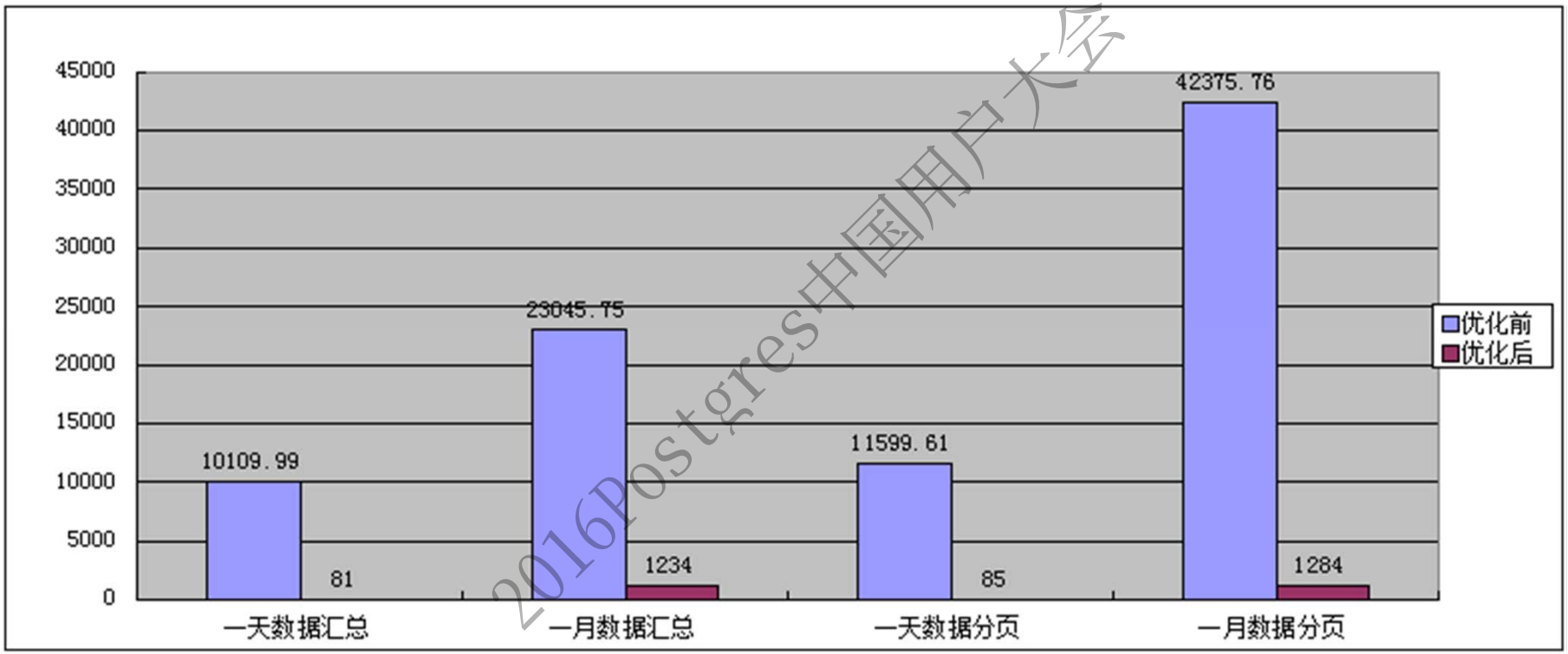
PGXZ技术解密—集群分区表



Coordinator: 负责纵向分表，不感知表分区(横向分表)的逻辑, 看到DataNode上就是一张逻辑表。
DataNode：负责横向分表，根据分表字段将一张逻辑表划分为多张物理表，承载分区表的主要实现逻辑

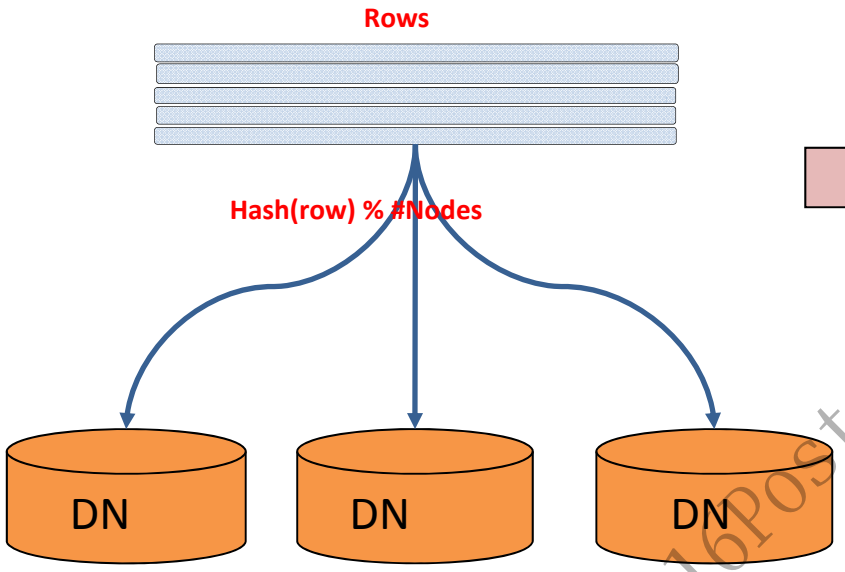
PGXZ技术解密—集群分区表性能结果

性能对比：优化后(集群分区表)比优化前(继承表)在业务场景下提升1~2个数量级



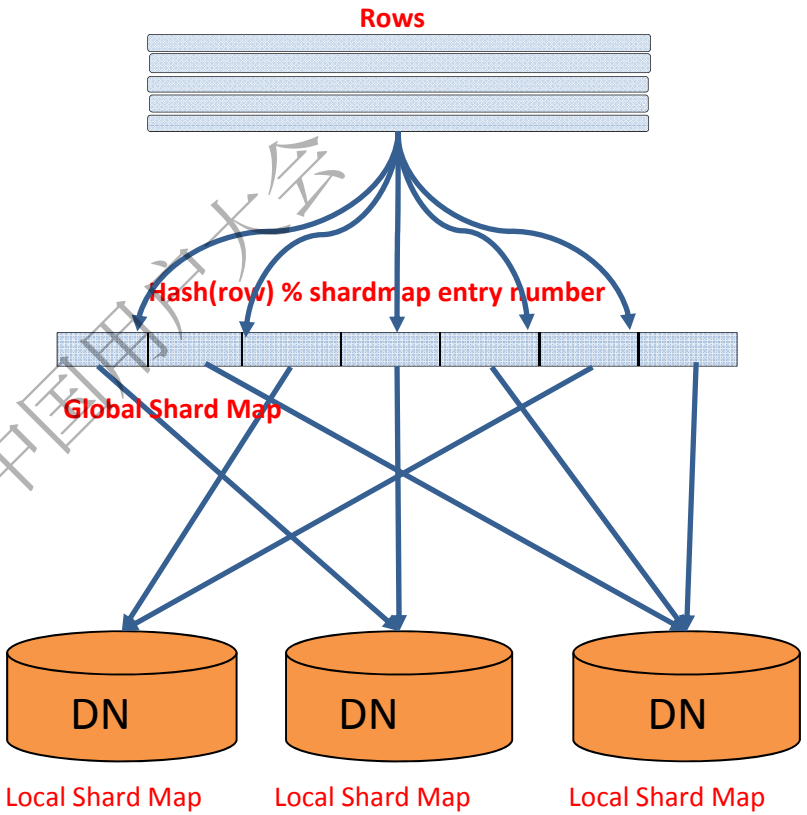
PGXZ技术解密—在线缩容扩容能力

PGXC中的hash表



记录的存储节点位置计算公式：
DN = Hash(row) % nofdn

PGXZ Shard表的数据分布

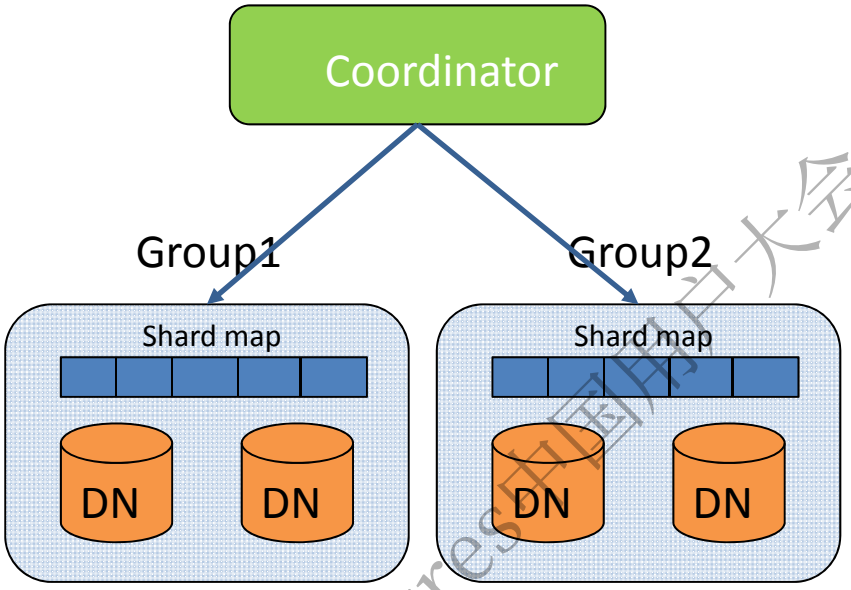


- 引入shardmap表，建立hash值和存储节点位置的映射.
- 数据分布算法只决定数据存储在哪个shardid中，shardmap中记录shardid对应的物理节点.

PGXZ技术解密—为什么需要解决数据倾斜问题？

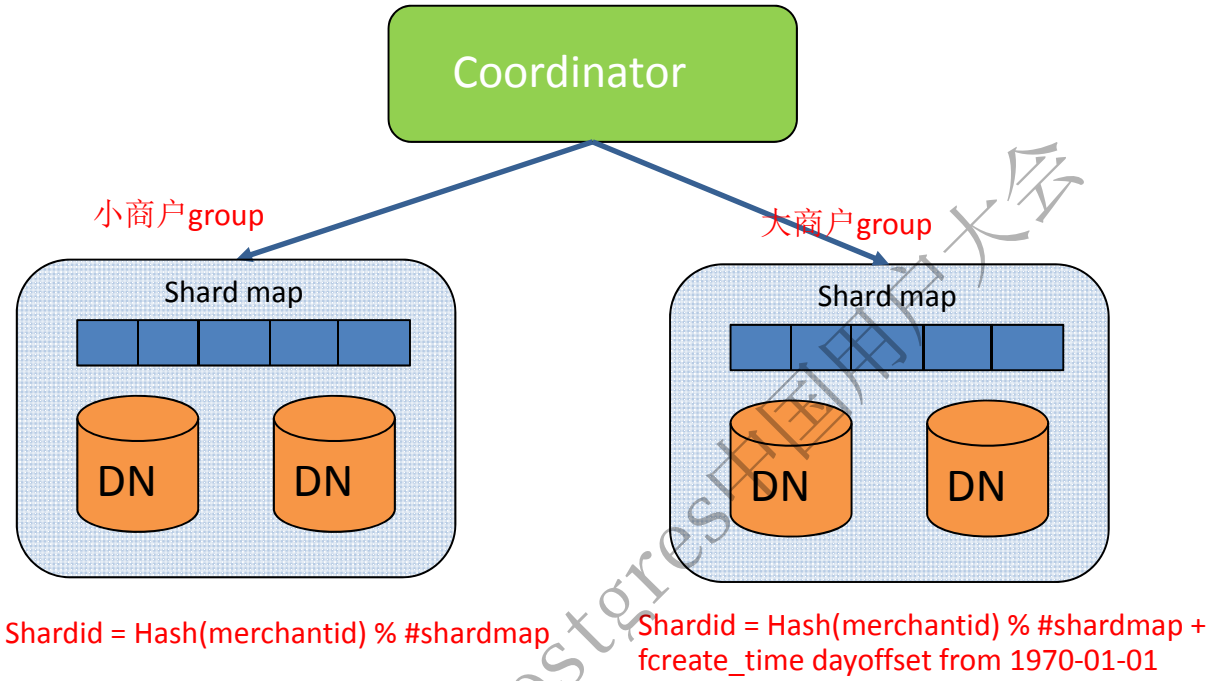
- 商户数据量存在差异，大商户每天数据量超过30G
- 使用hash进行分布时，每个商户的数据只能落到一个物理节点。
- 单个物理节点的磁盘空间有限，大商户会加速磁盘空间的消耗，对同节点的其他商户造成影响。

PGXZ技术解密—数据倾斜解决方案，数据节点组(GROUP)介绍



- 一个节点组可以有一个或者多个数据节点
- 每个节点组有自己独立的shardmap，也就是说每个节点组可以独立的扩容缩容.
- 一张表的数据，可以根据白名单存储在不同的节点组中，不同的节点组可以采用不同的数据分布策略.
- 协调节点可以访问所有的节点组.

PGXZ技术解密—数据倾斜解决方案，数据分布策略

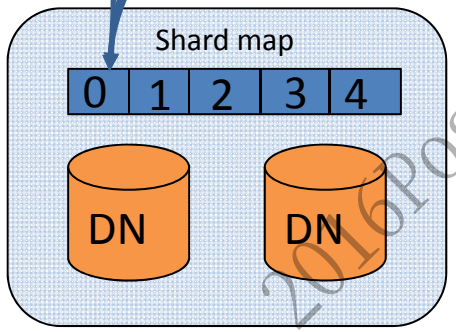


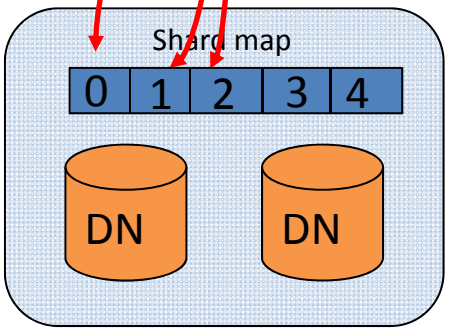
- 根据统计信息确定哪些商户是大商户。
- 普通的商户使用普通的shard表分布逻辑。
- 对大商户建立白名单，白名单中的商户使用特殊的数据分布逻辑：引入交易发生的日期作为平衡因子把同一个商户的数据按照天存储到不同的shardid。

PGXZ技术解密—数据倾斜解决方案，举例：

Merchant_id	Fcreate_time
Small1	2016-04-07 12:35:42
Small1	2016-04-08 11:35:42
Small1	2016-04-09 13:35:42

Merchant_id	Fcreate_time
Huge1	2016-04-07 11:35:42
Huge1	2016-04-08 11:35:42
Huge1	2016-04-09 11:35:42





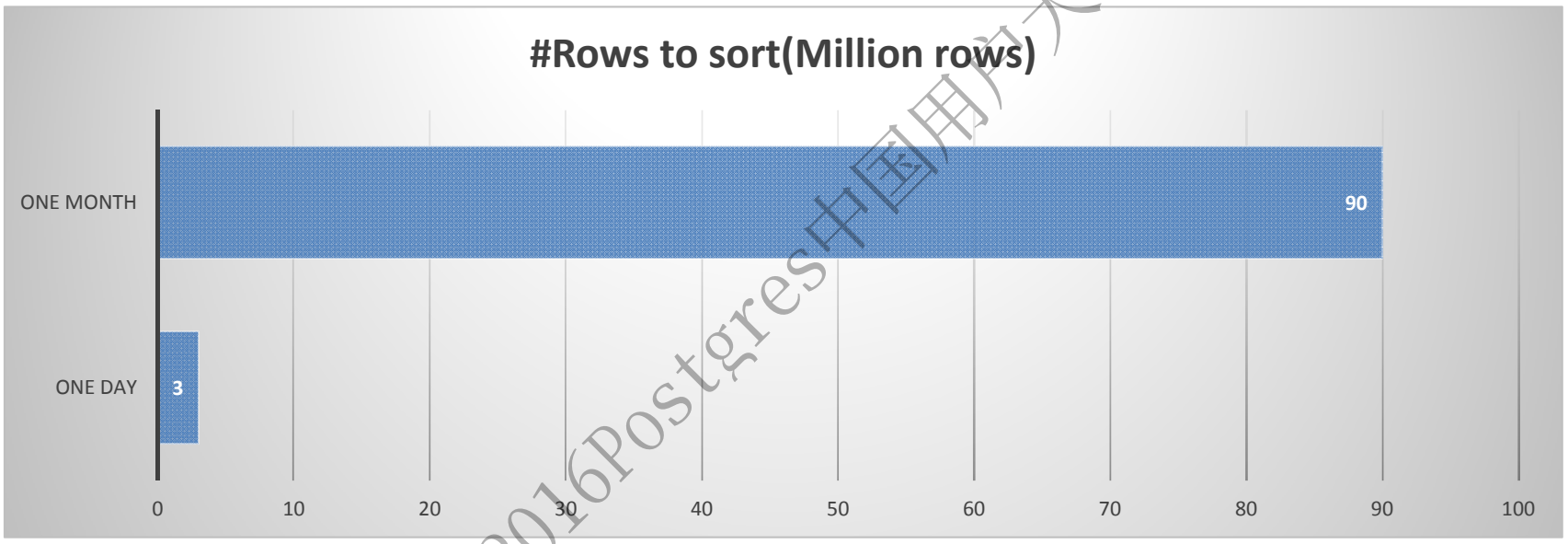
Shardid = Hash(merchantid) % #shardmap

Shardid = Hash(merchantid) % #shardmap +
fcreate_time dayoffset from 1970-01-01

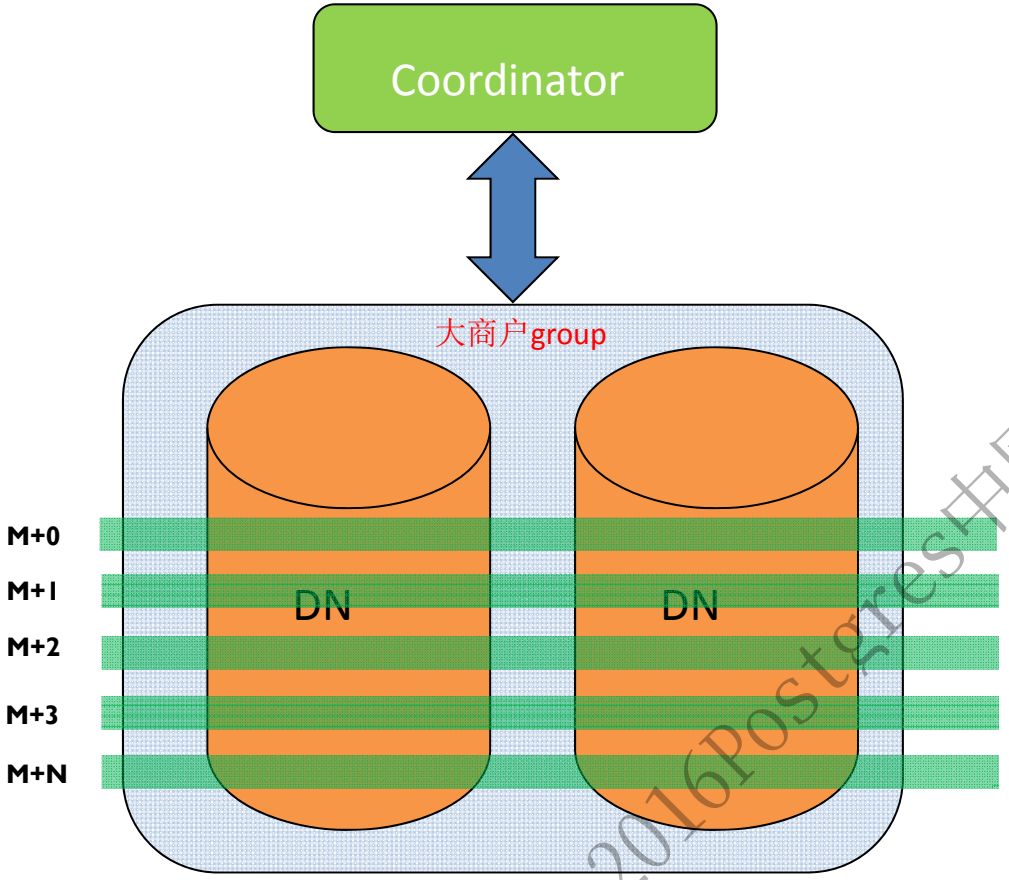
PGXZ技术解密—亿级数据快速排序，业务场景

平台上经常接到这样的SQL:

select * from t_trade_ref where (ftotal_amount between 1 and 683771 and fcreate_time >= '2015-07-31 00:00:00' AND fcreate_time < '2015-08-30 00:00:00' and fmerchant_id = 7777777) ORDER BY ffinish_time DESC LIMIT 10 offset x;

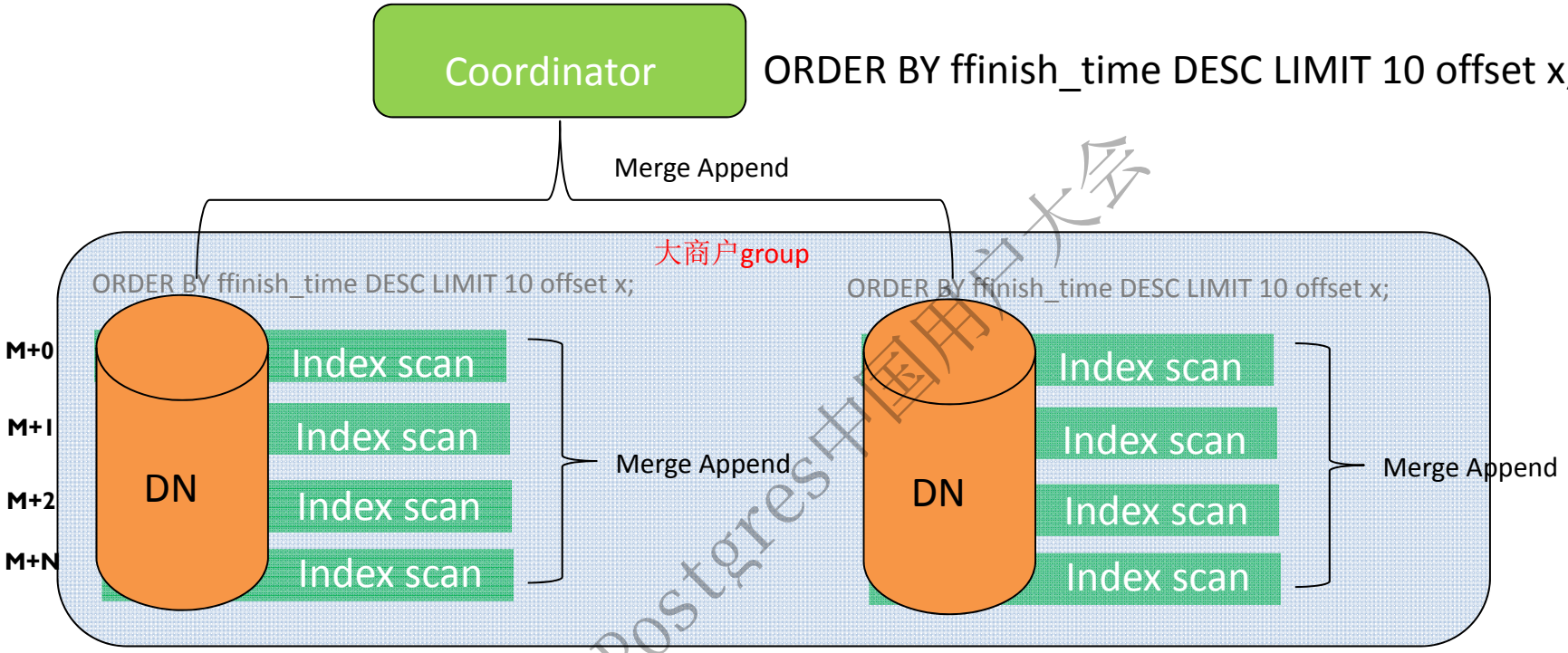


PGXZ技术解密—亿级数据快速排序，表定义



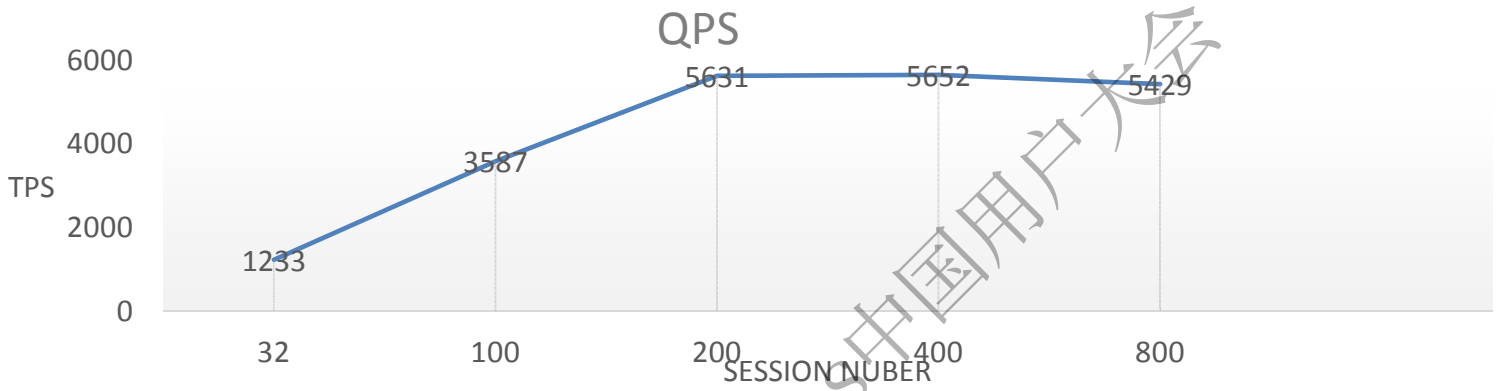
- 表建立为分区表，每个月一个分区。大商户的数据使用特定的数据分布逻辑，数据按照天被放到了不同的存储节点。
- 在 fmerchant_id 和 ffinish_time建立联合索引。 这样在排序的时候可以直接使用使用这个索引。

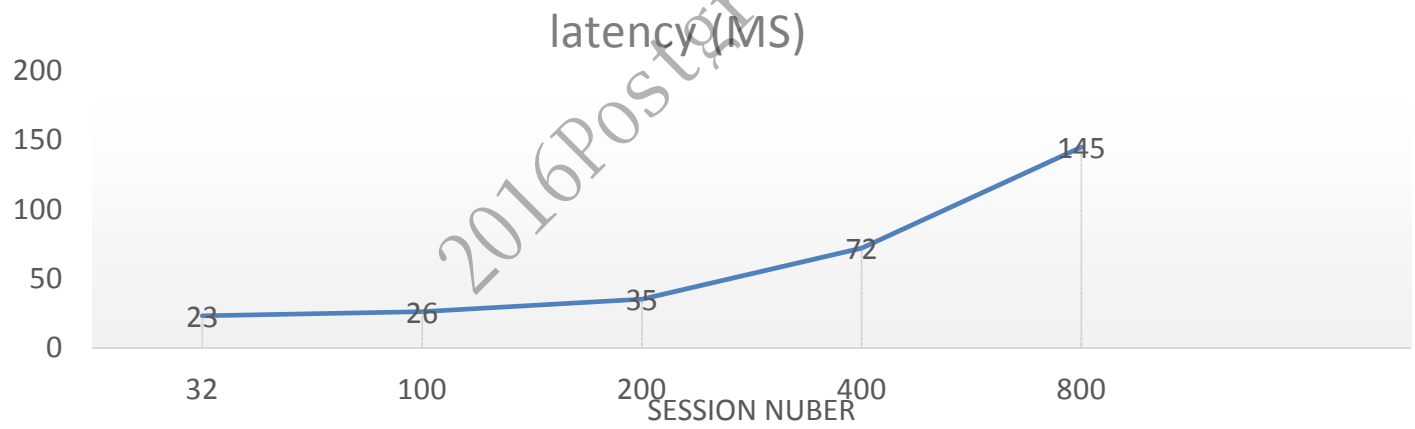
PGXZ技术解密—亿级数据快速排序，执行器优化



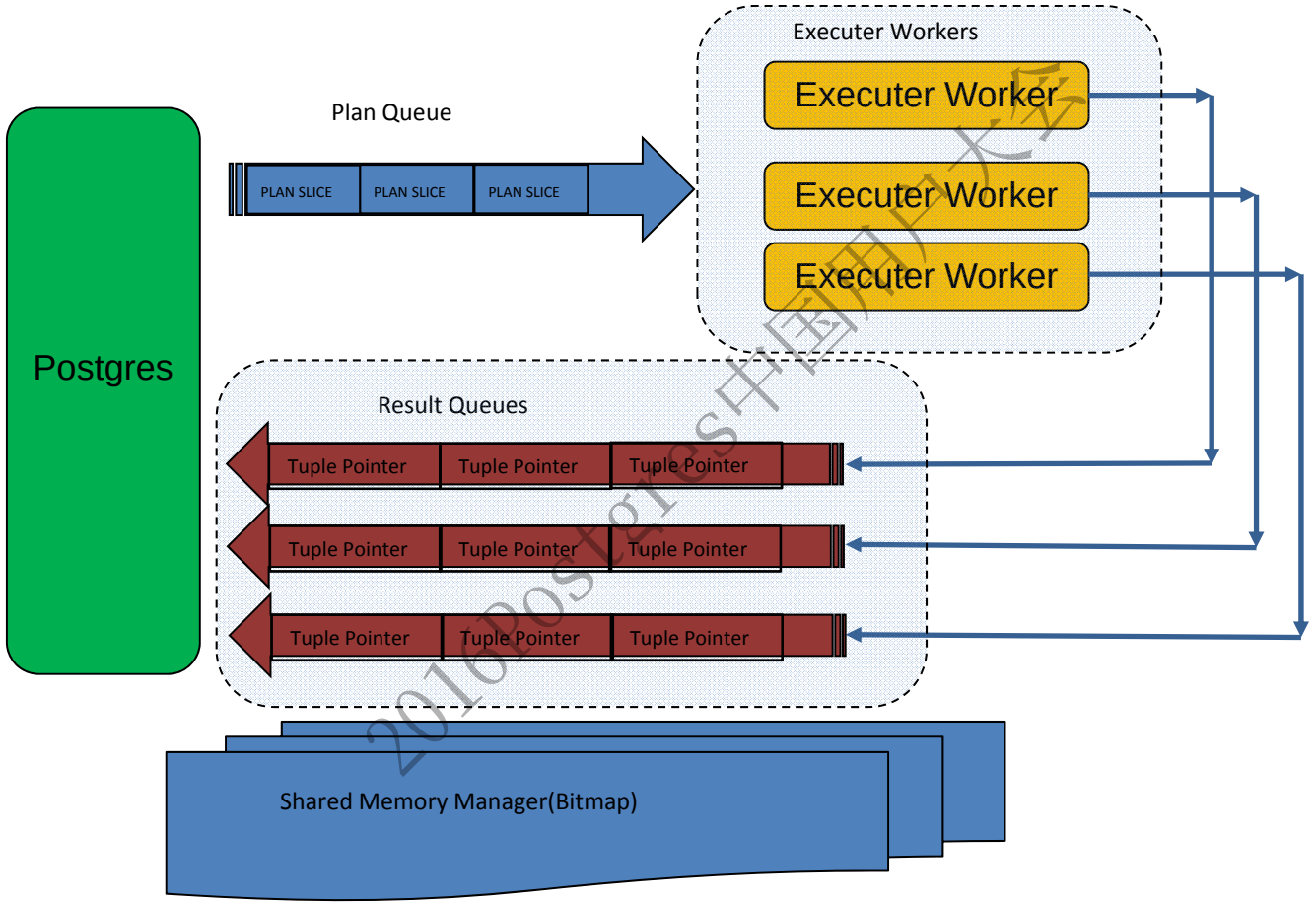
- CN 下推 limit offset。
- DN 使用之前建立的索引扫描每个表分区。
- DN 使用Merge Append算子合并每个表分区的数据。
- CN 使用Merge append算子合并每个DN的数据。

PGXZ技术解密—亿级数据快速排序，效果

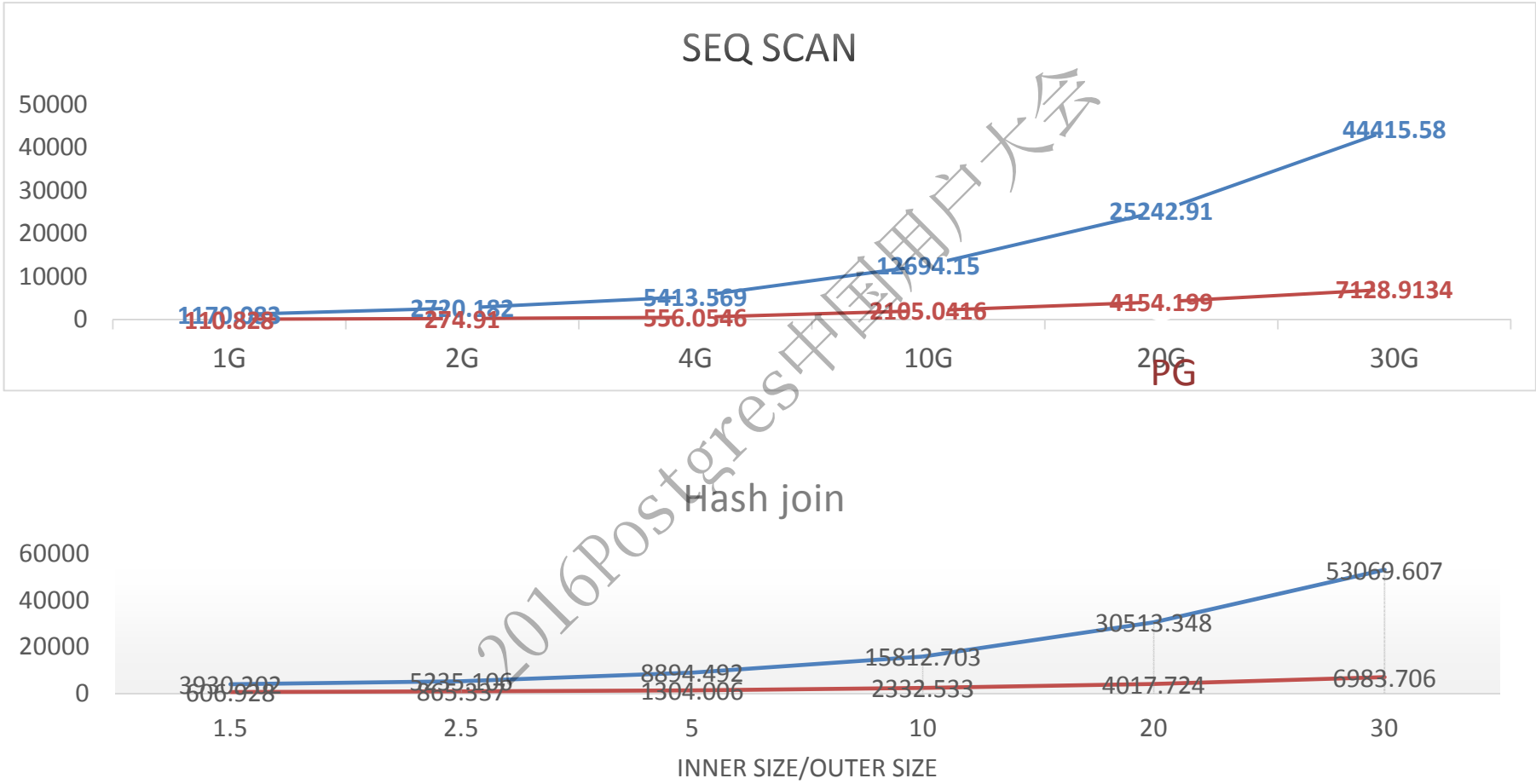




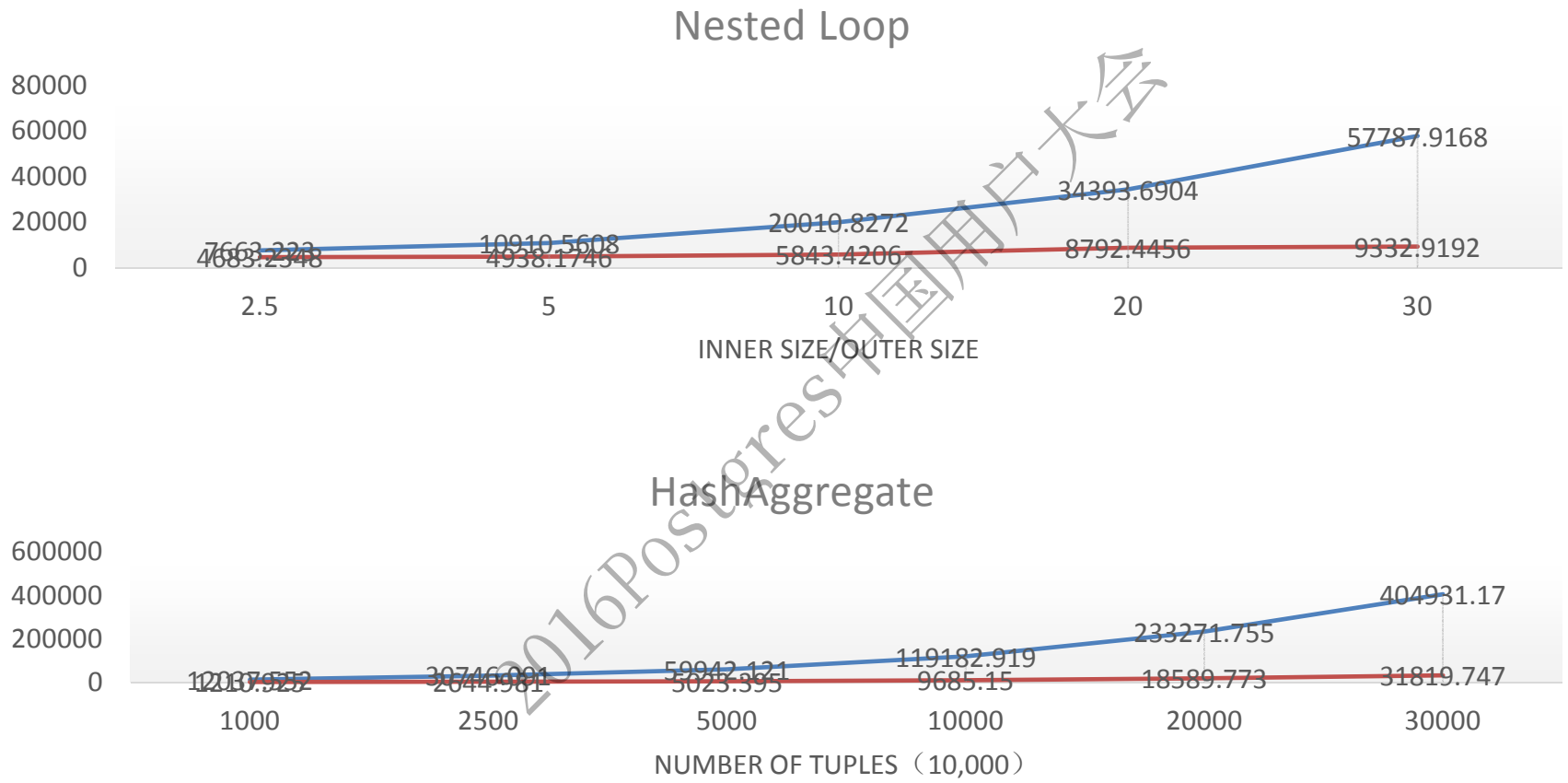
PGXZ技术解密—SQL多核并行执行



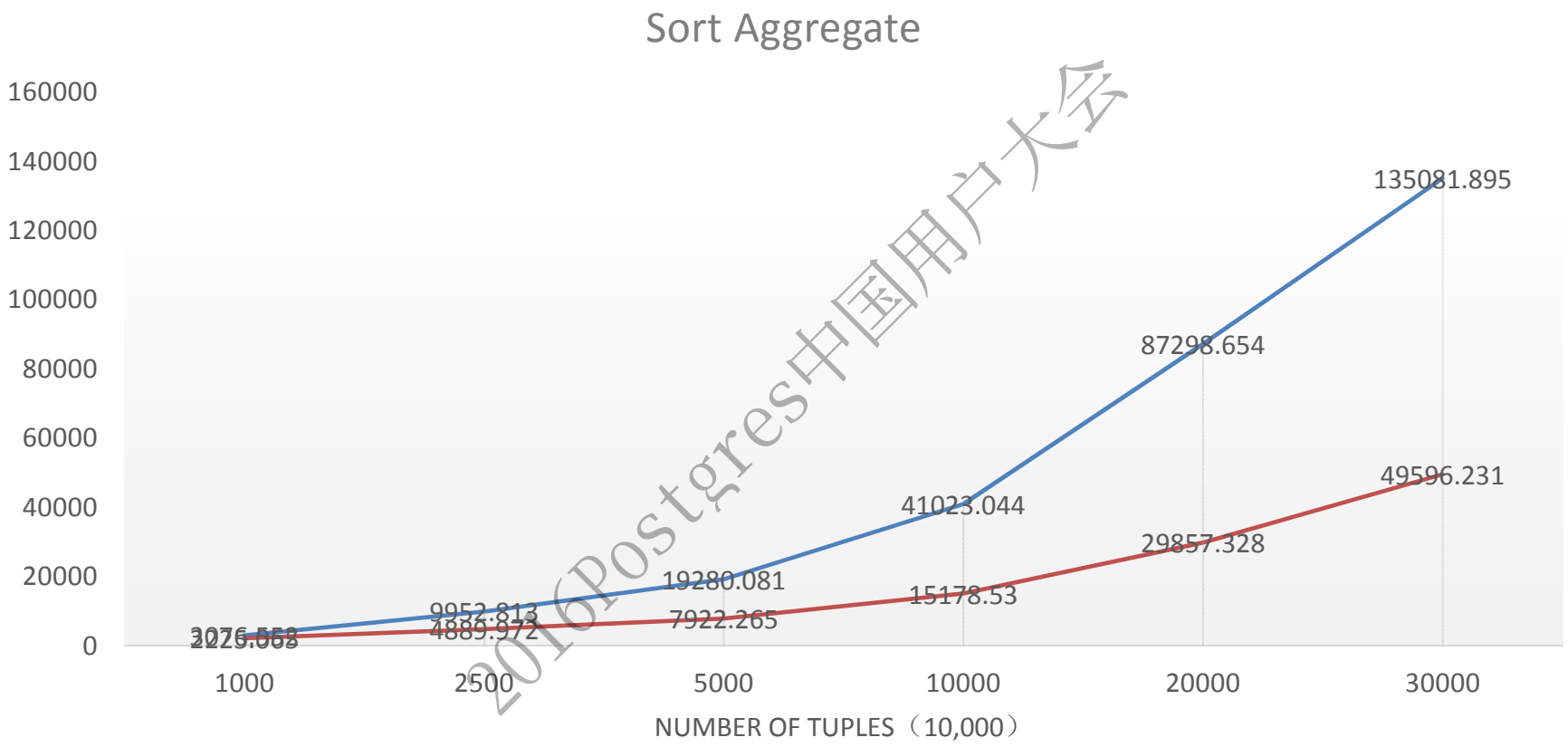
PGXZ技术解密—SQL多核并行优化结果1



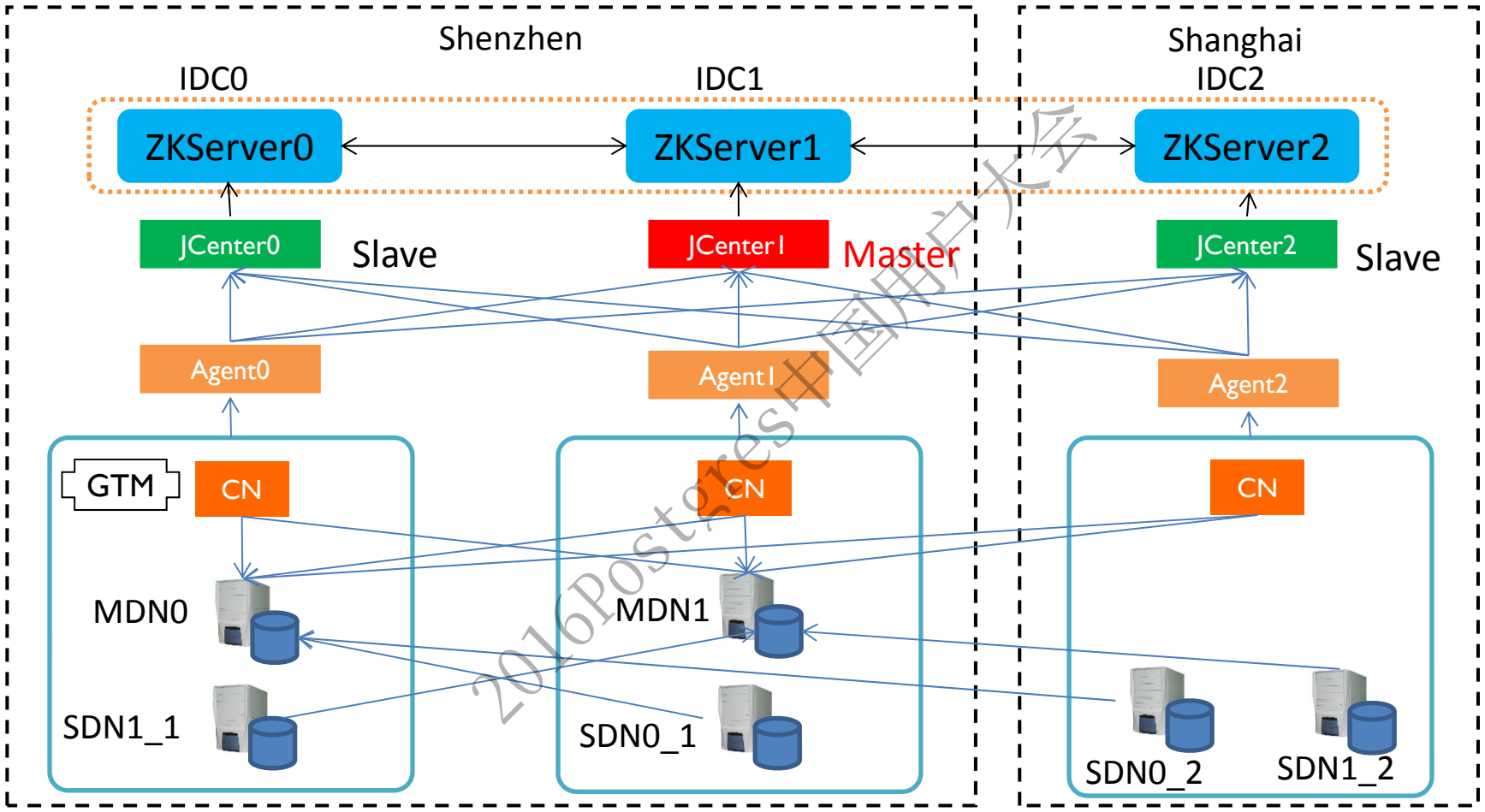
PGXZ技术解密—SQL多核并行优化结果2



PGXZ技术解密—SQL多核并行优化结果3



PGXZ技术解密—自动两地三中心容灾架构



PGXZ技术解密—自动两地三中心容灾原理

- 每个数据节点下面带两个备机，同步备机部署在同城IDC，异步备机跨城部署。
- 协调节点在每个IDC中都部署，确保任何一个IDC故障时都有协调节点可用。
- 每个IDC至少部署一个监控中心。
- 每个数据库节点统计部署监控agent监控上报节点状态，agent上报状态给所有的监控中心。
- 监控中心只用一个为主用，其他的都是热备，监控中心的主用由ZK自动选举产生。
- 发生故障时主用监控中心自动裁决节点的倒换策略。

PGXZ运维中的问题：索引膨胀

问题：

- 业务的交易模型导致系统中大量的UPDATE，引起索引膨胀，最高时膨胀超过300%
- 每张表上有7个以上的索引，导致磁盘占用过大，成本压力巨大

解决方案：

1、使用CREATE INDEX index_new CONCURRENTLY;建立和目标索引结构相同的索引。

2、EXCHANGE index index_new, index;使用我们自己开发的索引交换命令替换新建的索引

3、删除老索引，DROP INDEX index_new;

PGXZ运维中的问题：在线磁盘空间回收

问题：

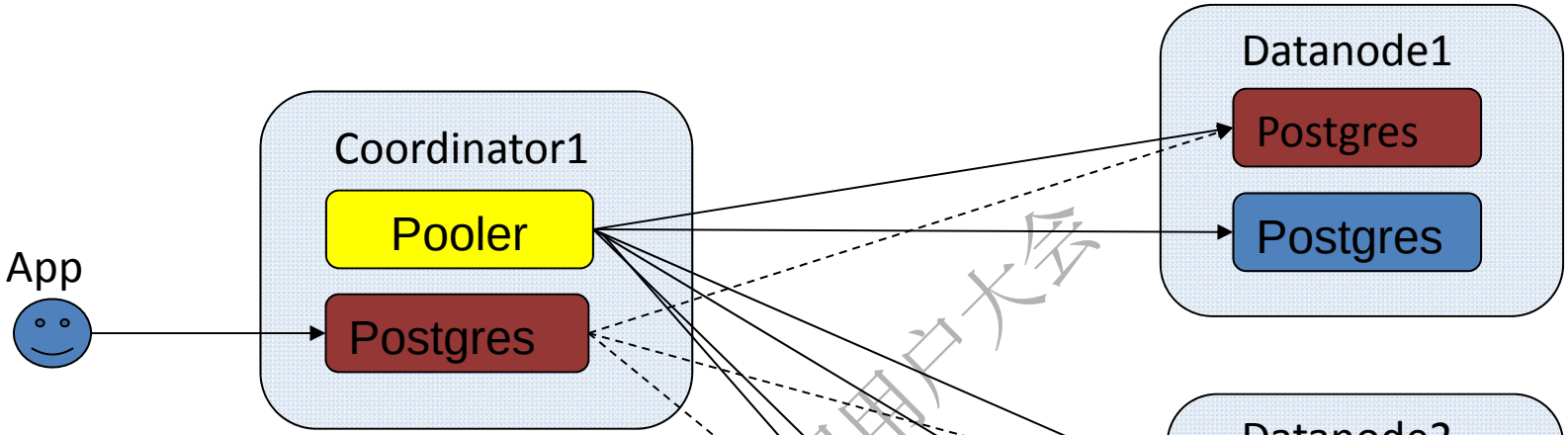
- 业务特点决定了数据的时效性，几个月以前的数据修改的次数会减少
- 在修改量少的表分区，磁盘空间没有办法有效的利用，磁盘中的空洞浪费了大量的空间。
- 在扩容后，数据迁出的节点的空间也需要通过释放无效的记录来释放磁盘空间。

方法：

腾讯自研命令：VACUUM FULL table CONCURRENTLY;

- 功能：不中断业务对表进行VACUUM FULL，重建整张表和表上的索引，达到释放磁盘空间的目的

PGXZ运维中的问题：系统连接总数的问题--CN到DN的连接管理方法



1、Pooler负责建立和管理通往其他CN和DN的连接

2、CN的会话跟其他的CN或者DN通信时，

从Pooler的连接池中获取连接的文件句柄

3、CN会话获取的文件句柄是Pooler通过sendfd的副本，

Pooler中会管理对应文件句柄的空闲状态

4、集群的最大连接数 = CN最大连接数 * CN个数= DN上的最大连接数

PGXZ运维中的问题：Pooler中的连接管理优化

DN最大连接数：

DN上的最大连接数=空闲内存 / 每个连接的平均内存

也就是说系统的最大连接数和系统的连接的平均占用内存成反比，因此系统的连接生命周期管理的主要目的：降低每个连接的平均占用内存。

解决方法：

1、连接生命周期管理，Pooler对于达到生命长度的连接进行重连

2、DN连接内存管理，对于连接占用超过配置的连接进行重连

Thanks!

Q & A