微信支付商户系统架构背后的故事

[李跃森](https://www.qcloud.com/community/user/16481600148101231)标签： [系统架构](https://www.qcloud.com/community/tag/100) ， [PostgreSQL](https://www.qcloud.com/community/tag/99" \t "_blank)

2016-10-12 15:20:45 21825

李跃森，[腾讯云PostgreSQL](https://www.qcloud.com/product/postgresql?fromSource=gwzcw.49215.49215.49215" \t "_blank)首席架构师，腾讯数据库团队架构师，负责微信支付商户系统核心数据库的架构设计和研发，PostgreSQL-x2社区核心成员，获多项国家发明专利。从事PG内核开发和架构设计超过10年。

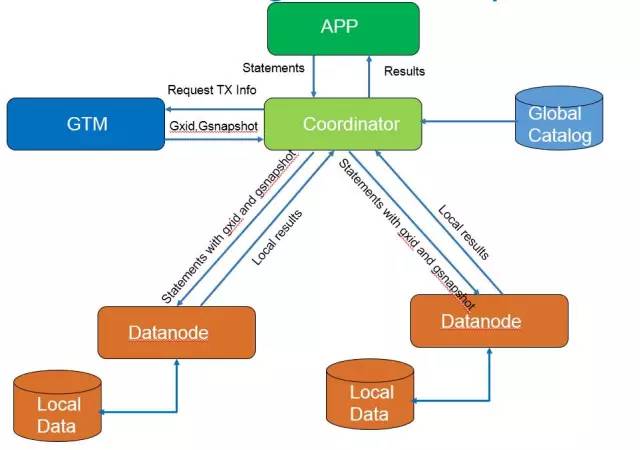
2015年之前，微信支付业务快速发展，需要一款数据库能够安全高效的支撑微信支付商户系统核心业务，这个重任落在了腾讯数据库团队自研PostgreSQL上。

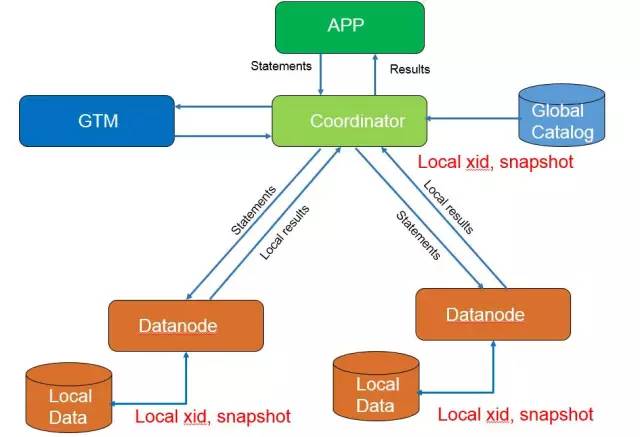
2016年7月，腾讯云对外发布[云数据库PostgreSQL](https://www.qcloud.com/product/postgresql?fromSource=gwzcw.49215.49215.49215" \t "_blank)，提供腾讯自研的内核优化版和社区版两个版本，以及提供分布式集群架构（分布式集群内部代号PostgreSQL-XZ）两种方案。目前云数据库PostgreSQL在腾讯大数据平台、广点通、腾讯视频等腾讯多个核心业务中稳定运行。

**腾讯自研PostgreSQL分布式集群 PostgreSQL-XZ**

腾讯PostgreSQL-XZ是由PostgreSQL-XC社区版本地化而来，能支撑水平扩展数据库集群。虽然PostgreSQL-XC很强大，但在性能、扩展性、安全、运维方面还是有明显的瓶颈，而腾讯PostgreSQL经过多年的积累，在这些方面都有较大提升和强化。由于是用于微信支付的核心数据库，腾讯PostgreSQL被定位为安全、高效，稳定，可靠的数据库集群。下面将以腾讯PostgreSQL-XZ为代表介绍腾讯自研PostgreSQL所做的优化和改进。

**一.事务管理系统的优化**

PostgreSQL-XC在事务管理系统方案本身有一个明显的缺点，那就是事务管理机制会成为系统的瓶颈，GTM（Global Transaction Manager全局事务管理器）会限制系统的扩展规模。如图1所示，是每个请求过来CN(Coordinator 协调节点)都会向GTM申请必需的gxid（全局事务ID）和gsnapshot（全局快照）信息，并把这些信息随着SQL语句本身一起发往DN（Datanode数据库节点）进行执行。另外，PostgreSQL-XC的管理机制，只有主DN才会获取的gxid，而备DN没有自己的gxid，因此无法提供只读服务，对系统也是不小的浪费。  
  
图1

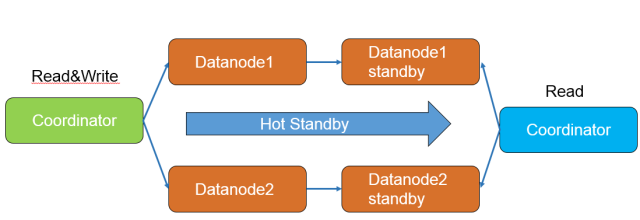
而腾讯PostgreSQL-XZ改进了事务管理机制，改进后，CN不再从GTM获取gxid和gsnapshot，每个节点使用自己的本地xid（事务ID）和gsnapshot（快照），如此GTM便不会成为系统的瓶颈；并且，DN备机就还可以提供只读服务，充分利用系统闲置资源。如图2，优化后的事务管理系统架构如下：  
  
图2

**二.备机只读实现与优化**

当然，事务管理系统的优化为进行备DN只读提供了基础，然而原始集群并没有负载、调度等能力。在这方面，我们也做了大量的创新，总结起来包括：

1. 正常CN和只读CN进行分离。
2. 正常CN存储主用DN的元数据信息
3. 只读CN存储备用DN的元数据信息
4. DN之间使用hot standby（热备份保护）模式进行日志同步

通过这些方式，集群可以提供带有智能负载能力的备DN只读功能，充分利用系统资源。

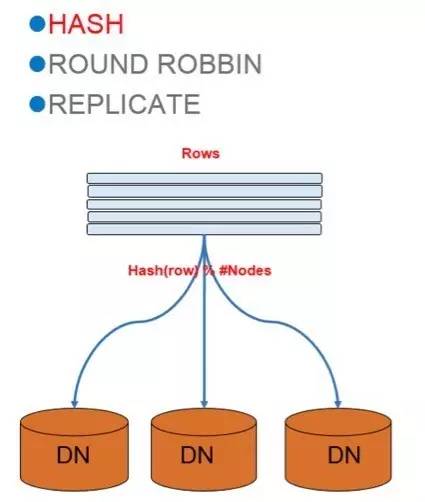
  
图3

**三.业务最小中断的扩容方案**

业务的快速增长不可避免的需要对资源进行扩容，社区版本的实现使得扩容成本高昂，需要对业务进行长时间的中断。因为，在社区版本PostgreSQL-XC中，通过 DN=Hash(row) % nofdn的方式决定一条记录的存储节点：

也就是说，先对分布列计算hash值，然后使用这个值对集群中的节点个数取模来决定记录去哪个节点（如图4）。

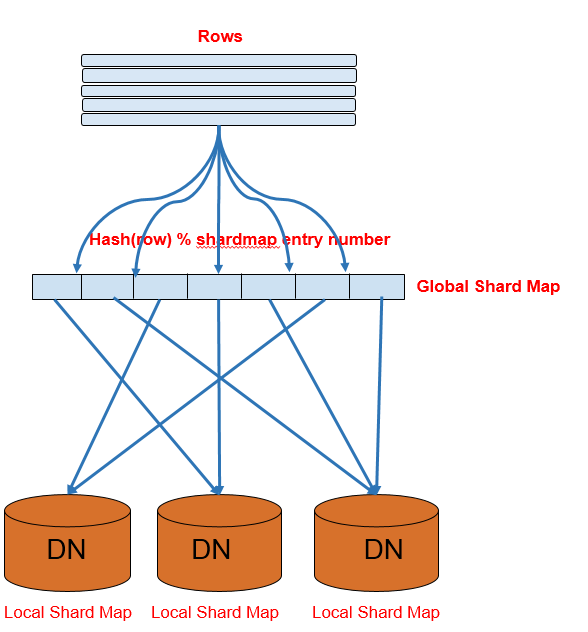
这种方案简单，但实际应用中需要长时间停机扩容。这是因为，扩容后节点数会变多，数据无法按照原有的分布逻辑进行读写，需要重新分布节点数据。而再均衡数据需要停机并手工迁移再均衡到各个节点。对于规模较大的交易系统来说，由于原有节点存储的是海量数据，再均衡过程可能会持续好几天。相信这是业务完全无法忍受的。

  
图4

因此我们引入了一种新的分表方法—sharded table。Shardedtable的数据分布采用如下（图5）的方式：

1. 引入一个抽象的中间层--shard map。Shard map中每一项存储shardid和DN的映射关系。
2. Sharded table中的每条记录通过Hash(row) % #shardmap entry来决定记录存储到哪个shardid，通过查询shardmap的存储的DN。
3. 每个DN上存储分配到本节点shardid信息，进而进行可见性的判断。

通过上面的方案，在扩容新加节点时，就只需要把一些shardmap中的shardid映射到新加的节点，并把对应的数据搬迁过去就可以了。扩容也仅仅需要切换shardmap中映射关系的，时间从几天缩短到几秒。

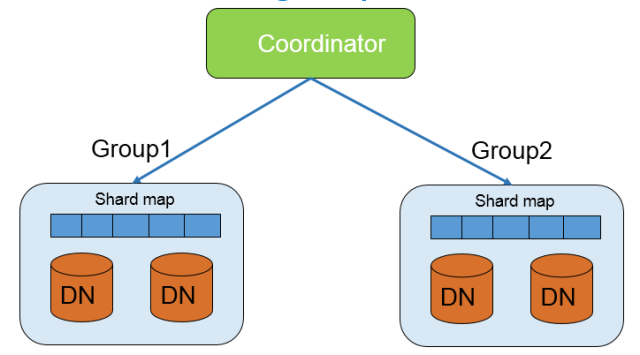
  
图5

**四.数据倾斜解决方案**

数据倾斜是指，在分布式数据库系统中会因为物理节点、hash或shard分布原因，导致某些DN物理空间不足，而另外的物理空间剩余较大。例如，如果以商户作为分布key，京东每天的数据量和一个普通电商的数据量肯定是天地差别。可能某个大商户一个月的数据就会把一个DN的物理空间塞满，这时系统只有停机扩容一条路。因此我们必须要有一个有效的手段来解决数据倾斜，保证在表数据分布不均匀时系统仍然能够高效稳定的运行。

首先我们把系统的DN分为group（如下图6），每个group里面：

1. 包含一个或者多个DN
2. 每个group有一个shardmap
3. 在建sharded表时，可以指定存储的group，也就是要么存储在group1，要么group2
4. CN可以访问所有的group，而且CN上也存储所有表的访问方式信息

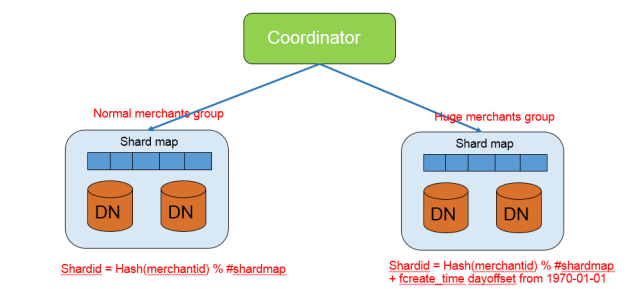
  
图6

对于系统中数据量较大用户进行特别的识别，并为他们创建白名单，使用不同的数据分布逻辑（如下图7）：普通用户使用默认的数据分布逻辑，也就是：

Shardid = Hash(merchantid) % #shardmap

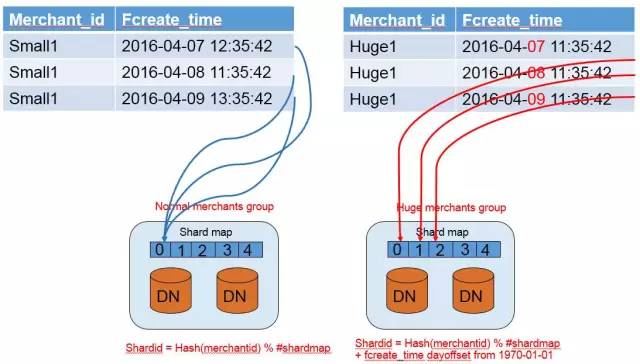
大商户使用定制的数据分布逻辑，也就是：

Shardid = Hash(merchantid) % #shardmap + fcreate\_timedayoffset from 1970-01-01

  
图7

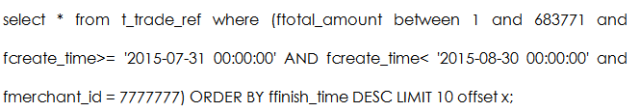
通过在大商户group分布逻辑中加入日期偏移，来实现同一个用户的数据在group内部多个节点间均匀分布。从而有效的解决数据分布不均匀问题。

下面是一个例子（如下图8）：

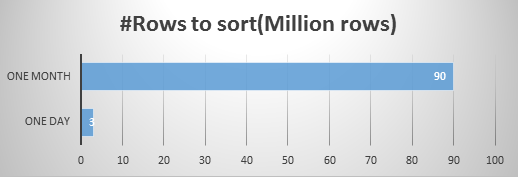
  
图8

**五.9000W记录高效排序解决方案**

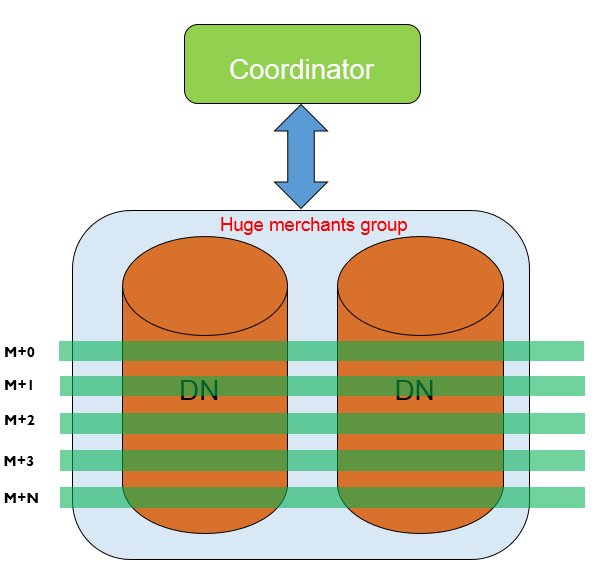
业务在列表查询场景下会收到如下的查询SQL：



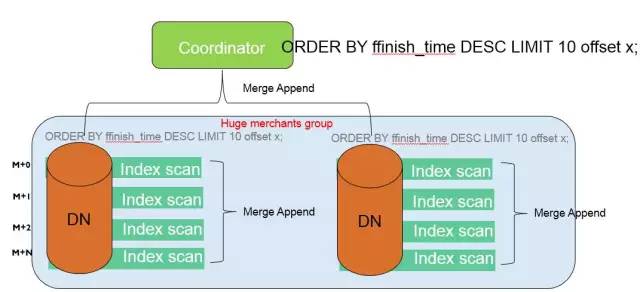
在微信支付的场景中，某个商户每天的数据有300W，一个月数据超过9000W条，也就是说PostgreSQL需要面向一个9000W数据级数据进行快速排序，而且业务逻辑要求需要秒级输出，快速获取排序结果。



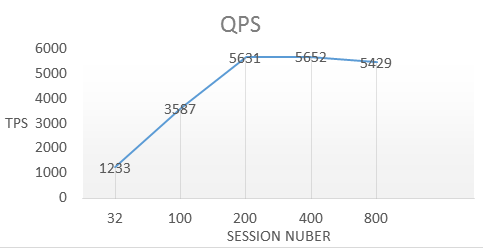
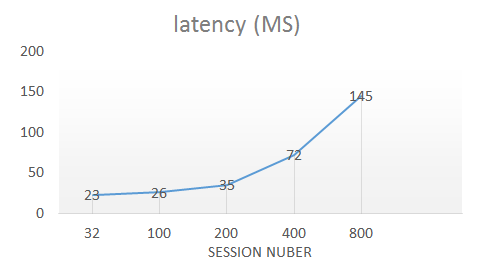
为此，我们提供表定义方案，即建立集群分区表。根据上述需求，可以采用按月分表，即每个月一张表，并对排序字段ffinish\_time建立索引，这样每个分区进行扫描是可以使用索引。



我们再通过一系列执行计划的优化，CN下推order by和limit offset子句到DN；DN上在执行对应的sql使用使用Merge Append算子对各个子表执行的结果进行汇总输出，这个算子本身会保证输出是有序的，也就是说对子表进行索引扫描，同时Merge Append又对各个子表的结果进行归并，进而保证节点本身的结果是排序的。CN对多个DN的结果同样使用Merge Append进行归并，保证整个输出结果是有序的，从而完成整个排序过程。



下面是我们对排序进行的性能测试结果：

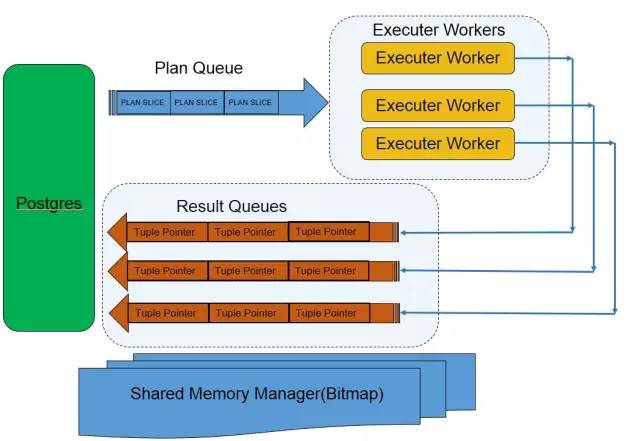
  


通过在24核CPU，64G内存的机型上进行测试，9000W数据的排序在最短可以在25 ms内完成，QPS最高可达5400。

**六.并行优化**

随着当前硬件的发展，系统资源越来越丰富，多CPU大内存成了系统标配，充分利用这些资源可以有效的提升的处理效率优化性能。腾讯在2014年底开始进行PostgreSQL多核执行优化。

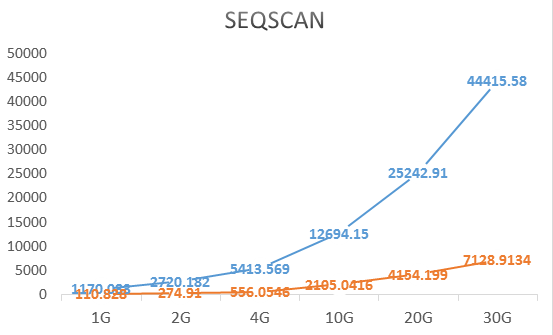
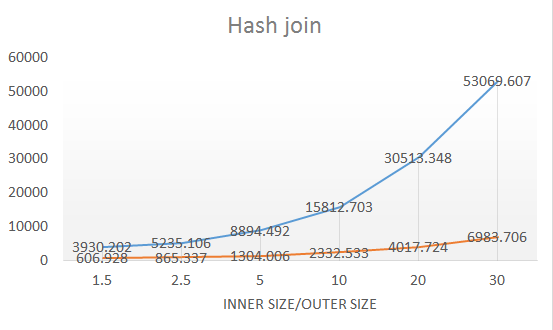
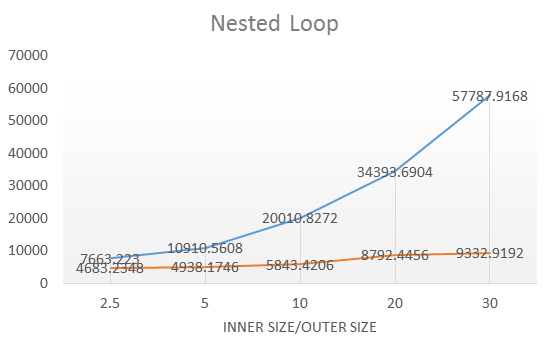
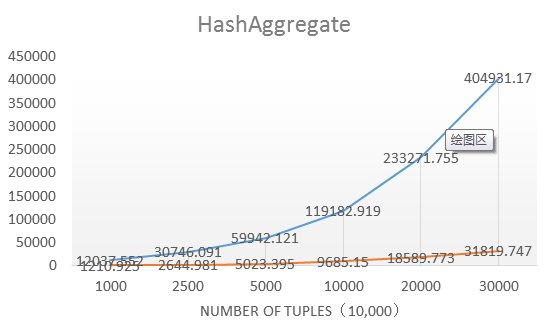
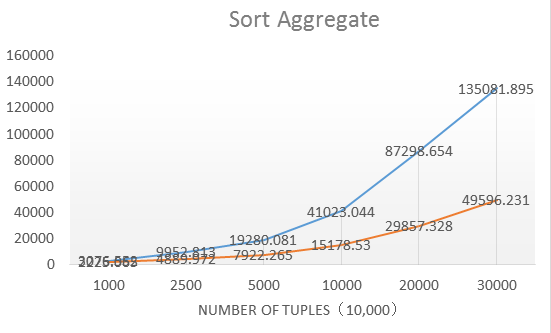
目前PostgreSQL9.6社区版也会包含部分并行化特性，但是没有我们这边这么丰富，下面介绍下腾讯PostgreSQL并行化的原理和效果：



* 系统创建一个全局的共享内存管理器，使用bitmap管理算法进行管理
* 系统启动时创建一定数据的Executor，这些Executor用来执行执行计划的碎片
* 系统会创建一个计划队列，所有的Executor都会在任务队列上等待计划
* 每个Executor对应一个任务结果队列，Executor在输出结果时就把结果的指针挂到结果队列中去
* 计划队列，结果队列，计划分片执行结果都存放在共享内存管理器中，这样所有的进程都可以访问到这些结构
* Postgres会话进程在收到sql时，判断是否可以并行化，并进行任务的分发；在结果队列中有结果时就读出返回

我们完成优化的算子：

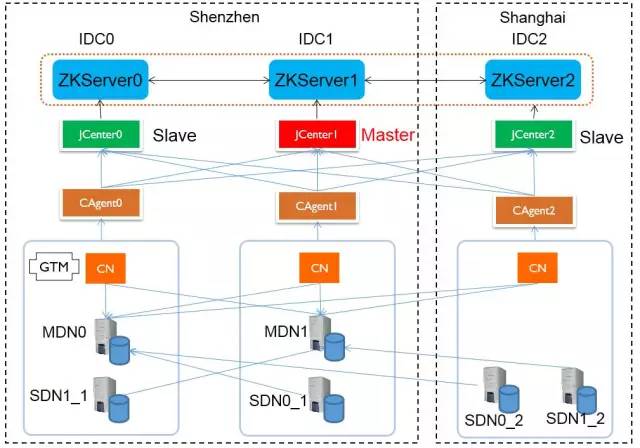
* Seqscan
* Hash join
* Nestloop join
* Remote query
* Hash Agg
* Sort Agg
* Append

通过在24核CPU，64G内存的机型下测试，各个算子的优化结果：  
  
  
  
  


整体来说性能普遍是优化前的10-12倍，优化的效果比较明显。

**七.腾讯PostgreSQL-XZ的两地三中心容灾**

两地三中心容灾是金融级数据库的必备能力，对于金融类业务数据安全是最基本也是最重要诉求，因此我们为了保障高效稳定的数据容灾能力，也为PostgreSQL-XZ建设了完善的两地三中心自动容灾能力。具体的两地三中心部署结构如下：



同城节点间采用强同步方式，保障数据强一致；异地采用专网异步同步。

节点内，每台物理机上部署CAgent，agent收集机器状态并进行上报，并进行相应的告警和倒换执行功能。

每个IDC至少部署一个JCenter，JCenter负责收集上报每个agent上报的状态到ZK集群。这么多个JCenter中只有一个是主用，主用的JCenter除了进行状态上报还进行故障裁决和倒换。在主用的JCenter异常后，系统通过ZK自动裁决挑选一个备用的JCenter升主。

JCenter和CAgent是两地三中心的控制和裁决节点。

对于数据库节点，CN在每个IDC至少部署一个。DN在每个中心部署一个，一个为主，另外两个并联作为备机放在主机上，一个为同步备机，另外一个为异步备机。

在主机故障宕机时，JCenter优先选择同城的备机升主。

目前，腾讯云已经提供[云数据库PostgreSQL](https://www.qcloud.com/product/postgresql?fromSource=gwzcw.49215.49215.49215)的内测使用，并将提供内核优化版和社区版两个版本来满足更多客户的要求。