

阿里云开发者社区 | 阿里云数据库产品事业部

# PostgresConf.CN & PGConf.Asia 2020

PostgreSQL在阿里云的实践与发展

- | 阿里云 PG 独家实践
- | 生产级应用经验分享



阿里云开发者电子书系列



微信关注公众号：阿里巴巴数据库技术  
第一时间，获取更多技术干货



阿里云开发者“藏经阁”  
海量免费电子书下载

# 目录

浅谈数据库发展趋势	4
云原生数据仓库的机遇和挑战	23
传统数据库异构上云实践	38
下一代云数据库： MyBase 专属集群	49
乘云而上， ADB 云原生数仓发展之路	62
云原生数据仓库 TPC-H 第一背后的 Laser 引擎大揭秘	74
UniqueKey:让你的查询跑的更快	86
Implementation of Global Temp Table	97
RDS PostgreSQL 管控体系介绍	109
DTS 及其在 PG 数据库生态中的应用	124
PG 数据库生态选型思路与最佳实践	141
PostgreSQL CDC 的最佳实践	160

# 浅谈数据库发展趋势

作者 | 叶正盛 阿里云数据库产品经理 数据库产品与运营负责人

## 一、数据库市场概述



数据库发展可以分为 5 个阶段：

**第一阶段**，上个世纪 50 年代，那时候计算机的基础设施还是大型机，整个全球大型机也非常少，大概小于 100 台。

那时候的大型机主要用在国防、科学研究这些方面，当时数据库大概在 60 年代出现了层次数据库和网站数据库，比较有代表的是 IBM IMS，这款数据库现在用的比较少，在一些金融领域偶尔能看到。

**第二阶段**，上个世纪 70 年代，这时候开始流行小型机，计算机除了在科学研究、国防方面应用以外，已经开始渗透到大型的商业处理，这时候也诞生了关系型数据库，包括比较有名的 Oracle，还有 IBM 的 DB2 等等。

**第三阶段**，到第三个阶段，这时候 PC 机还有 X86 服务器以及局域网开始普及起来，全球的计算机也非常多了，并且开始应用在各个领域，包括像 ERP、CRM 还有财务这样的一些信息化系统，开始全面的推开。

另外在个人办公方面，像办公娱乐等等，会有非常多的应用，这个时候也诞生了很多新的数据库，包括像关系型数据库也在全面发展，另外还有数据仓库以及一些单机数据库，还有现在还经常能看到的 Teradata、SQL Server 以及 PostgreSQL 等等。

**第四阶段**，在 2000 年左右，这个时候开始了互联网，全球的计算机也接近 10 亿台。这时计算机开始全面应用，包括像媒体、雅虎、新浪等等，还有搜索、社交以及电子商务和娱乐等等。

因为互联网的蓬勃发展，数据库也涌现了非常多出名的开源数据库，包括像 MySQL、Redis、MongoDB 等等。

**第五阶段**，在 2015 年左右，定义为在云+端时代，也就是现在我们所处的时代。这个时候开始有很多新媒体，移动的 APP，O2O 以及我们今天大量应用的云计算，还有物联网和最近一年比较火的在线办公、教育、娱乐等等。

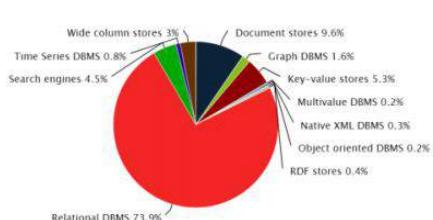
这个时候云数据库开始全面发展。

### 全球数据库类型统计 (DB-Engines)

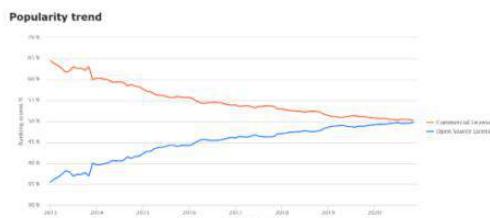
#### DB-Engines(360种数据库, 2020.11)

PostgresConf.CN &  
PGConf.Asia 2020 | 阿里云 | 奥运会全球指定云服务商

关系型 & NoSQL



商业 & 开源生态



The top 5 commercial systems, November 2020

Rank	System	Score	Overall Rank
1.	Oracle	1349	1.
2.	Microsoft SQL Server	1038	3.
3.	IBM DB2	162	6.
4.	Microsoft Access	117	11.
5.	Splunk	90	13.

The top 5 open source systems, November 2020

Rank	System	Score	Overall Rank
1.	MySQL	1242	2.
2.	PostgreSQL	555	4.
3.	MongoDB	454	5.
4.	Redis	155	7.
5.	Elasticsearch	152	8.

DB-Engines,是一个非常专业的数据库百科全书网站，从 DB-Engines 上来看，数据库的种类非常多，截至 2020.11 月份已经收录了 360 种数据库，从数据库的模型看，主要分为关系模型和 NoSQL。

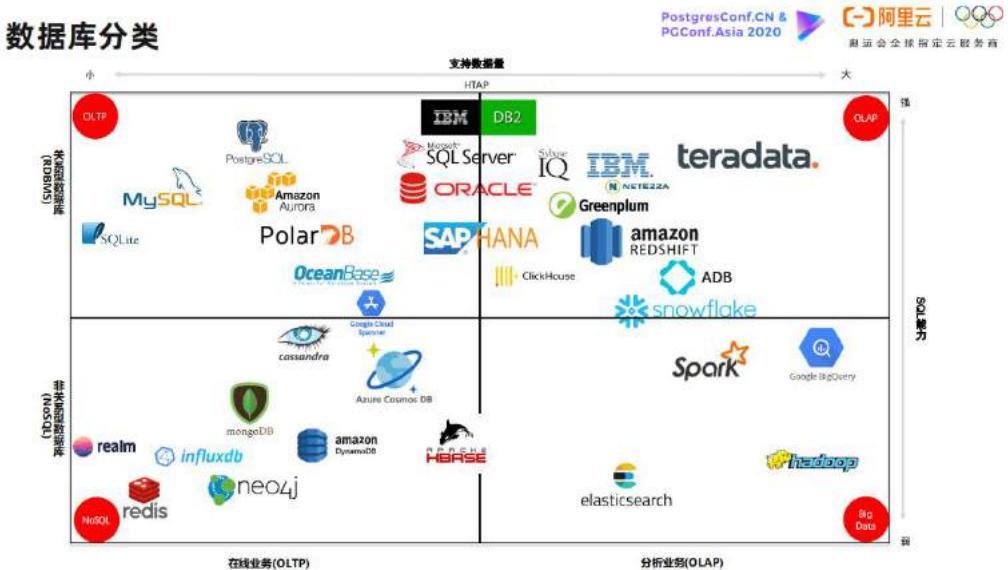
关系模型大概是占 3/4 左右的影响指数；

NoSQL 种类非常多，有文档、图，还有时间序列等等。

从商业和开源两个角度来看，开源数据库发展得非常迅猛，现在开源数据库和商业数据库基本上平分了整个市场。

下面是 Top5 的商业数据库和开源数据库列表，开源的里面有最流行的 MySQL，还有功能强大的 PostgreSQL 等等。

## 主流数据库应用场景分类



分为 4 个大类：

- 左上角是 OLTP 类型，是应用于在线业务的关系型数据库，这里面有非常多的引擎，像 MySQL、PostgreSQL 都是非常出名的产品，以及像 AWS 的 Aurora，还有阿里云的 PolarDB、OceanBase 等等，都在这个领域里面占据了比较高的市场影响力。

- 右上角是比较专业的 OLAP ( 数据仓库 ) 领域。
- 这里面有 Teradata，还有 AWS 的 Redshift 以及非常火的 Snowflake，都属于这个领域。还有基于 PG 开发的 Greenplum.
- 在中间有几个比较老牌的数据库，像 DB2、SQL Server，还有 Oracle，还有 SAP HANA，这些它基本上在 OLTP 和 OLAP 两个领域能够应用。
- 在左下角是新兴的 NoSQL 数据库，种类也非常多，在不同的方向都有比较典型的产品。

这里面列举了一下，像 Redis 在缓存方面非常强，还有 MongoDB、Neo4j 以及 Cassandra 等等，还有 HBase 都在 NoSQL 里面是比较强的产品。

- 在右下角属于大数据领域，也有非常多的产品，像 Elasticsearch、Hadoop、Spark 等等。

整体上来讲，从左边到右边，它处理不同的数据库类型，处理的数据量是从小到大，然后从底下到顶上，它的各种数据库的 SQL 能力也有不同的描述，基本上越往上它的 SQL 能力越强，越往下 SQL 能力越弱。

这个是从应用场景以及数据库的能力方面来做的一个宏观的划分。

## 数据库——云计算关键技术

### 为什么数据库这么重要？

现在很多的厂商都在做数据库，包括国际科技巨头，像 Oracle、Google 等等，以及国内的阿里巴巴、华为，还有腾讯、百度等等。

在今天，基础设施 IaaS 这一层其实建设得比较完善，大家都认可未来的应用是智能化的应用，从基础设施到智能化应用，很关键的一环就是数据库，这里面包括数据的产生、存储和消费将如何高效的去处理，都需要数据库来完成。

数据库可以说是未来基础技术里非常关键的一环。

## 数据库——云计算关键技术

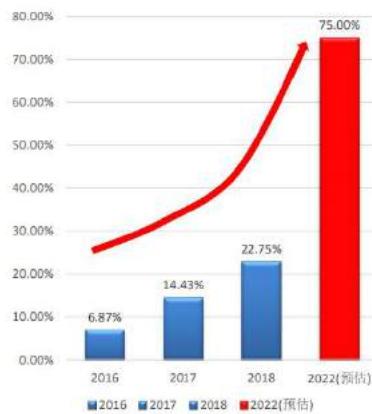
PostgresConf.CN &  
PGConf.Asia 2020 阿里云 |  
奥运会全球指定云服务提供商



## 二、数据库发展趋势

### 数据库发展趋势（云是未来， by Gartner）

云数据库快速发展，2022预计75%的数据库都在云上



数据来源于Gartner报告

PostgresConf.CN & PGConf.Asia 2020 阿里云 |  
奥运会全球指定云服务提供商

魔力象限中云数据库厂商成长最快



国际上最权威的科技调研机构 Gartner，它的建议和数据非常直接——数据库的未来就是云。

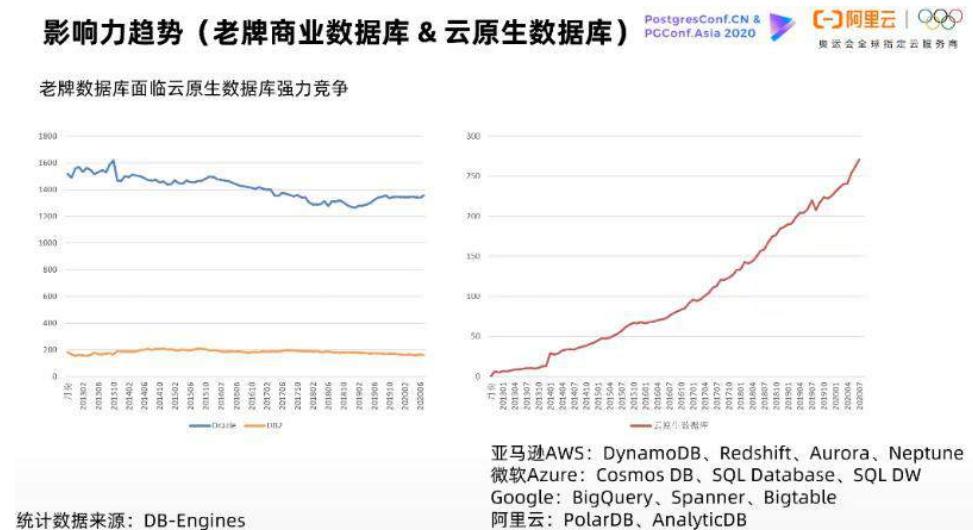
它有一个比较明确的预估。2022 年基本上有 3/4 的数据库都跑在云上，这是它的预估数据。而前面 2016、2017、2018 年可以看到云上的数据库占比也在逐年的递升，2018 年已经达到了 22.75%，2019 年预计超过 30%。这个是在云上数据库的部署比例。

再看右边，这是 Gartner 数据库 OPDBMS 的魔力象限图，标框的是云厂商，魔力象限中我们看到云厂商是成长最快的。

在魔力象限的右上角是 leader 象限，在左上角是挑战者象限，这两个象限中都是云厂商在这唱主角，包括像 Microsoft、Google 以及阿里云，这些都是在数据库里面成长非常快的企业。

另外也可以看到 IBM 曾经的老牌数据库厂商，现在已经从 leader 掉到了远见的象限。这是 Gartner 关于数据库的发展趋势，它非常明确**云就是未来**。

## 影响力趋势



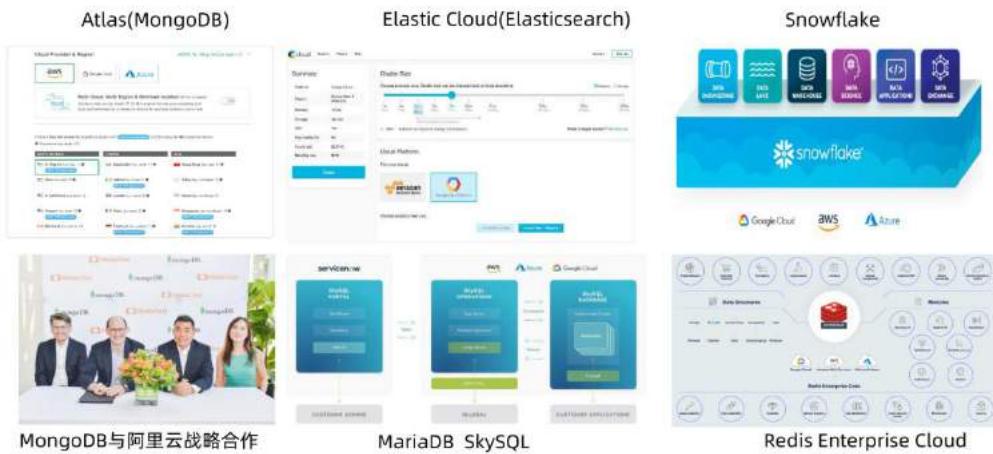
我们从 DB-Engines 上再来看一些数据，从两个图表来看，左边这个是我们比较老牌的 Oracle 和 DB2，两个数据库影响指数是比较平稳的。

右边我们统计了所有的云厂商的云原生数据库，包括 AWS、DynamoDB、Redshift 等等，以及微软的 Cosmos DB，还有 Google 的 BigQuery，阿里云的 PolarDB、AnalyticDB，这些都是云原生数据库。可以看到云原生数据库从 2013 年 1 月份开始蓬勃发展。这个也是对传统的老牌商业数据库带来了非常大的挑战。

### (一) 通用数据库支持多云部署是最重要的战略方向

## 通用数据库支持多云部署是最重要的战略方向

PostgresConf.CN & PGConf.Asia 2020 阿里云 | 阿里云  
奥运会全球指定云服务商



- 像 Atlas ( MongoDB ) 的多云部署平台，可以部署在 AWS 还有 Google 以及微软云平台。MongoDB 跟阿里云也是非常重要的战略合作，基本上 MongoDB 它在几大主流的云平台上都有部署模式，这个是 MongoDB 能够发展非常快的一个主要原因。Atlas 也已经成为 MongoDB 的主要收入来源。
- 第二个，Elasticsearch 有一个非常强大的云平台叫 Elastic Cloud。这里面也支持如 Azure、Google，比较有名的平台都可以在上面部署。
- 第三个 Snowflake，现在估值已经超过 700 亿美金了，它也是一个非常标准的多云部署架构，而且它只支持在云环境下部署。目前他已经支持了 Google、AWS 以及 Azure。
- 还有 MariaDB，它有一个多云的平台叫 SkySQL，也是支持了国际上几个比较重要的公共云厂商。还有 Redis 推出了 Enterprise cloud,也是一个企业版的多云部署，也是支持国际这几大公共云厂商。

基本上目前比较有影响力的产品，都在朝多云部署这个方向去布局。像 MongoDB 还有 Elasticsearch 以及 Snowflake 这些产品。都已经取得了巨大的成功，都得益于多云的部署，可以让更多的用户使用到产品，客户可以更方便的把业务跑在云上。

## 通用数据库支持多云部署是最重要的战略方向

产品	是否开源	AWS	Azure	Google	阿里云	腾讯云	华为云	业务增长
Oracle	否	√	✗	✗	✗	✗	✗	=
SQL Server	否	√	√	√	√	√	√	↑
DB2	否	✗	✗	✗	✗	✗	✗	=
Teradata	否	✗	✗	✗	✗	✗	✗	=
MySQL	是	√	√	√	√	√	√	↑
PostgreSQL	是	√	√	√	√	√	√	↑
Redis	是	√	√	√	√	√	√	↑
MongoDB	是	√	√	√	√	√	√	↑
Elasticsearch	是	√	√	√	√	√	√	↑
Snowflake	否	√	√	√	✗	✗	✗	↑

- 开源对于提升产品影响力非常有价值
- 有竞争力的产品+支持多云部署是被验证成为非常有效的模式

上面这个表格，展示了目前市场上影响力比较大的产品的多云部署情况，以及他们是不是开源。

像 Oracle、SQL Server、DB2 以及 Teradata，这些都不是开源的产品。他们也有不同的战略，比如像 SQL Server，是非常 open 的，它基本上在所有的公共云厂商都支持部署，这也是 SQL Server 虽然是一个比较老的数据库，但是业务增长非常快。

像 Oracle、DB2 以及 Teradata，这几款产品是闭源的，也不支持在多云部署，业绩增长比较平缓。

另外很多款有名的开源数据库，像 MySQL、Redis 等等，这些都是开源的，主流的云厂商基本都支持，这些产品影响力也变得越来越大。

而 Snowflake 是一个商业的数据仓库，目前通过支持 AWS 、Azure、Google 三大云厂商，目前发展也非常迅速。

从这些主流产品可以看到，开源对于提升产品的影响力非常有价值，但要把一个产品做的商业成功和开源或不开源，是没有直接关系的。Snowflake 不开源，但其实他也做得非常不错，还有 SQL Server 也发展的非常不错。

**总结：做出有竞争力的产品，支持多云部署，这个是被验证为非常有效的模式。**

多云部署可以让你的产品被更多的用户更方便的使用，这个是它最重要的价值。

而且未来云计算一定是最重要的发展方向。如果一个通用数据库不支持在云上部署，他将会失去大量的用户。

## (二) 数据库发展趋势 (存储计算分离)

存储计算分离，是一个技术方向，也是今天云计算带来的技术红利。

以阿里云 PolarDB 为例子，阿里云 PolarDB 是今天阿里云增长最快的数据库，下图右边是它去年的增长曲线。



PolarDB 的技术架构，整体上它可以分成三层架构：

- 最底层是分布式存储 PolarStore，通过 ParallelRaft 可以保证数据库数据存储的高可用，以及数据的一致性；
- 中间的计算节点就是数据库运行节点，PolarDB 是支持一写多读，它有一个主节点，可以扩展很多的只读节点，最多可以扩展 15 个。数据库的计算节点和 PolarStore 存储节点是通过 RDMA 高速网络通信；
- 上面是智能代理 PolarProxy，智能代理也是非常重要的一个组成。应用程序可以通过智能代理，做自动的负载均衡和读写分离以及 HA 切换等等。

通过存储计算分离 PolarDB，可以做到快速的弹性，因为它的数据都统一放在分布式存储方面，扩容的时候，数据不用迁移，计算节点是无状态，可以快速的扩展。

另外可以做到几乎没有主备延时，自动读写分离。

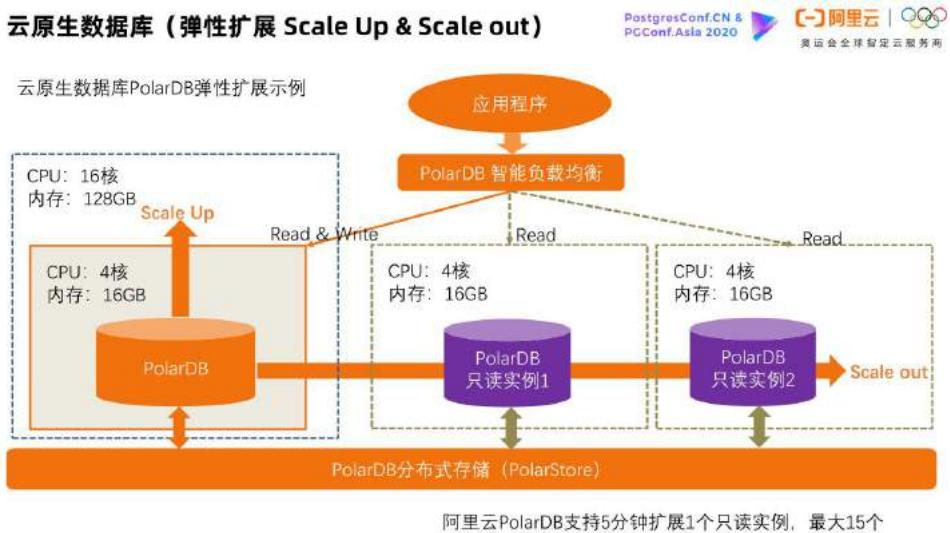
通过分布式存储可以做到最高的单实例存储 100TB 数据。

目前 PolarDB 也兼容很多款主流的引擎，包括 MySQL 和 PostgreSQL 基本上是完全兼容。

另外也是高度兼容 Oracle，基本上 95%以上的语法兼容度。

存储计算分离对于数据库来讲，它的弹性能力也是非常吸引用户的。PolarDB 快速发展，特别是在新冠疫情期间，我们可以看到教育行业还有游戏行业发展非常快。今天在阿里云上如猿辅导、好未来这些客户，还有游戏里面的心动网络、米哈游等等都是全面选择了 PolarDB，这也是用户通过存储计算分离实现弹性扩展的产品认可。

### (三) 数据库发展趋势（存储计算分离，弹性扩展）



上图是一个弹性扩展的例子，假如说你有一个 PolarDB 数据库，它的配置是 4 核 16 G，如果说你的业务快速发展，要扩容，可以先用最简单的 Scale UP。可以很快的扩到 16 核 128GB，数据不需要搬迁，因为数据在底下的分布式存储上面。

如果说在 Scale UP 这个方向，发现你的负载仍然满足不了，还可以用横向扩展，可以添加非常多的只读节点，PolarDB 可以添加到 15 个，因为它采用分布式存储，所以它的数据是完全共享的。只要把计算节点启动起来，就实现了横向扩展。这个时候基本上都可以满足业务负载。

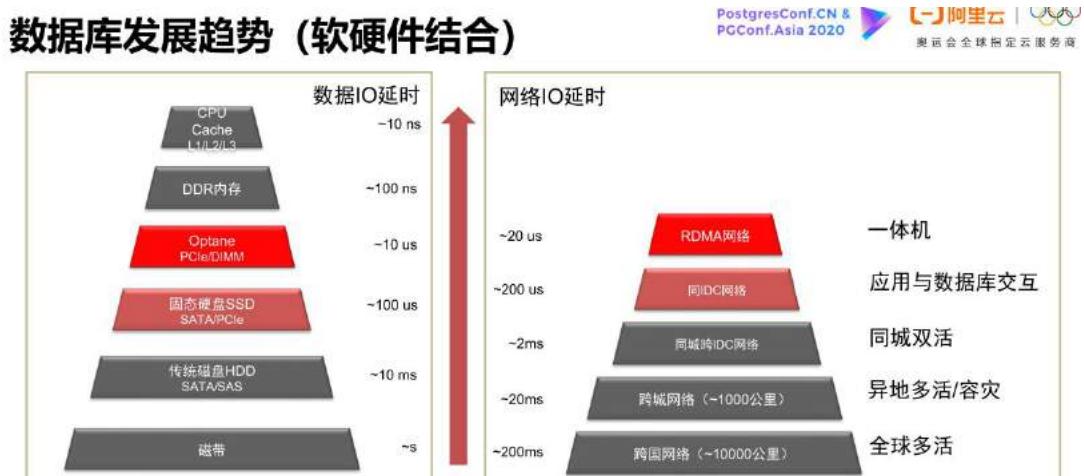
很重要的一点，它有智能负载均衡，扩展了节点之后，如果没有智能负载均衡这一层，应用程序需要去重新的修改发布，对于业务系统来讲是非常不便的。

如果有了智能负载均衡，应用是可以完全不用修改的，只需要关心业务需要什么样的负载，然后快速给 PolarDB 扩容，它可以做到 5 分钟扩展一个只读节点。

这个是存储计算分离带来弹性扩展的优势，也是未来数据库的主要发展方向。

我们看到其实不只是 PolarDB，还有其他的一些云原生数据库，基本上都是这样一种架构。

#### (四) 数据库发展趋势 (软硬件结合)



对于数据库来讲，IO 是一个非常重要的瓶颈。我们把 IO 分为两类，一类是数据 IO，一个是网络 IO。

数据 IO，我这边列了比较常见的存储数据的和计算设备。最底下是磁带，磁带现在已经用的比较少了，主要是用来做归档备份，我们就不去谈。再往上是传统的磁盘 HDD，访

问延时大概在 10 毫秒，数据库用的比较多的固态硬盘 SSD，它的访问延时是大概在 100 微秒。

另外一个就是 Optane，它的访问延时大概在 10 微秒，Optane 其实是一个非常重要的硬件，对于数据库来讲，它有可能会带来革命性的变化。

其实很多厂商都在基于 Optane 做定制化的软硬件结合的设计。包括像阿里云的 PolarDB，还有像一些缓存数据库，都在使用 Optane 的这种低延时和持久化的能力来设计数据库软件逻辑。

上图右边是网络延时，分为几个比较典型的场景：

- 最上面是 RDMA 网络。在做软硬件结合设计的时候，RDMA 网络非常有价值，它的延时在 20 微秒左右。比较典型的是一体机，在一个柜子里面把计算节点和存储节点通过 RDMA 网络把它连起来，这个时候通过低延时是能够解决扩展的问题和响应延时。
- 第二个是同 IDC 的网络大概在 200 微秒，这个是我们常见的应用程序和数据库交互的延时。
- 接下来就是 2 毫秒左右的同城跨 IDC 的网络延时，同城容灾的典型场景。
- 1000 公里左右跨城的网络，大概在 20 毫秒，比较典型的场景是异地多活和容灾，需要去解决的网络延时的问题。
- 最长的是全球的跨国网络，像中美大概有 1 万公里通过海底光缆连接，这个是全球多活需要去解决的问题。

所以每个场景它需要去解决的问题是不一样的，因为它的网络延时时差非常大。RDMA 网络其实是今天在软硬件结合里面非常好的一个点。

## (五) 数据库发展趋势（数据库大数据技术一体化）

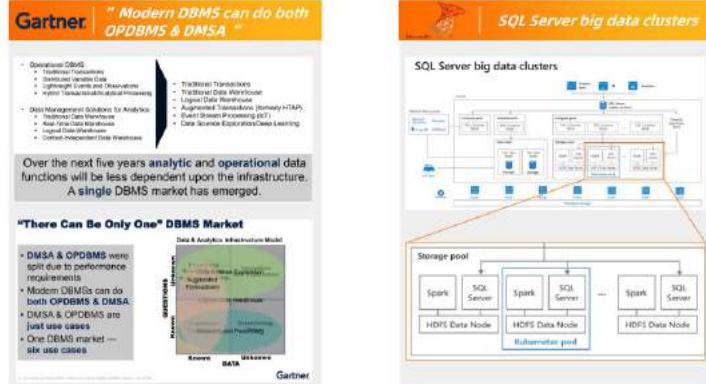
## 数据库发展趋势（数据库大数据技术一体化）

PostgresConf.CN & PGConf.Asia 2020 | 阿里云 | 集训会全称指定云服务商



"Modern DBMS can do both OPDBMS (Operational DBMS) & DMSA (Data Management Solution for Analytics. A **single** DBMS market has emerged"

"DMSA and OPDBMS two MQs will be merged into a single Cloud DBMS MQ"



这个是来自 Gartner 判断，业界技术上也是往这个方向去演进，Gartner 非常明确的提出这个观点，在今年的魔力象限报告里面，已经把在线数据库、数据分析/数据仓库合并成了一个魔力象限，统称为 Cloud DBMS(云数据库魔力象限)。

它的观点非常明确，第一是”云”，云就是未来数据库的发展方向；第二个是数据库、大数据会一体化发展，只有一种数据库领域。

微软的 SQL Server 已经是这个思路，SQL Server 容易理解为它是一个比较标准的 OLTP 数据库。但在 SQL Server2019 里面有很多这种大数据的技术应用，包括像它集成了 Spark，可以访问 HDFS 等等一些外部大数据领域的技术，SQL Server 是一个非常典型的例子。

## 为什么会数据库大数据技术一体化？

### 数据库发展趋势（数据库大数据技术一体化）

PostgresConf.CN & PGConf.Asia 2020 | 阿里云 | 集训会全称指定云服务商



从大数据技术诞生来讲，从最早 Google 提出的三篇论文，大概是在 2003 年到 2006 年这几年像 BigTable、MapReduce 编程模型以及分布式文件系统 GFS，是大数据技术的诞生初期。

- 在 2006 年到 2014 年，称为探索期。出现了百花齐放的情况，特别在 NoSQL 领域出现了非常多的产品，比如从 Big Table 演变的开源的实现，像 HBase，还有比较早期的 Cassandra 的版本，都是在探索。

在编程模型方面，早期大家在使用 MapReduce 的编程模型，但是后来大家发现编程模型效率太低，描述起来远远没有 SQL 方便。因为对于开发者来讲，SQL 其实是非常友好的编程语言，所以 Hadoop、Spark，都推出了 SQL 接口。

在分布式文件系统方面，GFS 在开源里面有 HDFS 的实现，以及出现了分布式的文件系统 Ceph。

- 从 2014 年之后，大家开始审视大数据技术的问题，很多大数据相关的技术，要真的在应用系统里面全面应用，还是要融合很多数据库的技术进去。很多数据库的产品，也在使用大数据的技术进行整合和研发。

比较典型的像 NoSQL 慢慢的出现了发展到 NewSQL 这样的方向。

分布式数据库都属于 NewSQL 这个领域，像 Google Spanner、Cassandra 在后面的版本 V3/V4 还有国内比较有名的 TiDB，都属于偏向分布式数据库方向。

在编程模型方面，大家已经把 SQL 作为最重要的一个编程实现，开发原生 SQL，而不是像以前只是做一个 SQL 接口，用原生 SQL，开始在里面加存储过程。像 SparkSQL，还有 Snowflake 都在支持存储过程等等。

在实现上的话更强调数据实时计算可见，而不仅是 batch 运算。

另外在底下的分布式文件系统，慢慢的大家看到，其实云厂商在这块其实做得非常快，都推出了云原生的分布式存储，有对象存储、块存储等等，像 AWS s3、阿里云的 OSS 和云盘等等，这些都是非常方便的分布式存储。

有了分布式存储，再构建上面的分布式计算就非常方便，而在 10 年前没有好的分布式存储，要做这件事情其实是非常困难的。再加上存储计算分离的架构，已经把数据库和大数据的技术融合，技术门槛下降非常多。

像 TiDB 它是 NewSQL 的代表，同时也融合了很多 Oracle 这样的技术。另外像 Azure 的 SQL DW ( Synapse ) 它把企业数据仓库和大数据分析结合在一起，这是它产品描述里面的信息。

## (六) 数据库发展趋势 (智能化、自动驾驶)



简单来说自动驾驶是用汽车来比喻，它提供了数据库的自诊断、自修复、自优化和自安全这样的能力。

图中截取了在阿里云上的数据库自治服务 DAS 的一个产品，它是阿里云上做自动驾驶的一个能力实现。

如果你在阿里云上购买了一个数据库，并且开启了自治服务，就可以享受到很多自动化的能力，可以让 DAS 系统自动的创建和删除索引，它会根据你的 SQL 负载以及 SQL 类型去自动的选出最合适的索引；同时它能够做到自动限流，比如当有黑客攻击，或者说你自己做的大促营销，这个时候如果你的系统出现了瓶颈，比如说 CPU 利用率大于 80%，活跃会话非常多，它可以做到自动限流，把一些高负载的 SQL 直接限住，这样可以让你的数据库不会受到太大的影响，也留给你更多的时间来做故障或者问题的处理。

第三个是自动扩容，它可以根据你的 CPU 使用的比例，或者说存储空间的使用情况，能够自动的去扩展 CPU、扩展存储空间。

如果说你的负载又降回来了，这个时候它也可以自动的回缩，这些功能都可以通过配置实现。

这里面使用了很多机器学习和专家服务的数据库专家经验，**数据库的自动驾驶未来一定会演变为全自动驾驶**。



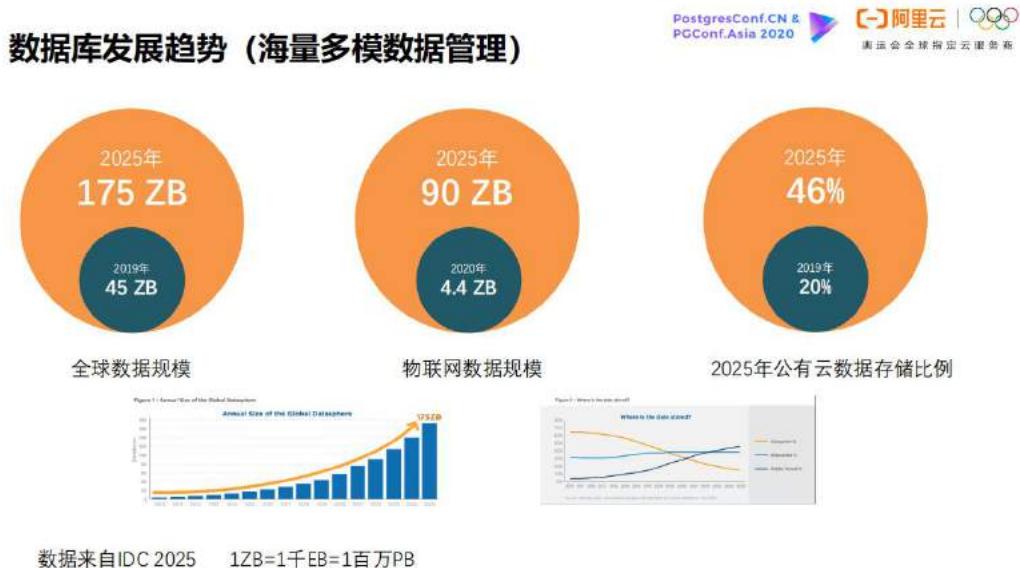
把数据库运维分成两个阶段：

- 在我们比较早期，像监控告警，还有诊断建议，都是我们比较传统的运维，DBA 其实是非常擅长的，我们可以把它理解为汽车里面的传统汽车，就是你完全是需要驾驶员去控制，你可以看到屏幕，看到你的车速，如果说汽车有一些故障，它也会给你一些告警，但是他不会去帮你去做自动的决策处理，全部是需要驾驶员来做，这个是传统运维解决的问题。
- 今天自动驾驶这个方向，它其实可以做到很多的自动化的能力，包括刚才给大家介绍的自动的负载限流，自动优化 SQL、自动索引创建等等，这些都是自动优化故障修复相关的内容。
- 然后像自动的容量评估和扩缩容，还有自动的安全保护，当出现安全攻击的时候，它能够去防护 SQL 注入，这些相关的能力全部自动完成。这个就有点像今天特斯拉，它其实是已经开始迈入到一个自动驾驶的基本能力了，但还不是很高级。

未来数据库自动驾驶，它一定是会越来越强，它可以把各个场景逐渐的去突破。

今天可以看到 Oracle 它也在抓这个方向，它提出的数据库自治服务，阿里云的数据库自治服务 DAS，以及像 Azure SQL，它们都提供了非常多的这种自动驾驶的能力。

### (七) 数据库发展趋势（海量多模数据管理 1）



这个数据是来自 IDC 的报告，IDC 预估 19 年全球大概有 45ZB 的数据，1ZB 大概是 100 万 PB，这个数据会快速增长，预估到 2025 年有 175ZB 的总数据。

在物联网领域，在 2020 年大概是 4.4ZB 的数据，预估到 2025 年会产生 90ZB 的数据，物联网是未来数据增长的一个主要来源。

在这些数据存储到公共云会快速增长。在 2019 年大概 20% 的数据是存储在公有云，预估在 2025 年会有 46% 的数据存储在公有云，还有很多数据可能会存储在私有云等等。

### (八) 数据库发展趋势（海量多模数据管理 2）



## 为什么会产生这么多数据?

来源有很多，不管是 IoT、车联网、机器人、设备数据、手机数据等等，这些都可能产生很多数据。

总体我们把它归类为两类：

### 1. 结构化的数据：

结构化数据，比较常见的就是 PostgreSQL 里面或者 MySQL 库里面管理的一条条记录，这个我们称它关系型数据。然后还有像 JSON，还有 XML，这些以前大家把它定义为半结构化数据，但是从今天的数据库技术来看，它其实已经是非常结构化了。JSON、XML 都是数据非常标准化，应用程序很容易处理。

还有图数据（图是图形,就是有关联关系的图形，不是图片的意思）、时序数据（就是每个时间打一个采集点，这种在监控里面会看的比较多）、空间数据以及向量数据（是其他方面计算出来的多维数据），这些我们统称为结构化的数据。

### 2. 非结构化数据：

未来主要的增长是来自于非结构化数据（超过 60%都是非结构化数据），比较典型的一个是日志，这些是一些文本信息；另外一个是文本（常见的一些大片的文本）、图片、还有语音和视频以及通用文件。比如说我们一个 PPT，它其实也是属于一个数据，但是它放在文件里面，但是我们今天还没有很好的技术手段来对它做检索中，把它统称为非结构化的数据。

非结构化数据，通过我们的计算是可以把它抽象出结构化的数据。

比如说日志，我们通过计算，能够把它转换成表格数据，图片、视频也是能够抽取出来，形成向量数据。

这些结构化数据和非结构化数据，未来都可以加工，然后通过智能化 AI 这样一些技术做智能的数据分析

这个是未来多模形态，我们称它为多模海量数据管理，它是很重要的一个发展趋势。

目前阿里云有灵动 ( Lindorm ) 这样一个产品，Azure 有 Cosmos DB 等等，基本上各个大的厂商都在投入这一块。

### 三、总结

整体上来讲数据库发展技术可以归为以下几点：

1. 数据库大数据一体化；
2. 云原生+分布式；
3. 智能化和自动驾驶；
4. 海量多模数据管理；
5. 软硬件一体化；
6. 安全可信。安全可信没有展开来讲，但这块也是所有的数据库发展里面非常重视的一环，这里面挑战非常的大。现在在可信计算还有全链路加密方面，投入非常大，未来对于整个数据库发展也是一个基石。

# 云原生数据仓库的机遇和挑战

作者 | 占超群 阿里巴巴集团研究员 数据库 OLAP 产品部负责人

## 一、数据仓库的发展历程

### (一) 业界趋势



可以看到，目前业界中数据生产和数据处理正在发生质变，其主要趋势有：

1. 数据规模爆炸性增长：2020 年，全球数据规模约 40ZB，预计到 2025 年全球数据规模相比今天将会增长超过 4 倍；
2. 生产/处理实时化：到 2025 年，实时数据的占比会超过 30%，且到 2022 年，预计 50% 的业务将会采用实时分析；
3. 生产/处理智能化：随着摄像头和 5G 的兴起，未来的非结构化数据增速在加快，并且占比越来越高，会超过 80%，非结构化数据年增速将达到 55%；
4. 数据加速上云：到 2023 年，75% 的数据会上云，到 2025 年，49% 的数据会直接在云上存储。

随着业务的变化和数据的质变，数据处理将会面临新的挑战，主要有：

- 数据一致性：平均一个客户的数据源会超过 5 种；
- 分时实时性：分析的实时性要求越来越高；
- 系统复杂性：基本上一个大数据系统将会包括超过 40 个组件，复杂性急剧增加，运维成本越来越高；
- 学习成本：各种组件之间的组合、使用等难度越来越大，学习成本将会不断增加。



通过前面数据和数据处理的趋势以及数据业务面临的挑战可以看到，这几十年来，数据存储和数据处理的技术发生了很大的变化。

- 60 年代，关系数据库之父 E.F.Codd 博士提出了关系模型，促进了联机事务处理(OLTP)的发展，诞生了如 Oracle、DB2 等数据库帮助核心业务如银行实现在线交易的普及。
- 1993 年，关系数据库之父 E.F.Codd 博士提出多维数据库、多维分析的概念以及十二条准则，认为 OLTP 已不能满足终端用户对数据库查询分析的性能需求，SQL 对大型数据库进行的简单查询也不能满足终端用户分析的多样性要求，促进了在线分析处理(OLAP)的发展，出现了 MOLAP(Multidimensional OLAP)、ROLAP (Relational OLAP )、HOLAP ( Hybrid OLAP ) 计算模型和引擎，诞生了如 IBM Cognos、Oracle Essbase、Greenplum 等数据仓库帮助业务实现海量数据存储、建模、业务分析探索的普及。

- 2003~2006 年，Google 发表《The Google File System》、《MapReduce: Simplified Data Processing on Large Clusters》、《Bigtable: A Distributed Storage System for Structured Data》三篇海量数据存储、处理重要论文，促进了大数据技术的飞速发展，诞生了如 Hadoop HDFS、Hadoop MapReduce、Tez、HBase、Spark、Flink 等为代表的分布式文件系统、分布式计算框架、分布式宽表存储系统，加速了大数据应用向 5V ( Velocity、Volume、Variety、Value、Veracity ) 方向发展和普及。
- 2012 年至今，随着云计算的发展，云计算的资源池化、存储与计算弹性扩展等基础设施升级以及计算存储分离、在离线一体化等技术创新，促进了数据处理开始朝一份数据开放计算、存储计算分离的云原生方向演进，诞生了如 Snowflake、AWS Redshift、AWS Aurora、AWS Athena 为代表的新一代云原生数据库、数据仓库、数据湖，加速了数据处理向在线化、在离线一体化、结构化与非结构容和处理演进，加速业务走向数字化、数智化创新的新形态。



通过上图所示的发展历程，我们可以看出，云正在重新定义数据库和大数据。云计算的本质是什么？它本质是资源的高效池化。计算机的本质是什么？计算机就是解决存储问题和计算问题。那么今天数据库和大数据的传统的形态什么呢？就是处理数据的生产、存储和消费。

通过云计算的资源高效池化和分布式扩展能力，今天我们可以实现一种新的 Cloud Native Data Warehouse 来解决数据规模的问题、数据实时性的问题、数据的复杂计算问题以及数据成本的问题。总得来说，是通过存储计算分离、分层存储、计算弹性和数据库与大数据一体化的方式来解决上述这些问题。

## (二) 阿里巴巴在线分析/数据仓库的发展演进历程



数据的质变和数据业务挑战以及整个业界的发展，其实在阿里巴巴也得到了很好的验证和实践。如上图所示，其实 2008 年之前，阿里巴巴的 Oracle RAC 已经是亚洲最大的 Oracle 集群。

到了 2009 年，我们开始尝试用 Share-Nothing 的分布式方式看能不能解决数据规模和查询性能瓶颈的问题，因为当时 Oracle 的架构很难达到我们当时那种数据增速下的处理能力，但是依旧有问题，比如说我们面临的有高并发查询的问题、高可用的问题、高并发写入的问题。到了 2011 年我们开始尝试用开源的生态来做，我们基于 HBase 的 HTable 接口来扩展它的计算能力，也尝试用 MySQL 的分库分表以及 Hadoop Mapreduce 预处理来形成组合解决方案，但是依旧解决不了数据一致性的问题和实时查询的问题。随着我们在规模、高并发、低延迟上的一些挑战越来越严峻，我们开始尝试自研 AnalyticDB（简称 A DB）。

这个时候我们可以解决几个问题，第一是规模的问题，可以到 PB 级的规模，第二我们可以支持很高的并发，第三可以把延迟降到毫秒级，第 4 个来说，作为互联网产品，高可用是基本要求。

随着这几年的发展，在 2015 年，我们开始全面的把能力开放到云上作为普惠服务。到了 2019 年我们开始以云原生为基础和核心策略进行大规模架构重构，包括把规模进一步增大，至少能处理百 PP 级别，同时我们支持实时的增删改查，这样的话就可以让传统的数据处理无缝地迁移到我们的系统。

到了 2020 年，我们的整个系统全面 Cloud Native 化，无论是我们的 ADB for MySQL，还是我们的 ADB for PostgreSQL 以及我们的数据湖的产品，并逐步的走向 Cloud Native+Serverless，整个存储是 Serverless 的，计算是弹性的，同时我们也在尝试支持云原生时代的大规模分布式 HTAP。

### (三) 下一代数据仓库关键技术

结合数据的质变、数据业务的挑战和业界的发展趋势，我们可以看出下一代的数据仓库的关键技术主要包括：

- HTAP：数据库大数据一体化的技术，包括行列混存、混合负载、分布式计算与分析等；
- 云原生+分布式：包括 CPU/内存/存储分离、Concurrency Scaling 等云原生技术、分布式存储与分布式计算技术；
- 智能化：包括自感知、自决策、自恢复、自优化等技术；
- Multi-Model 多模：支持多样数据的技术，比如 JSON/KV/Text/Vector 等数据类型；
- 软硬件一体化：在云上用户不关心硬件的形态、部署的形态，更关注服务的形态，因此软硬件一体化就显得非常重要；在我们实验环境采用 GPU 加速可以实现性能提升 10 x 以上。
- 安全可信：安全的重要性不言而喻，主要包括可验证日志与计算、全链路加密技术等。

## 二、云原生数据仓库核心技术

### (一) 数据处理分析维度

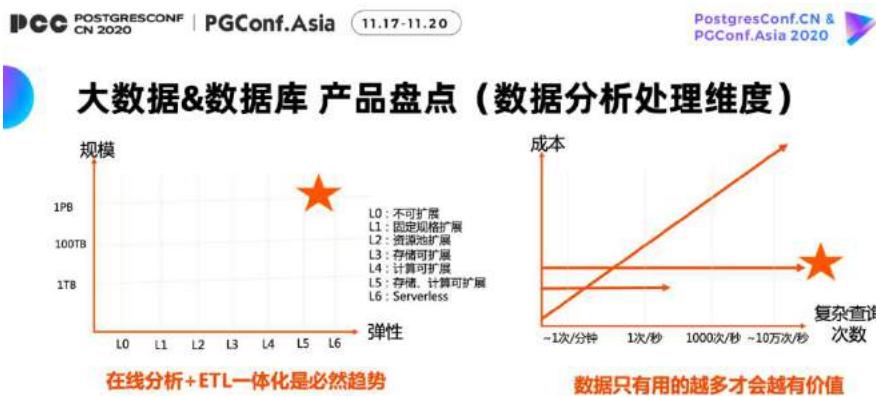


如上图所示，可以看到整个数据处理的产品其实非常多的，我们该怎么选择呢？其实有几个维度：

第一，查询计算延时 vs 数据规模的维度。不同的产品在这方面面临的角度是不一样的，比如说像传统的 OLTP 在规模很小的时候，它的演示很低，但是如果规模一大，延迟就增加的非常快。怎么样把这个产品做大，在规模和延迟间取得平衡，这是面临的一个挑战。

第二个是灵活性 vs 数据规模的维度，比如说传统的 Oracle ESSBase、IBM Cognos 等，在灵活性和规模之间有一定的 trade off，它会解决一个灵活性弱但是规模大的问题，主要是通过预先建模，但是传统的数据仓库不进行预先建模，而是实时的计算。

第三点来说数据实时性 vs 查询性能的挑战，比如很多产品性能很高，但是数据延迟很大，可能有的数据是一天前的，或者通过流计算预建模的，如果既需要数据实时又需要高性能在线查询怎么去处理？可以看到在性能和写入实时上有很多的折中方案，比如说 Spark Delta Lake 是在做异步的 Write On Merge 才能保障基本查询性能。这里面很多词是大家容易混淆的，比如说 CAP、BASE，还有 ACID，针对数据处理和分析，其实我们都是要去深入分析和思考的。



第四点，随着规模的越来越大，成本要求也越来越高，这时候要提供什么样技术来降低成本呢？存储计算分离、弹性是关键，这里面不同的弹性方式和分布式服务之间也有一些不同点，有的是横向的扩容(Scale out)，CPU 内存和磁盘一起扩，有的是纵向的扩容(Scale up)，比如扩内存、扩阵列存储，但是未来的形态应该是 Serverless 的，存储可以无限扩，计算可以无限扩，并且扩展是按需进行的，甚至可以做到 Query 级别，这是云原生的趋势。

最后一点，就是成本，大家可能认为传统大数据技术方案的成本足够低，数据仓库的成本比较高，其实是需要仔细分析的，传统的大数据是把数据先写进来，不去做任何处理，数据仓库和数据库呢？它先建索引、数据预处理、Cache，提升性能，所以它的初始化成本会高一些，但是查询成本很低，因为查询可以复用索引以及 cache。从图中可以看出，随着整个查询次数越来越多成本越来越低，业务越来越在线，必须有新的技术形态，高成本初始化但是低成本查询才是未来。

## (二) 阿里巴巴的云原生数据仓库实践

从下图可以看出阿里巴巴的产品主要分为如下几大部分，第一是数据湖，它主要是做异构数据的处理，包括以 Serverless 为核心的 SQL 查询、Spark 计算、机器学习、图处理和流处理，在数据做完 ETL 之后，会回到我们的数据仓库。数据仓库有两个产品，一个叫 AnalyticDB for MySQL，主要提供了超大规模的高性能处理能力，兼容 MySQL 生态；还有一个是 Analytic for PostgreSQL ( PG )，强兼容 PG 和 Oracle，能实现 HTAP 一体化的能力，能支持 UDF 以及事务。底层可以接各种各样的数据源，对接各种各样的开发工具，还有我们上层的各种应用生态也可以对接，这样的话可以形成一个完整的从数据采集、同步、加工管理和上层应用一体化的生态。从图中可以看出，云原生数据仓库 AnalyticDB 是其中的核心。

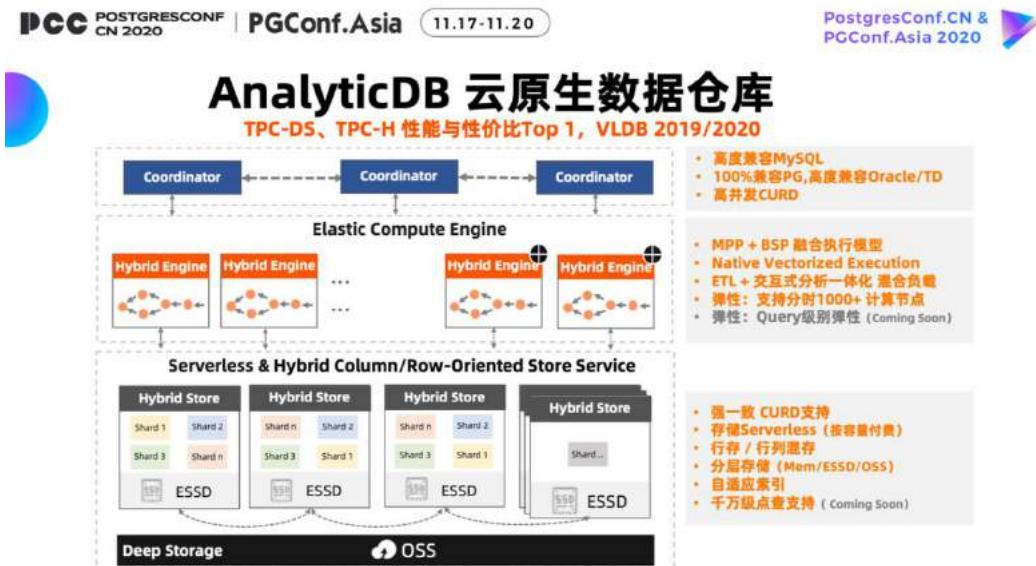


### (三) AnalyticDB

下图所示是 AnalyticDB 的架构，基于云原生的存储计算分离、弹性模式，主要分为存储层、计算层、接入层：

- 存储层采用分层存储架构，底层是一个持久化低成本存储，完全基于阿里云对象存储 OSS，成本可以低至 0.144 元/GB/月；基于 OSS 会将数据进行分布式写入、自适应缓存、管理，每个 Shard 里的数据采用 RAFT 协议实现实时强一致复制；再上面我们会采用行列混存、自适应索引技术提供高性能写入和检索能力，且将整体能力采用 Serverless 服务模式提供对外服务支持开放的计算；
- 计算层采用在离线一体化（MPP+BSP）技术实现交互式分析、ETL 处理一体化，计算层采用 CodeGen+JIT 技术以及结合 Vectorized Execution 实现高性能执行，采用 Cascades CBO 实现优秀的执行计划选择；采用资源组隔离以及自适应混合负载技术实现多种计算场景的混合负载支持；整体的执行引擎可以按需动态扩展实现低成本支持波峰波谷的计算能力。
- 最上面是接入层，原生实现了高度对 MySQL、Oracle 协议的兼容以及 100% 对 PG 协议的支持；采用单机数据库的体验（CURD 支持）支持大规模数据写入和分析。

整个产品目前有两个形态，兼容 MySQL 和兼容 PG 版本，融合了数据库和大数据一体化技术，实现了 MPP+BSP 融合执行模型，还有向量执行、混合负载等，可以实现上千个节点的弹性能力。



云原生数据库的发展趋势是数据库和大数据一体化，主要解决三类问题：

1. 多维分析：支持任意列的 Join 和复杂长计算任务、ETL，典型的技术如传统的 Hive、Impala 等；
2. 明细查询：如何提供高并发点查，典型的技术包括 Apache Kudu、HBase、MySQL 分库分表；
3. 实时查询和写入：支持实时的增删改查（CURD），其面临的挑战非常大，因为其 QPS 很高（百万级），每秒可能写入上百万到上千万数据，一般做法是通过数据库的分库分表来完成。

那么，上面的三类问题传统解决方案是数据复制三份采用三类系统来解决，有没有可能一起解决呢？答案是可以的，我们可以基于云原生为基础通过如下图所示的四个技术来一体化解决这三个问题：通过行列混存+Multi-master 来解决明细查询和多维分析的性能和实时性的问题；然后通过混合负载和资源组以及融合执行引擎来解决多维分析、简单查询性能问题以及负载问题；通过智能索引和智能压缩来解决明细查询和实时查询写入的性能问题。

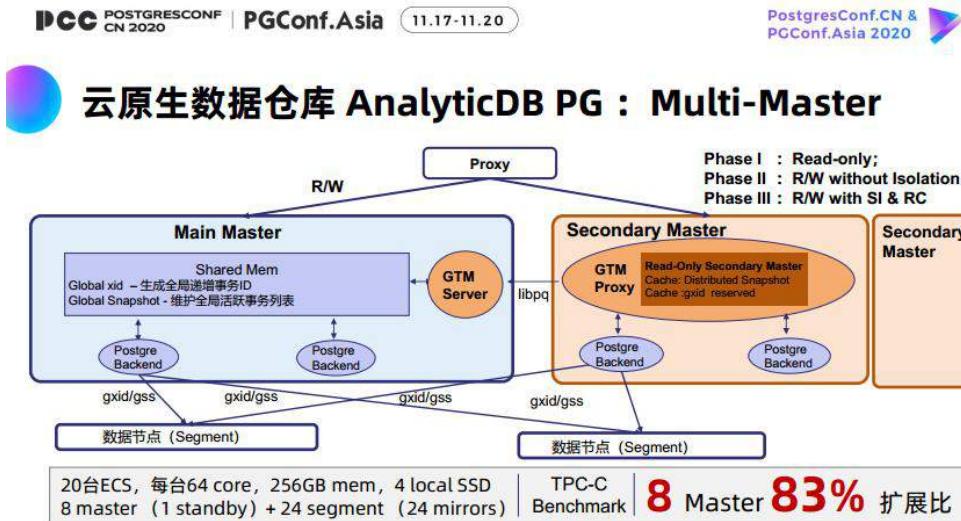


下面介绍其中的几个关键技术点：

### 1 ) Multi-Master

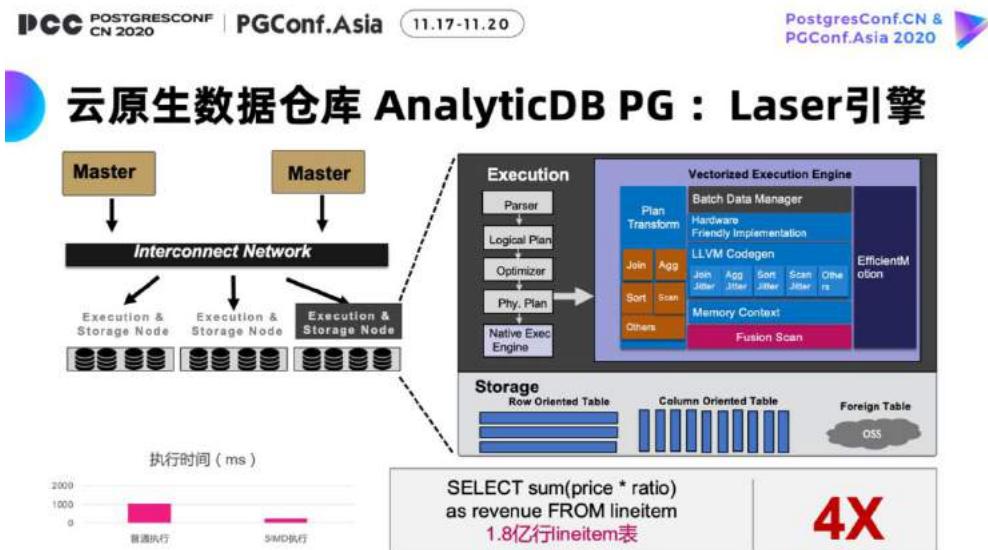
Multi-Master 主要解决的是海量数据场景的实时增删改查问题，其主要技术如下图所示，整体原理是采用 2PC (二阶段提交)，采用 GTM (Global Transaction Manager)

来支持分布式事务管理和协调。为了提升并发性能和线性扩展比，前端采用 SLB(负载均衡服务)实现接入层路由；采用 GTM Proxy 缓存 GXID、Snapshot 数据来减少网络通信交互，最终实现写入和查询的高性能线性扩展。



## 2) Laser 引擎

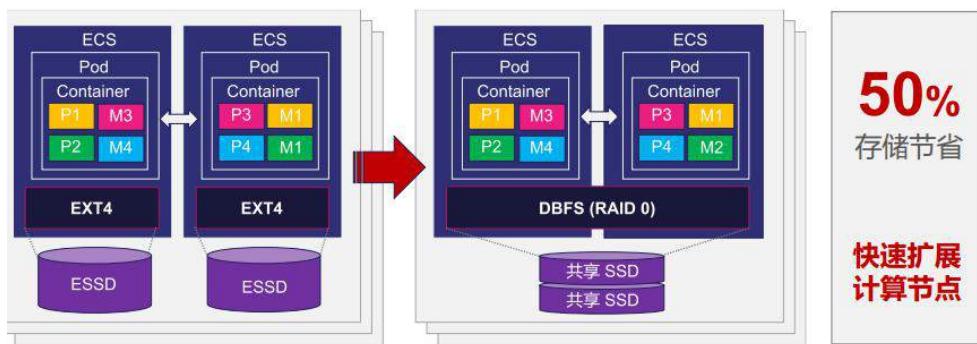
Laser 引擎是我们完全自研的执行引擎，其架构如下图所示，可以看出其是基于向量化执行和 Codegen 来实现的，首先会将 SQL 翻译成 AST 抽象语法树，基于 Cascades CBO 优化器生成最优执行计划（Plan），然后会将 Plan 生成 IR Code，针对不同的硬件（X86/ARM/GPU）生成最优的执行机器码，CodeGen 支持大部分算子包括 Join、Sort、Scan 等，整个性能的提升非常明显。



### 3 ) 云原生存储

第三个技术是云原生存储，云原生存储解决什么问题呢？主要就是传统的数据仓库和数据库基本都是主备复制架构，主库、备库都是各自物理存储，无论是同步还是半同步还是异步，都会面临三个问题：成本问题、数据一致性问题（RPO）、恢复时间问题（RTO），尤其在数据仓库的场景，规模很大，都是PB级，这个问题更进一步放大。后来我们开始尝试用云原生的形态来处理，采用共享存储（ESSD+DBFS）来提供整个存储文件系统服务，可以达到大幅节省存储成本、RPO=0以及极低的RTO（秒级切换）。另外因为我们的存储是集中式的，可以保障我们整个节点拉起非常快速，并可以在超大规模的场景快速拉起多个计算副本，提升IO和计算能力。

## 云原生数据仓库 AnalyticDB PG : 云原生存储



### 4 ) 极速存储引擎

存储引擎采用行列混存技术，可以根据实际情况自由选择。在列存储场景会面临的一个问题就是如何做检索和高并发点查，尤其是需要做各种复杂的过滤条件的时候，传统做法就是建立索引，但是索引在列多的时候会阻塞写入，并且建索引的成本很高。后来我们使用了多维的索引聚集技术，采用 Rough Sets、Character Map 和多维任意列聚集排序，在很多场景可以实现超过 60 倍的性能提升，其核心是粗糙集（Rough Sets）技术的使用，可以实现不需要索引的时候快速过滤掉不需要计算的数据 Block，如下图所示。

第二个技术就是在数据越来越大的时候，我们使用了自适应的压缩技术，融合了多种压缩算法，去降低数据存储成本。压缩既要考虑压缩/解压速度也要考虑压缩率。通过自适应压缩可以让整个存储成本下降超过 40%，再结合字典压缩，在典型场景中的存储成本下降了 8 倍。

## 云原生数据仓库 AnalyticDB PG : 极速存储引擎



### 5) DB4AI

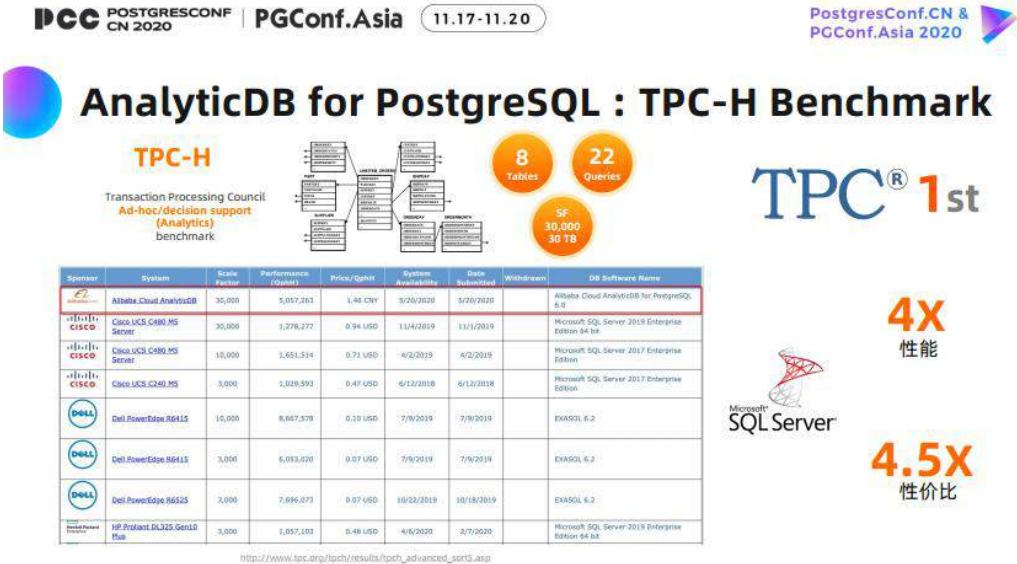
随着非结构化数据的爆炸性增长，比如像图片、语音、视频还有文本等非结构化数据，这些数据增长非常快，业务的需求也很多元，需要跟结构化数据做融合。

如下图所示架构，在底层我们把非结构化数据转为向量（Vector），然后建立多种索引，然后做数据和节点的 Sharding，对非结构化数据做 Sharding 是一个很有挑战的事情，在 Sharding 之上提供统一的 SQL 查询；并且我们还支持用户自定的 UDF 和机器学习算法来做向量的索引和检索能力，同时我们还充分利用了硬件能力来做加速提升 10x 以上性能，最终实现对向量的高效索引与分析，实现了利用 SQL 和 SQL 扩展能力来实现结构化数据和千亿级非结构化数据的融合和分析。

## 云原生数据仓库 AnalyticDB PG : DB4AI



这么多年的技术研发和实践，我们取得了非常不错的成果，其中一个就是今年5月份在全球权威评测机构TPC发布的TPC-H榜单上取得了不错的成绩，在测试集上我们取得了全球第一的成绩，相比Microsoft SQL Server 2019在性能上提升了四倍，性价比提升了4.5倍，这也是中国厂商第一次取得如此优异的成绩。



### 三、最佳实践和案例

下面是几个大型实践和案例。

#### (一) 某央企大型传统商用数据库替换

该案例中某企业需要将全省甚至全国的数据进行集中，之前的数据是分布在各地的，没有全局数据，无法做详细的分析能力和业务调度，只能采取汇报和上报数据的方式，其面临的一个问题就是如果某个业务的数据是跨区域的就无法进行联合分析和业务协同，后来采用我们的AnalyticDB，实现了全国所有数据的大集中，性能大幅提升，实现了分析的实时化、全国统一业务调度和客户精细化管理，且有着不错的兼容性和研发效率。



## 某央企大型传统商用数据库替换

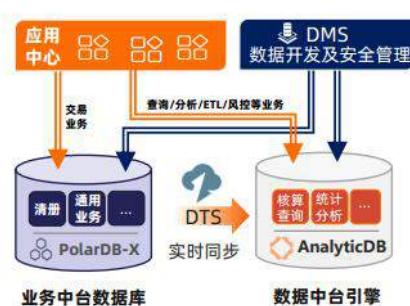


## (二) 某超大型部委客户

该案例是某超大型部委客户，也是第一次实现全国大集中，通过与 PolarDB-X 形成大规模实时事务处理和分析，实现了从交易型的数据实时无缝同步到数据仓库，在数据仓库中实现最核心的业务逻辑，可以帮助业务人员实现实时、多样的复杂精准核算和测算。该系统支持高并发、低延迟的复杂查询服务百万级营业厅工作人员，支持海量数据实时数据实时可见和高效入库，支持金融级别的精准计算。



## 某超大型部委客户



- 支撑高并发、低延迟的复杂查询：
  - 8亿级 普通用户 和 100万级 工作人员
  - 60表Join, 3000行SQL
  - 支撑基于**千亿级数据**的**复杂查询**在**秒级**内响应
- 支持海量实时数据实时可见和高效入库：
  - 在线交易的实时数据(**百亿级**) CURD(增删改查)
  - 月末统计等分析数据(**千亿级**)在**小时级别**内同步
- 支持金融级别的精准计算：
  - 计算的准确性 (1000+位Decimal)
  - 提供针对各种金融运算函数的**全面支持**

### (三) 某部委典型客户

该业务其实是一个大数据平台业务，要把以前客户的各种场景的数据，比如 Kafka、Hadoop、Oracle、MySQL 分库分表中的数据都统一到我们的 AnalyticDB 数据库中，来实现在线业务的升级，包括传统的报表业务、可视化业务、研判业务以及一线的各种工作人员的日常数据任务，实现了非常高的并发能力，QPS 过千，实现了在线、离线业务的融合，性能也有了大幅提升，这是之前不敢想象的。



# 传统数据库异构上云实践

作者 | 韩国盛 阿里云数据库高级产品专家

我们先来看看数据库市场的情况，根据数据库引擎流行图综合排名来看，目前 PostgreSQL 稳居前十，排名第四位。前三位分别是 Oracle, MySQL, Microsoft SQL Server。关系型数据库今天已经形成了两商用加两开源，并驾齐驱四天王的整体格局。Oracle 数据库仍然是流行度和市占率最高的数据库。PG 近两年均保持热度上升，如果保持现在的增长趋势，PostgreSQL 则有可能在 4~5 年间接近 SQL Server 的流行度。目前国产数据库多以 PG 数据库引擎作为底层引擎，PG 在中国将迎来历史性的爆发和增长，目前正是投资 PG 和学习 PG 的大好时机。



说到传统数据库，异构上云存在三座大山。我们发现目前异构迁移普遍存在以下几个困难。

第一，决策难。在异构迁移的观察中，我们发现目标数据库引擎众多，企业要求数据库既要通用性，又要扩展性，还要多模数据处理的便捷性；既要高并发，又要实时复杂分析。然而传统数据库无法满足以上需求。

第二，投入大。企业有非常严重的历史包袱，如果原有的团队和业务的技术栈都是传统的 Oracle 技术栈，适应其他产品的周期长。迁移如果涉及到大量的代码改造，周期长，风险高，收益低。通常目标引擎数据库和 Oracle 数据库的兼容性非常的差，用户需要大量的改造。

第三，风险大。多年积累的业务逻辑被大面积修改及重构，除了带来非常大的工作量外，也同时带来很大的业务不确定性。同时由于缺乏有效的迁移方法和迁移工具，迁移改造工作量也很难评估，迁移周期都非常的长。别人的成功迁移往往也很难进行复制。



面对以上的三大挑战，我们希望通过构建新价值和生态价值链，降低成本和风险。

首先要构建新价值。商业价值：数据库服务化改变企业 IT 投资预算模型，提升企业资金利用率。业务价值：数据库服务化帮助客户提升业务迭代速度，降低试错成本，聚焦业务价值提升。

然后是丰生态。开发生态：以行业为抓手构建行业开发生态。人才生态：积极与社区合作，产学研联动，构建完善的人才培养机制。服务生态：广泛吸纳传统数据库服务合作伙伴。

最后达到降低成本和风险的效果。决策成本：量化技术风险，ROI 投入产出比。迁移改造成本：提供高度语法及特性兼容，一键迁移。运维成本：云托管自助化运维降低重复工作，帮助企业聚焦高价值工作。



这是一张抽象出来的传统数据库建设，以及云上服务化开通的对比图。从这张图的左边我们可以看到，传统自建数据库涉及到从机房、硬件、软件、网络等各种基础设施的采购、安装配置、调优等多部门的配合。建成后，还需要专职的 DBA 团队来进行管理及运维，工作繁杂，而且建设周期长。从企业的角度来说，正是由于其链路长、周期长，也变得无法敏捷的去应对市场环境的变化。而这张图的右边，一键开通数据库服务，即买即用，企业无需再关心大量的基础设施，而且有阿里云专业的数据库团队，降低企业运维的负担，企业可以聚焦于业务价值，快速的去响应市场需求。



除了前面提到的优势以外，看看数据库云上服务化的关键能力。

第一个是开通和管理。云上数据库提供数据库服务的自助一键开通，自动化运维，补丁升级以及日常的数据库集群管理的服务。有监控、告警等，满足企业数据库日常运维 90% 的需求。

第二是架构。云上数据库提供了主备的高可用、一写多读、跨机房和跨地域的容灾异地备份以及恢复，可以满足企业业务连续性及数据安全性的需求。

第三是性能及伸缩性。原数据库的一个基本假定，就是底层的计算和存储的资源化和标准化。通过软硬一体的优化，使用最新的硬件技术提供高性能的同时，提供分钟级的变配及增删节点，满足业务的需求及成本的平衡。

第四，开发。云上的数据库服务与自建数据库提供一致的开发使用体验，完全满足企业客户的业务开发、数据分析等等的需求。

第五，数据安全。云上的数据库服务提供完备的数据库用户权限管理、操作审计，提供网络安全白名单保护、

数据透明加密和链路加密、以及更多的企业安全特性，来满足企业对于数据安全的重视和需求。



云计算厂商在互联网时代都加大了对数据库技术的研发投入，也找到了一条自主研发的新道路。

云原生数据库融合了云计算的服务能力和弹性架构，开源数据库的简洁应用和开放生态，以及传统数据库的售后管理和处理性能等各方面的优势，通过融合创新，弯道超车，在云环境下能够为用户提供更好的数据库服务。

云原生的数据库是为了更好的服务于云环境下的应用而诞生的，它是一种融合了众多创新技术而跨界的云数据库服务，本质上是云的能力和 SQL 能力的融合。

云原生数据库的一个显著特征是提供满足企业级数据业务服务的解决方案的能力，并把这种方案沉淀成产品能力，提供  $7 \times 24$  小时的高可用服务，提供读写分离的自适应复杂均衡能力，提供故障自动恢复的业务连续保障能力，提供数据存储容量自适应增长的能力，提供计算资源的即时在线扩展的能力，还可以跨区域实现数据的容灾能力。

而云原生数据库从诞生起就具备企业级数据解决方案的能力。云原生数据库的用户共享的不一定是局限于物理服务器的计算资源，更多的共享是超高速的网络环境，以及具有先进预设计算环境的 IDC 机房，还有足够可靠的安全防御体系。在极大的降低了高科技产品的准入成本后，风险与故障成本也被共享。同时我们提供智能化的运维诊断服务，在没有专业的 DBA 的情况下，我们可以观察到数据库在过去任意时刻是否存在性能瓶颈，瓶颈在哪里。即使企业中没有专业的 DBA 也能轻而易举的发现数据库的性能问题。

云原生关系型数据库 PolarDB 依托云原生存储计算分离架构，专注解决大容量、超高并发吞吐、大表瓶颈等数据库瓶颈问题，提供 PostgreSQL 完全兼容版，以及 Oracle 语法高度兼容版。PolarDB 逐渐成为云原生数据库的事实标准，并于 2019 年作为第一作者参与中国信息通信研究院发布的云原生数据库标准。



与传统数据库相反，云原生数据库天然拥有云计算的能力，并且兼具开源数据库的应用开放特点，以及传统数据库的管理和处理性能优势，是云时代下企业数据库的最佳选择。随着移动互联网的发展，数据量巨增，企业上云也成为大势所趋。企业对数据库提出了更高的扩展性和可用性的要求。

2019年，双11电商业务全面使用PolarDB，PolarDB在2019年双11的处理峰值为每秒8700万次事务处理，支撑55万笔订单每秒，冲击世界纪录。



再重点介绍一下PolarDB的重要的技术特性以及业务价值。

第一，提供按需开通，弹性扩收容，以及按量使用计费，相比传统的IT投资预算模型可以有效的降低企业资金占用成本，同时最大限度的满足企业用户业务发展的需要，从而达到降低成本的目的。

第二，阿里云构建了强大的技术团队，包括大量的内核级的研发人员，提供数据库内核层面的优化，使用最新的软硬件一体给我们企业客户提供高性能、高稳定的数据库服务。利用高速网络RDMA和读写分离，提升业务性能，保障一致客户体验。

第三，通过数据多副本，共享分布式存储，和多可用区数据备份，确保企业客户面临各种意外场景下的业务连续性，以及数据的安全性。

第四，阿里云通过自研的分布式存储和弹性扩容，从而应对海量的数据需求，有效提升业务的敏捷性。

第五，兼容主流数据库引擎和标准的 SQL 语法，满足客户多场景的需求，加速业务创新。



在商业模式上面也大胆进行创新。开通数据库服务一般都是预付费模式，而云计算方法中非常重要的两点，是弹性和按量。PolarDB 提供全新的水电计费模式，不再占用企业大量的昂贵的现金流为未来买单，提供存储、计算均按使用量进行计价，用多少付多少钱，提高企业客户资金周转率。而且在数据库容量的扩容上，用户也会有更好的体验。用户会接受按量付费模式，再也不需要为了 10% 的扩容预支浪费成本。



PolarDB 利用云基础设施提供秒级无锁备份及恢复。PolarDB 的快照功能有以下特点。

第一，并行于每个段定期做 snapshot，将 WAL 保存至 OSS（提供永久付费备份）。

第二，持续进行备份，并不影响性能或可用性。

第三，在还原时，从 OSS 返回相应的段快照与 WAL 日志进行还原。

第四，以并行和异步方式应用 WAL 直到数据库恢复到要求的时间点。



企业业务迭代的开发、功能测试、性能测试、数据分析等场景，均需要有高保真的环境。PolarDB 提供克隆能力满足企业开发/测试需求，克隆数据而不复制数据，瞬间创建一个数据库克隆；仅在发生写入时复制数据(COW)–当原始数据和克隆数据不同时。

应用场景包括：

- 克隆生产数据库进行版本开发测试验证。
- 克隆生产数据库进行数据库性能测试。
- 克隆生产数据库为分析一个时间点的数据快照，不影响生产环境。



随着计算机技术的发展，小型机和 EMC 高端存储可以很容易的被高性能的 x86 server 和分布式存储代替。Oracle 数据库以其卓越的性能一直占据数据库王者的地位，使用其他数据库替代的风险很大。阿里的 Oracle 迁移历程可以说是整个阿里巴巴的技术变革。

阿里巴巴集团从 2006 年开始考虑异构迁移，到 2013 年完成最后一套 Oracle 数据库下线，历时 7 年时间，投入超过 1000 人的技术实力空间。

Oracle 数据库和应用异构迁移，在这个过程中，阿里巴巴技术团队研发了许多优秀的数据库产品，最终顺利将整个阿里集团的 Oracle 数据库迁移到其他的数据库上。PolarDB 提供高度 Oracle 语法兼容性。在接口层，ORACLE 原生接口级兼容。

在语法层，包含几乎所有 ORACLE 数据库对象，高度兼容 ORACLE 语法，同时对高度兼容 Oracle 重要特性：DBLINK、分区表、PL/SQL 等特性。在逻辑层，逻辑概念相近，快速理解数据库结构。

在物理存储层，采用 shareeverything 架构，与 ORACLE 保持一致；文件组织结构、ACID 概念与机制接近。



数据库的异构迁移并不仅仅是业务层、系统层、底层数据库引擎的切换，更重要的是应用的适配。

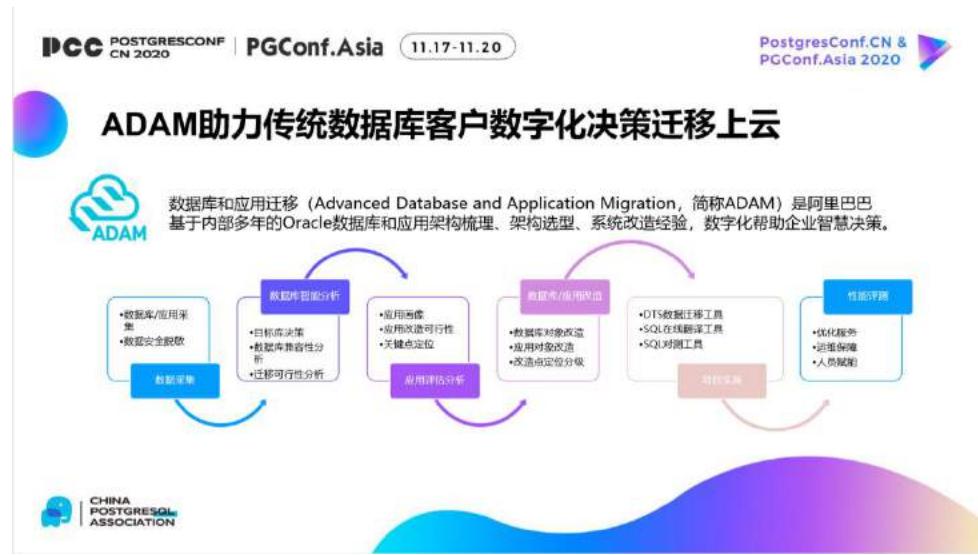
这个过程是相当难的，主要原因是其他的数据库和 Oracle 数据库的 SQL 引擎并不适配，需要做很多改造，在这个过程中存在很多的陷阱。阿里云将阿里巴巴持续多年的 Oracle 迁移过程，整理了一套方法论，并依据方法论研发了一个产品，数据库和应用迁移，简称 ADAM。

ADAM 是阿里巴巴基于内部多年的 Oracle 数据库和应用架构梳理、架构选型、系统改造经验，数字化帮助企业智慧决策。其目标是解决近年来传统 IT 系统基于 IO 架构创新、转型存在的难题。

一方面 IT 技术不断创新，国产 IT 产品不断丰富。如何有效的选择替代产品，尚无行之有效的方法论。

另一方面，IT 系统异构迁移存在的数据迁移风险、应用异构风险，目前没有完整的解决方案。在此基础之上，ADAM 产品应运而生。

最终 ADAM 给用户的方案，会满足用户 3 年到 5 年的规划，帮助企业用户实现转型。



我们通过经验产品化，通过构建标准化的服务流程，引入国内专业的第三方服务商，帮助我们一起提供给客户专业的异构迁移改造，以及迁移服务。阿里云同时也构建了原厂的培训认证体系、

全球技术支持分级服务、以及全场原厂的产品专家服务，来给我们的客户提供更完善的技术服务，从而帮助我们的客户标准化、规模化的完成传统数据库的异构上云。



# 下一代云数据库： MyBase 专属集群

作者 | 周正中 阿里云数据库高级产品专家

## 一、数据库发展历程

数据库的发展历史，可以往前追溯到 50 年代。整个数据库的发展历程从国防使用的大型机到商业银行使用的这种小型机，到企业应用 ERP 等等。从层次数据库、网状数据库到关系型数据库，再到互联网化开始的开源数据库，以及现在的云数据库引领新的潮流。



有一份 Gartner 报告数据显示出全球的云数据库发展趋势，预计到 2022 年 75% 的数据都在云上。未来云数据库大有可为。在魔力象限当中，云数据库厂商成长最快。



接下来简单讲一下阿里云数据库这一块目前到了一个什么样的状态。阿里云数据库是全托管的云数据库，2012年首次推出。在推出云数据库之前，阿里内部这个数据库已经应用了很长一段时间，经历了双11的考验，有非常厚的底蕴。

目前来讲，有数十万的用户，以及数百万的实例数。几乎市面上企业能用到的数据库，在阿里云上都有对应的云服务。对用户来讲，云数据库实际上是一个开箱即用的服务，省去了很多运维的成本。比如说，一分钟就可以创建一个实例，自动的参数调优配置，内置的监控告警，数据迁移也可以做到一键迁移。

另外，性能比开源的更快。内核层面的优化，参数层面的优化，快速的弹性的扩展，普遍来讲性能提升大概在30%左右。稳定性业界领先，可用性SLA：99.99%。另外在硬件这块的选型，还有压测这块，我们也做了非常全面的测试，使得我们在硬件底层更加的稳定。

在安全性这一块，我们推出了传输以及存储本身的加密，TDE跟SSL的支持，以及支持全球非常知名的一些安全认证。我们还有内置备份恢复，避免删库跑路，可以恢复到一周内任意时间点，以及回收站避免误操作。这就是我们阿里云数据库大概的一个现状。



在全托管数据库服务这块，一些关键能力简单介绍一下。

第一，高可用：

- 99.99% 可用性 SLA 保障
- 自动化高可用管理
- 秒级故障切换

第二，安全可靠：

- RPO=0 (100%数据可靠)
- 进不了：私有网络安全可靠
- 劫不走：数据链路加密保障传输安全
- 看不懂：透明数据加密确保数据无人能解
- 逃不掉：SQL 审计记录数据库所有行为。

第三，高性能：  
 • Ali 内核性能强劲  
 • 线程池  
 • 极速热点行更新（50 万 QPS）  
 • Query Cache  
 • CloudDBA：性能槽点无处可藏  
 • 性能洞察  
 • 自动优化：阿里云资深 DBA 就在身边。

第四，强扩展：  
 • 全球容灾  
 • 全球部署：异地多活，异地只读  
 • 敏捷弹性，分钟级扩展  
 • 只读实例，成倍提升性能  
 • 自动读写分离。

### 全托管数据库服务的关键能力



#### 高可用

- 99.99% 可用性SLA保障
- 自动化高可用管理
- 秒级故障切换



#### 安全可靠

- RPO=0 (100%数据可靠)
- 进不了：私有网络安全可靠
- 劫不走：数据链路加密保障传输安全
- 看不懂：透明数据加密确保数据无人能解
- 逃不掉：SQL审计记录数据库所有行为



#### 高性能

- Ali内核性能强劲
- 线程池
- 极速热点行更新（50万 QPS）
- Query Cache
- CloudDBA：性能槽点无处可藏
- 性能洞察
- 自动优化：阿里云资深DBA就在身边



#### 强扩展

- 全球容灾
- 全球部署：异地多活，异地只读
- 敏捷弹性，分钟级扩展
- 只读实例，成倍提升性能
- 自动读写分离

我们前面说了全托管的云数据库有这么多的好处，但是也有很多企业会觉得有一些困惑。比如说，用了云数据库之后，DBA/运维人员的日常工作就少了，是不是就要失业了？原来积累的一系列工具和自动化脚本用不了？云数据库出了问题，因为没有权限，没有办法及时登录操作系统去手工干预解决，干着急？运维的习惯全部被打破了，无法命令行操作，特别不习惯？业务复杂度很高，云数据库没有办法个性化，不如自建灵活方便？这也是全托管云数据库服务给 DBA 带来的日常的困惑。

### 全托管云数据库服务给DBA带来的日常的困惑



- 用了云数据库之后，DBA/运维人员的日常工作就少了，是不是就要失业了？
- 原来积累的一系列工具和自动化脚本用不了？
- 云数据库出了问题，因为没有权限，没有办法及时登录操作系统去手工干预解决，干着急？
- 运维的习惯全部被打破了，无法命令行操作，特别不习惯？
- 业务复杂度很高，云数据库没有办法个性化，不如自建灵活方便？

中大型的企业现在逐渐成为了上云的主流，它们对于数据库本身的一些要求跟中小型企业会不一样。中大型企业一般来讲都会有非常专业的 DB 团队，还有运维团队，他会考虑更多的东西。除了我们全托管已经解决的这些问题，比如说，高可用，高可靠，高性能以及强扩展。另外，他还会考虑，比如说，综合成本（购买成本、迁移成本、系统利旧，维护成本），稳定可控（稳定安全、权限开放、无缝上云、DBA 运维能力保留），解决规模瓶颈（大容量，高并发、高性能），帮助业务转型（完整服务、解决方案、代差能力）。

### 中大客户对数据库更高要求



全托管云数据库 服务解决了	还需要什么？
高可用	<b>综合成本</b> (购买成本、迁移成本、系统利旧，维护成本)
安全可靠	<b>稳定可控</b> (稳定安全、权限开放、无缝上云、DBA运维能力保留)
高性能	<b>解决规模瓶颈</b> (大容量，高并发、高性能)
强扩展	<b>帮助业务转型</b> (完整服务、解决方案、代差能力)

而今天全托管的云数据库服务不能完全解决这类问题。比如说，争抢：RDS 是大池子，一台主机可能分配多个实例，这些实例可能被不同的用户使用，DB 资源可能被其他公司的实例争抢？

习惯：原来积累的一系列工具和自动化脚本都用不了，无法复用原命令行管理系统等。

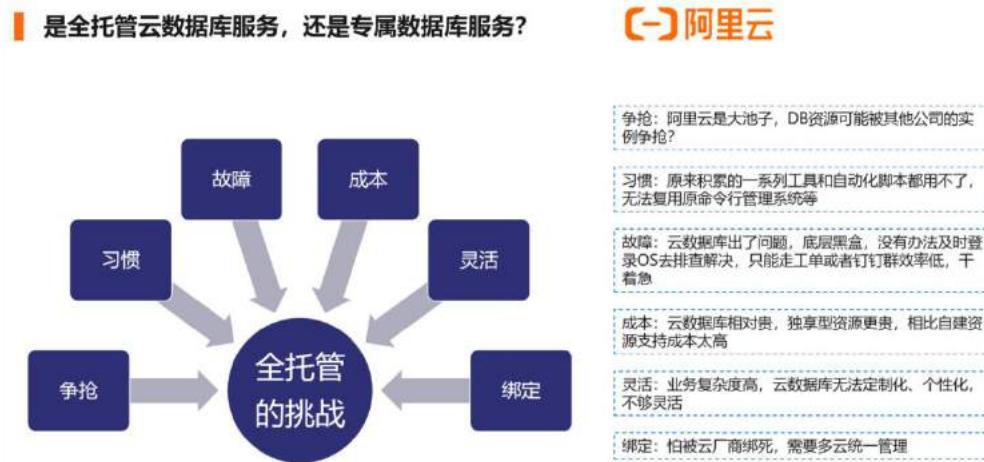
故障：云数据库出了问题，底层黑盒，没有办法及时登录 OS 去排查解决，只能走工单或者钉钉群效率低，干着急。

成本：云数据库相对贵，独享型资源更贵，相比自建资源支持成本太高。

灵活：业务复杂度高，云数据库无法定制化、个性化，不够灵活。

绑定：怕被云厂商绑死，需要多云统一管理。

这些都是全托管服务目前面临的挑战。



## 二、下一代云数据库形态： MyBase 云专属

下一代的云数据库形态， MyBase 云专属是专门针对中大型的企业上云所定制的一款产品。其实这也是顺应市场趋势的产品，从 2012 年推出 RDS 的全托管的云服务，到 2017 年推出了我们自研的云原生的 PolarDB 的这样一个产品。

在 2019 年，也就是今天的主角 MyBase 云专属这样一路走来，其实都是顺应了我们整个市场对云数据库的需求，从而产生这样的一些产品。 MyBase 云专属支持的引擎包括 MySQL/SQL Server/PG/Redis/MongoDB，还有 HBase 这样的一些非常通用的数据产品，后面也会陆续加入更多的引擎。

### 阿里云数据库发展历程

**阿里云**  
<https://www.aliyun.com/product/apsaradb/cddc>



一种新的面向中大客户的形态  
支持MySQL/SQL Server/PG/Redis/MongoDB/HBase

那么在我们今天再去介绍 MyBase 的核心能力之前，来看一下在云上原来没有 MyBase 的时候有哪几种选择。

第一种选择是云数据库服务。优势：

- Ali 内核，淘宝大促验证过的云数据库；
- 高可用，99.99% 可用性；
- 全托管，简单易用；
- 完善的备份、恢复、回档；
- DAS, DM S 工具支持；

劣势：

- 贵，成本是 ECS 自建的 1.4—2 倍；
- 灵活性差，不开放主机和数据库权限；
- 底层运维依赖于阿里支持体系，不能自主操作。

第二种是 ECS 自建。优势：

- 成本低；
- 自主运维，从 OS 到 DB，掌控力强；
- 从头到尾体现 DBA 价值；

劣势：

- 开源内核，不稳定，依赖社区做 bug 修复；
- 自主搭建高可用，存在数据一致性漏洞；
- 生态支持工具不完整，需要自主搭建；
- 运维管理投入成本高。

那么今天 MyBase 的推出，集两个形态的优势于一身，然后抛弃掉了所有的劣势，这就是云上我们为什么要推出 MyBase 专属集群，它兼具灵活、可控、技术兜底、成本低。

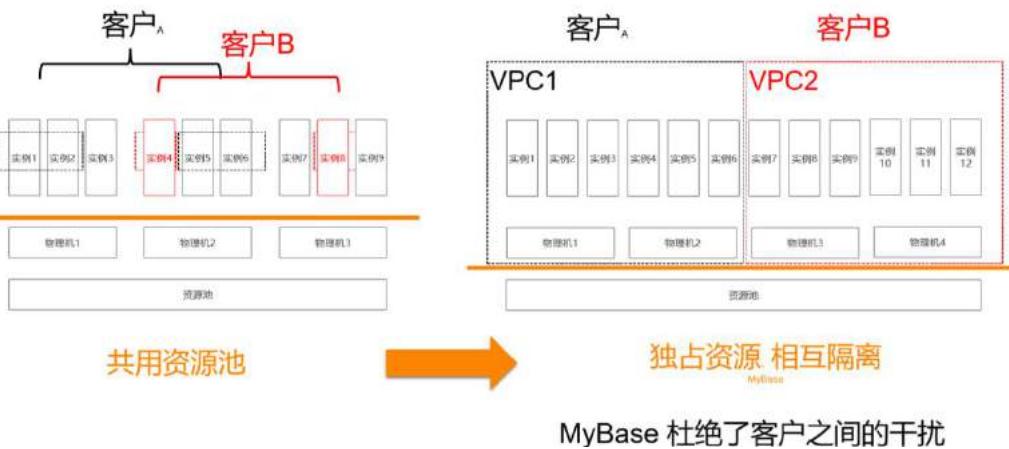
## 只能自建 or RDS? 还有一种选择： 专属集群(灵活、可控、技术兜底、成本低)



对于云数据库服务的形态，我们从共享资源池的形式变成了专属物理机的形式。如下图所示，左边是说在使用全托管的服务，用户创建一个实例，它到底创建在哪个物理机上，用户是完全不知道的。那么这个物理机上还有哪些客户跑了其它的实例，用户也不知道。如果有的话，那么相互之间可能出现一些干扰。那么我们专属集群把物理机这个层面透给用户。

也就是说，用户在创建 MyBase 集群的时候，它要勾选，比如说在集群里面创建几台物理机，那么这几台物理机的权限会完全的给到我们的用户，使得用户能够自己去掌控，这几台物理机不会有别的客户的实例。有了这样的权限的下沉之后，我们就可以做更多的东西，我会一一来介绍。

## 云数据库服务形态变化



MyBase 使用也是非常简单的。创建集群的时候，要指定你用了什么引擎，你的 CPU 的超卖比是多少。集群层面的一些参数配置好之后，就往这个集群里面添加主机的型号，选择本地盘的型号，也可以选择云盘的型号。集群创建好之后，主机也添加好了，接下来我们就可以去部署实例。跟我们创建 RDS 的逻辑是一样的，因为后面的实例的生命周期的管理这一块的代码跟 RDS 是完全一样的。

## 云数据库专属集群使用简便



The screenshot shows the process of creating and managing a dedicated cluster on the Aliyun Cloud Database Management Console. It includes three main sections:

- 1. 创建集群 (Create Cluster):** A configuration page for creating a dedicated cluster. It shows fields for engine type (MySQL, PostgreSQL, SQL Server, Redis), memory, storage, and network settings. A note indicates that the cluster is isolated from other users.
- 2. 添加主机 (Add Host):** A list of hosts currently assigned to the cluster, showing details like host ID, IP, port, and status.
- 3. 部署实例 (Deploy Instance):** A list of instances deployed within the cluster, showing details like instance ID, name, port, and status.

The URL <https://www.aliyun.com/product/apsaradb/cddc> is also visible at the bottom of the interface.

### 三、 MyBase 关键能力：降本增效

第三个部分简单介绍一下 MyBase 的关键的能力之一：降本增效。首先是资源超分配能力，适合游戏类客户，在同一个游戏的不同生命周期、不同热度的游戏库之间合理混布，提高资源利用率。因为整个集群的物理机透给你了，那么再去创建集群的时候，可以设置 CPU 超卖比。所谓的 CPU 超卖比，就是说一个企业里面可能会有一些业务，比如说，测试类的业务，或者是新上线的业务。这些业务对于资源的使用要求不是那么高，它可以分配出更多的实例出来。

比如说，8 core CPU，设置 200% 超卖比，相当于可以开出 16 core CPU 的实例，客户单 core 成本降低一半。除了 CPU，我们还支持磁盘超卖设置。

第二类就是说 Redis 跟 MySQL 混布的场景。因为这两个数据库的 workload 是完全相反的，组合起来去使用的话，我们的整机的 CPU 和内存的利用率可以提高，也就使得我们的整个成本下降。第

三种情况就是说针对不同的业务可以混布，合理利用超卖比，利用业务错峰实现更高资源利用率，从而降低整个的成本。

#### 资源超配能力

适合游戏类客户，在同一个游戏的不同生命周期、不同热度的游戏库之间合理混布，提高资源利用率

CPU 超卖比 200%



8 core CPU，设置 200% 超卖比，相当于可以开出 16 core CPU 的实例，客户单 core 成本降低一半  
除了 CPU，我们还支持内存、磁盘超卖设置

#### 阿里云

Redis 和 MySQL 混布

MySQL	Redis
CPU 12 core	CPU 4 core
Mem 12GB	Mem 128 GB

不同资源类型数据库混布，提高主机资源利用率

合理利用超卖比，利用业务错峰实现更高资源利用率



有了这个能力之后，我们就可以核算一下，以 MySQL 为例，降低 TCO 成本的逻辑，如下图所示：

- 专属集群：采购三台主机，月度费用 28305 元/月；按照 4U8G500G 实例来进行部署（36 个 HA 实例），每个 HA 的摊薄成本为 786 元/月。
- ECS 自建数据库：两台 ECS 自建数据库 HA，价格为 1480 元/月。专属集群的实例成本约为 ECS 自建数据库的 53%。基本上成本降了一半，这也是 MyBase 的一个能力所在，它可以帮助用户降低成本，这是第一个核心能力。

## 以MySQL为例，降低TCO成本



HA实例（月）	专属集群	ECS自建
折算成实例价格	786	1480
专属集群相比	53%	

逻辑：

- 专属集群：采购三台主机，月度费用 28305 元/月；按照 4U8G500G 实例来进行部署（36 个 HA 实例），每个 HA 的摊薄成本为 786 元/月。
- ECS 自建数据库：两台 ECS 自建数据库 HA，价格为 1480 元/月。专属集群的实例成本约为 ECS 自建数据库的 53%

注：1. 均为SSD磁盘。  
2. ECS为共享型 (ecs.s6-c1m2.xlarge, 最便宜)，SSD云盘

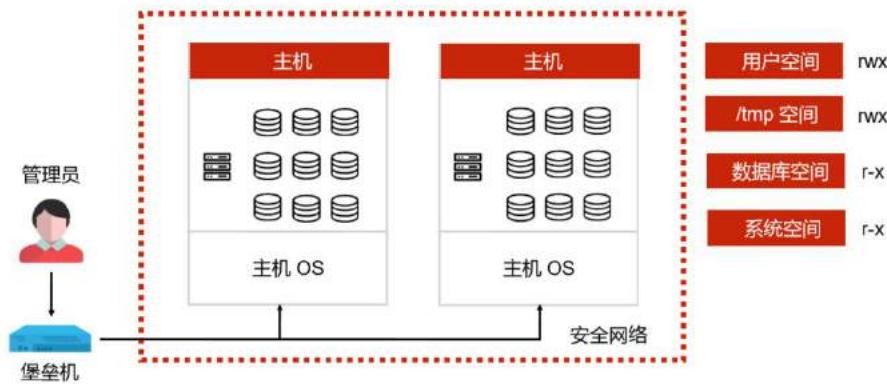
第二个核心能力是提高效率。相比于全托管的这种服务，我们 MyBase 是开放主机权限的。

管理员可以通过登录堡垒机连接到我们的主机，通过命令行连接到我们的主机操作系统。比如说，遇到一些紧急的情况，需要去排查性能诊断的时候，我们就可以登录到主机上面。主机上面也安装了一些常用的管理的包，比如说 DBA 常用的一些查看性能的或者查看网络的这样一些工具。

另外，通过堡垒机去登陆还有一个好处就是说，毕竟数据库对一个企业来讲是非常核心的资产，所有的操作都是需要经过堡垒机的审计的，所有的操作都会留档。

同时，堡垒机它自己也提供了一些安全的规则。一些比较危险的命令（例如造成删库跑路的 rm -rf 操作）可以通过配置规则去避免。

## 开放OS权限



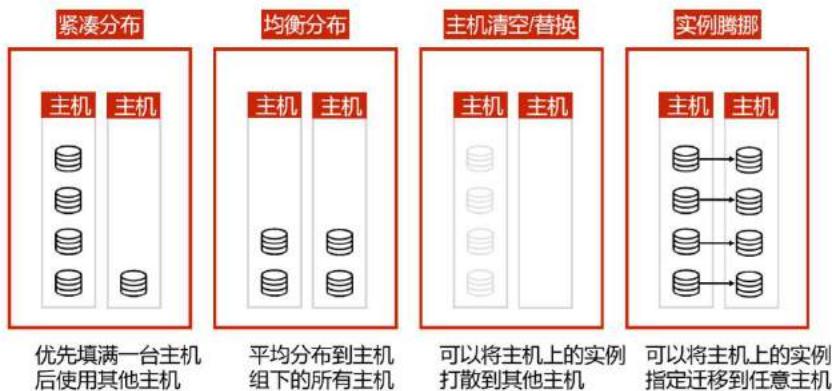
在 MyBase 这个产品里面，我们把后台的调度能力也透给用户了。比如说，在主机层面怎么去分配实例的调度能力。

第一，用户可以选择紧凑型的分布策略，就是说会优先把其中的某一些主机资源分配完之后，再去分配新的主机。它带来的好处就是说，始终有一些主机的资源还剩余比较多，那么就可以创建出更大的实例。

第二，第二种策略是均衡分布。在分配的时候，使得每个主机的剩余资源都差不多，它的好处就是说，我们的实例是可以设置弹性扩容的。

第三种是主机清空或者替换，可以将主机上的实例打散到其他主机。第四种是实例腾挪，可以将主机上的实例指定迁移到任意主机。

## 智能调度



## 四、MyBase 云专属之： PG 数据库引擎

MyBase 云专属的引擎，就是我们的 PG 数据库引擎。那么这款引擎到底有一些什么特点。在使用 MyBase 专属集群之前，用户如果自建的话，要用好开源数据库，需要三类角色。

一类角色叫老司机，一类角色叫汽车修理工，还有一类叫汽车设计师。

为什么会需要这三类角色呢，我们来看一下。当遇到架构、监控、优化等等问题的时候，对于一个企业来讲，它会怎么做。比如说遇到架构问题：

1. 极端情况：放弃治疗，换数据库，堆机器？...浪费时间、人力、资源，还不一定能解决问题。
2. 给足时间让员工自学、培养周期长，还有人才成长流失问题。
3. 高端人才可遇不可求，一对一 backup，长期闲置，人力成本高。
4. 求助搜索引擎、社区咨询，响应时间无法保障，特别是在深夜。

第二类，遇到 bug 怎么办。遇到 bug 的话，一般企业是不可能去长期雇用专职的内核的研发人员的。这种情况是非常常见的，所以遇到 bug 就必须要等社区修复，而这个周期真的不好说。

第三类，无法满足业务需求怎么办：

1. DBA、SA 有权查看、篡改、删除敏感数据。
2. 普通用户可以通过 security invoker 属性来设置陷阱函数。
3. 敏感信息(例如用户密码)可能明文出现在 SQL 审计 日志、历史 SQL 命令、共享内存(例如会话状态、pg\_stat\_statements 跟踪会话)、dblink 视图、外部表 user mapping 的定义中。

 阿里云  
用好开源，您需要：“老司机、汽车修理工、汽车设计师”  
You need full-stack for open-source and a lot of money.



遇到架构、监控、优化等问题怎么办？

社区BUG ~10Day+ (参考PG bug mail list):  
 1. 模糊情况: 放弃治疗, 换数据库, 堆机器?...浪费时间、人力、资源, 还不一定能解决到底。  
 2. 自学、学习、培养周期长, 人才成长流失问题。  
 3. 高端人才可遇不可求, 一对一 backup, 长期闲置, 成本高。  
 4. 求助搜索引擎、社区咨询, 响应时间无法保障, 特别是在深夜。



遇到BUG怎么办？

插件BUG ~xxDay+:  
 1. 报告插件作者 1Day  
 2. 等待作者修复, 5 ~ ? Day(漫长)  
 3. 下载patch, 验证 5 Day  
 4. 升级插件 1 Day



无法满足业务需求怎么办？

1. DBA、SA有权查看、篡改、删除敏感数据。  
 2. 普通用户可以通过security invoker属性来设置陷阱函数。  
 3. 敏感信息(例如用户名)可能明文出现在SQL审计日志、历史SQL命令、共享内存(例如会话状态、pg\_stat\_statements跟踪会话)、dblink视图、外部表user mapping的定义中。

如果使用阿里云 PG，我们不需要上述三类角色就能克服上述三类问题。

### 第一类，遇到架构、监控、优化等问题：

1. 资深 PG 专家资源池, 7\*24 技术咨询.
2. DBA 可以将更多时间投入到业务优化、架构设计等关键事务，消除能效比较低的重复劳动。

### 第二类，遇到 BUG 怎么办：

1. AliPG 兼容开源 PG，阿里云专业内核团队为 AliPG 代码兜底.
2. AliPG 与公共云一套源码，大量商业用户在使用，有 BUG 已早修复.
3. 升级简单，用户点升级版本按钮，或配置自动升级任务。

### 第三类，当数据库内置功能无法满足业务需求时怎么办？:

1. AliPG 拥有大量商业用户，覆盖众多行业，AliPG 围绕行业痛点，不断扩展实用功能：  
自主研发插件 18 款(详见产品手册)，覆盖 GIS, 冷热分离, 搜索, 图像识别, 安全加固等场景。

阿里云

## 使用阿里云PG,还要什么“老司机”

If you use Aliyun PG, you can focus on your business

遇到架构、监控、优化等  
问题怎么办?

遇到BUG怎么办?

无法满足业务需求怎么办?

1. 资深PG专家资源池, 7\*24 技术咨询。  
2. DBA可以将更多时间投入至业务优化、架构设计等关键事务, 消除效能比较低的重复劳动。

1. AliPG兼容开源PG, 阿里云专业内核团队为AliPG代码完稿。  
2. AliPG与公共云一套源码, 大量商业用户应用的选择, 有BUG已早修复。  
3. 云端, 用户点升级版本按钮, 或配置自动升级任务。

1. AliPG拥有大量商业用户, 覆盖众多行业。AliPG围绕行业痛点, 不断扩展实用功能:  
• 自研插件18款(详见产品手册), 覆盖GIS, 冷热分离, 授权, 图像识别, 安全加固等场景。

阿里云提前洞察行业需求, 用技术帮助客户突破商业边界, 是企业纷纷选择阿里云 PG 的重要原因。

总结来讲, 选择用阿里云 PG 的这些企业可以得到更多的一些能力, 比如说:

第一, 提升业务体验价值, 更快。行业场景: • 图像识别、向量相似搜索. • 实时营销、用户画像. • 短视频、实时推荐. • GISMOD 移动对象处理。

第二, 核心业务必要能力, 更稳、更安全。PaaS 多租户场景: • 大并发大量元数据、大量连接优化. • 资源隔离优化。安全加固场景: • 敏感信息加密, 函数调用陷阱规避, • SGX 全加密, • 最大保护、最高可靠、最高可用模式, • slot failover。

第三, 成本、部署形态必要能力, 更灵活、更可控。专属集群 PG: • 开放 OS 权限, 自主可控 • 自定义超卖, 降低成本. • 开放调度功能, 自定义调度策略。

阿里云

## 为什么企业纷纷选择阿里云PG

Why do companies choose Aliyun PG

提前洞察行业需求, 用技术帮助客户突破商业边界.  
Insight into industry needs ahead of time helps customers push the boundaries.

提升业务体验价值  
更快

行业场景:  
• 图像识别、向量相似搜索。  
• 实时营销、用户画像。  
• 短视频、实时推荐。  
• GIS MOD移动对象处理。

核心业务必要能力  
更稳、更安全

PaaS多租户场景:  
• 大并发大量元数据、大量连接优化。  
• 资源隔离优化。  
安全加固场景:  
• 敏感信息加密, 函数调用陷阱规避。  
• SGX全加密。  
• 最大保护、最高可靠、最高可用模式。  
• slot failover

成本、部署形态必要能力  
更灵活、更可控

专属集群PG:  
• 开放OS权限, 自主可控。  
• 自定义超卖, 降低成本。  
• 开放调度功能, 自定义调度策略。

# 乘云而上，ADB 云原生数仓发展之路

作者 | 韩锋 阿里云数据库高级产品专家

## 一、数据仓库发展趋势

首先回顾一下整个数据仓库的发展趋势，我将从下面这四个维度，海量、实时、智能和云化，来分析一下当前数仓发展的变化趋势。我们大致可以把它归结为以下 4 个方面。

第一个就是数据的体量问题，2020 年全球的数据规模大概在 40ZB，这是一个非常庞大的数字。目前的现状是数据非常庞大，并且增长的速度也非常之快。预测在 5 年之后，体量可能会变到现在的 4 倍以上。面对如此海量的数据以及爆炸性的增长，作为后端数据的存储和计算的能力怎么样去适配这种变化趋势。数据仓库作为一个承载数据的载体，不得不面临这样的一些问题。

第二个是关于实时处理方面。这些年来，数据实时处理的变化趋势是非常明显的。特别是一些新业务有更高的实时化的需要。整个实时数据的占比在不断的提高，为我们的数据计算层的能力带来了挑战。

第三部分是数据处理的智能化。随着我们对数据使用的不断深入，有越来越多的半结构化、非结构化的数据被提取出来。如何挖掘出这部分数据的价值，是我们在数据处理上面非常重要的一个点。

第四个部分就是加速云化。越来越多的企业把它的基础设施构建在云端，作为基础设施之一，数据库是一个非常典型的代表。在近期数据的整体使用趋势上，呈现出了海量的、实时的、智能的、云化的这样一些特点。



**数据使用整体趋势**

## 数据的计算存储正在发生变化

海量、实时、智能、云化

<b>01 规模爆炸性增长</b> <b>40ZB</b> 2020 年全球数据规模 <b>430%</b> 2025 年 vs 2020 全球数据规模	<b>02 生产/处理实时化</b> <b>30%</b> 2025 年 实时数据占比 <b>50%</b> 2022 年 新业务将采用实时分析比例	<b>03 生产/处理智能化</b> <b>80%</b> 非结构化数据占比 <b>55%</b> 非结构化数据年增速	<b>04 数据加速上云</b> <b>49%</b> 2025 年 数据存储云上规模 <b>75%</b> 2023 年 数据用云上规模
--	--	---	---

CHINA POSTGRESQL ASSOCIATION

我们看看之前的数据仓库的整个领域是什么样的特点。如下图所示，左侧是一些比较常见的传统数仓的玩家。也就是说，可能在 10 年以前，从这样的一些数仓产品里去选择一个就可以了。但是近年来，我们发现数据仓库领域的玩家正在发生变化。最右边罗列了 4 个比较典型的厂商。从 19 年到 20 年，这种新型的数据仓库产品的受众已经等量，甚至超过了传统数仓。大家越来越关注这种新型的数仓。

PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20 阿里云 PostgresConf.CN & PGConf.Asia 2020

# 数仓玩家变化

公司	产品	技术特点	产品来源
SAP	Sybase IQ	列存 + 行存混用	商用 (2010)
HANA	行存混合 + 内存计算	商用 or 原生一体机	商用 (2009)
HP	Vertica	列存 + MPP型	商用 (2011)
EMC	Greenplum	行存 + 列存 + MPP型混用	商用 (2010)
IBM	Netezza	定制 FPGA 架构通过连接片实现	专研一体机 (2011)
Teradata	Teradata	行存混合存储, RealSQL 引擎	2007 T2000 商用 (2012)
Microsoft	HBase	列存 + Key Value	开源社区 (2006年)
ORACLE	Exadata	列存 + 分布	原生一体机 (2010)
GBASE	Gbase 8a	列存 + MPP型	国内 (2008年)

DB-Engines Ranking

Rank (Approximate Popularity)

2012 2013 2014 2015 2016 2017 2018 2019 2020 2021

0.0m 2.0m 4.0m 6.0m 8.0m 10.0m 12.0m 14.0m 16.0m 18.0m 20.0m 22.0m 24.0m 26.0m 28.0m 30.0m 32.0m 34.0m 36.0m 38.0m 40.0m 42.0m 44.0m 46.0m 48.0m 50.0m 52.0m 54.0m 56.0m 58.0m 60.0m 62.0m 64.0m 66.0m 68.0m 70.0m 72.0m 74.0m 76.0m 78.0m 80.0m 82.0m 84.0m 86.0m 88.0m 90.0m 92.0m 94.0m 96.0m 98.0m 100.0m

Oracle Database, MySQL, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Google BigQuery, Snowflake, Microsoft Azure

我们来探究一下数据需求的阶段性的演进。最初级的阶段，我们对数据的要求是比较简单的，它更多的去做一些批量的处理和预定义的查询。我这个系统或者我的业务过去一段时间发生过什么，重点是一些数据库的基本能力和承载力，数据规模的问题，只要能解决这个问题就可以了。

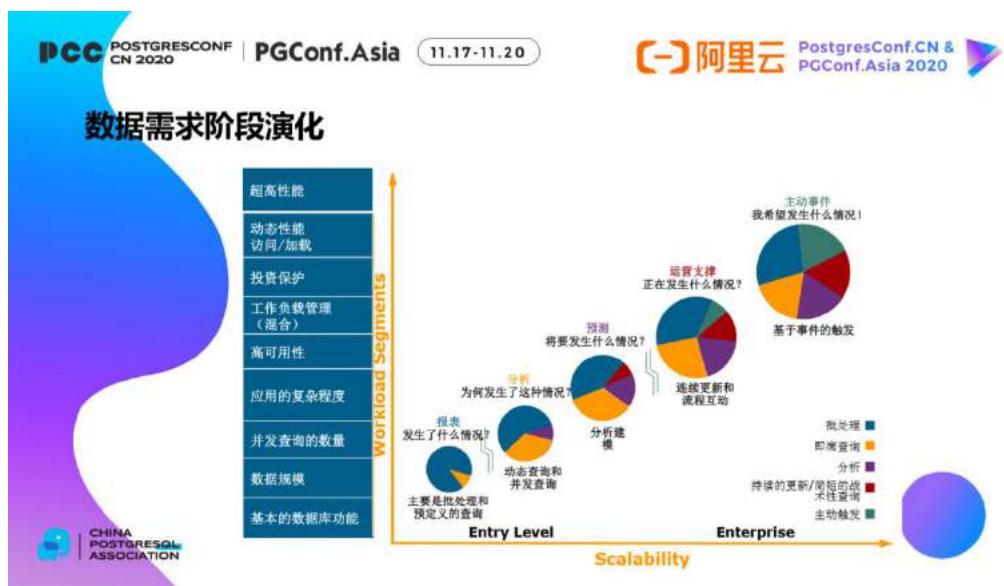
这是一个比较低级的阶段。

第二个是分析阶段，我们会有一些更加动态的查询和并发查询。动态的查询实际上来自于我们对一些实时的分析类的需求。并且会有更高的一些并发性，也就是有更多的人并发的执行这种非预定义的查询，显然这块对数据库的计算能力提出了更高的要求。同时，它的应用的复杂度也提高了很多。

第三个是预测阶段，我们有一些预测类的需求。基于我们自己做好的数据模型，去做一些预测类的分析和判断，未来会发生什么。对我们的建模能力，以及基于建模构建的数据分折的能力带来更高的要求。

第四个阶段叫做运营支撑阶段。我们这个系统正在发生什么，有大量的比较频繁的互动类的需求，连续的更新和流程互动。所以这里面会有一些对数据处理的实时性的要求。

第五个阶段会有另外一个特点，就是主动响应。主动的是事件的响应，我希望发生什么情况。随着数据需求的演进，应用特点的一些变化，对后端的数据存储计算平台提出了非常高的要求。



针对数据使用的一些变化，我们提出了一个观点，就是数据和数仓的演进趋势，正在从所谓的 Big Data 到 Fast Data。在新的环境下，大数据平台暴露出一些问题，这就是为什么我们推出了所谓的 Fast Data 这样一个理念。Fast Data 强调更快捷、实时的去做数

据的处理。这里有两个关键点，一个是 online，一个是 fast。我们更加强调对全量数据做实时的数据处理和实时的数据计算，同时还可以根据客户的需要去做弹性的扩展。

Fast Data 非常符合云上的理念，更多的强调的是按需做弹性处理的能力，包括我们的计算，包括我们的存储，这是我们看到的数仓变化的一个趋势。



从市场角度看，数仓领域也面临着非常多的变化。

第一，平滑迁移。保护企业积累多年的资产，而不是推翻重构。

第二，解决厂商的“卡脖子”问题，降低企业综合成本。

第三，解决弹性问题，满足企业对海量、多模数据的分析需求。

第四，云化需求。解决企业使用传统数仓繁重架构，满足对轻量级、多租户的要求。

第五，架构升级需求。满足企业从传统数仓->数据平台->数据中台的升级要求。

**数据市场变化**

- 平滑迁移，保护企业积累多年的资产，而不是推翻重构。
- 解决厂商的“卡脖子”问题，降低企业综合成本。
- 解决弹性问题，满足企业对海量、多模数据的分析需求。
- 云化需求，解决企业使用传统数仓繁重架构，满足对轻量级、多租户的要求。
- 架构升级需求，满足企业从传统数仓->数据平台->数据中台的升级要求。

CHINA POSTGRESQL ASSOCIATION

## 二、云原生数据仓库能力

前面谈到了数仓市场的一些特点，下面我们谈谈针对这些特点，云原生数仓应该具备什么样的能力。

第一个是数据库与大数据的一体化。过去企业构建数据基础平台的时候，往往通过一种混合的模式，有大数据平台，也有数据库平台，甚至还有其他类的平台，混合在一起为我们提供一个数据服务。随着技术的演进，我们可以把数据库与大数据一体化处理。对客户来说，可以用单一平台提供与原来相同的服务。

这里有几个关键的技术点。

第一，行列的混存。行存适用于某些业务的访问特点，列存符合另外一些特点。我们尽量通过一种层次结构满足不同的需求，所以我们有行列的混存。

第二，混合负载管理。用户使用数据的特征是不一样的，怎么样去做好这种资源的隔离，怎么样去满足不同的需求。

第三，融合计算引擎。如何满足客户对于计算能力，特别是实时性这块更高的一个要求。

第四，智能索引。如何快速的去满足客户的这种优化类的一些方案。我们具备的能力包括，多维分析，实时数据的写入和查询，明细的一些查询的诉求。这些能力的集成代表我们整个的数据库平台已经具备了数据库与大数据一体化的能力。

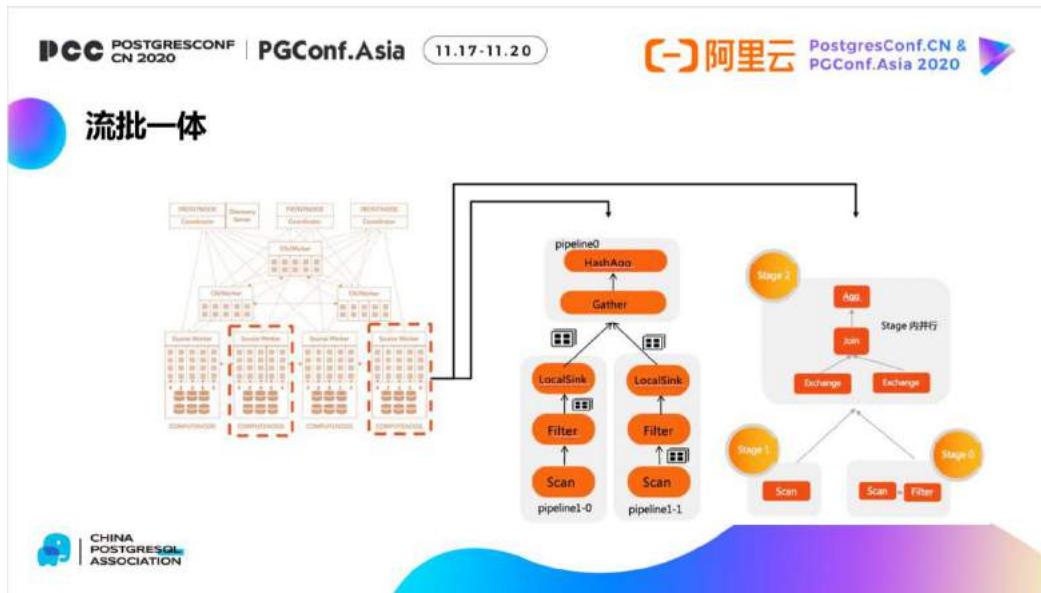


第二个能力是资源弹性。有几种不同的弹性，我们可以在计算层面去做一些弹性的处理，在存储层面同样可以做一些弹性的处理。在不同的层去做弹性的价值是不一样的。从底层来说，如果我们的存储数据的介质部分支持这种弹性能力，同时还支持冷热数据的分层，会大幅度的降低客户的数据的使用成本。在上面的计算层，可以通过我们的资源组的隔离，保证核心计算任务稳定运行。同时，可以按照客户的需求做更多的资源方面的一些弹性，技术方面的一些弹性。可以按照时间段，或者按照客户的需求去扩展他的计算能力。达到这样一个状态之后，对客户来讲有非常大的价值。



第三个能力是流批一体。流处理和批处理是过去处理数据的两种不同的方式。到现在，这两种处理方式有合二为一的趋势。我们将流处理和批处理合为一体，通过一个统一的方式去实现。

用户不用再去纠结到底是需要用流处理，还是需要用批处理。客户只要看他需要什么，我们给他反馈什么就可以了。



从过去的基于这种传统的分析系统，到现在的云原生架构，有一些直观感觉的变化。首先从扩展性来说，传统的数据仓库面临着很大的一个问题。客户很难在初期构建一个全量的数据仓库，在不断的发展积累中去做扩展。那么这个时候，如果你的数据仓库提供了一个非常好的弹性能力，是可以满足客户这方面的需求。

第二个变化是关于成本。数仓相对来说是一个投入比较大的一个平台。云数仓可以按需付费。这边的需求可能是对于计算的需求，也可能是对存储的需求，都是可以去按照客户的需要进行支付的。

第三个变化是关于架构。过去我们的架构是通过多种不同的技术栈组合在一起的一个混合架构。我们现在可以做到单一架构去解决所有的问题。

第四个变化是性能。过去只能做离线的处理，而现在能做非常好的实时性处理。



### 三、阿里云云原生数仓实践

阿里在云原生这方面做过非常多的实践。首先来看看阿里云关于数据仓库的一个大图。图的中间部分罗列了目前阿里云在数仓上面的一些产品的布局，这里边包括了几类的产品。第一个就是云原生数仓，AnalyticDB，简称 ADB 这样的一个产品。在它的下面我们也有一些基于云原生的数据湖分析这样的一个产品，简称 DLA。

同时，在某些特定的场景下，我们也可以去选择一款托管开源的产品，叫 Clickhouse。这就是我们阿里在整个数仓当中一个产品的布局。通过这样的一个数据的存储和处理平台，下端对接了很多的数据源。包括我们传统的数据库也好，一些大数据平台也好，包括我们的对象的存储，我们的这种日志服务，都可以作为我们的数据源去对接上来。横向部分，阿里云自有生态的一些产品和第三方的产品，也都是可以去支持。

最上面关于数据分析，数据可视化这块，我们也支持了很多自己的，或者第三方的一些产品。同时，也有非常好的一些接口，支持用户做一些自定义应用的开发。这就是我们看到的阿里云整体的数仓大图。



下面重点谈一谈云原生数仓 AnalyticDB 在 PostgreSQL 这个版本的一个基本的情况。这是一款高性能、海量扩展、兼容部分 Oracle 语法生态的金融级的数据仓库。这里罗列出它的几个特点。

第一个是关于体量的问题，这是一个非常典型的 MPP 的扩展架构，它可以支持 PB 级的数据规模。这样的一款产品，我们还在上面做了非常有代表性的一些能力，包括这种向量化计算的能力，执行引擎的能力，包括新一代的 SQL 的优化器。通过这样的方式去满足我们对于海量数据的一个处理能力。

第二个是关于稳定可靠。这样的一个产品是基于阿里飞天平台构建了一个非常智能的运维的一个能力。这方面阿里有非常多的一些实践，像这种智能诊断，智能的硬件管理，包括整个集群的这种高可靠，自运维的能力。

第三个是 HTAP。支持分布式事务，支持标准数据库隔离级别。支持高吞吐，高并发事务，TPC-C 交易负载。

第四个是关于 SQL 的完备度，体现了我们这个产品对外友好的接口，它支持一个比较完备的 SQL 2003 的标准，部分兼容 Oracle 的语法。同时也构建了一些工具平台，方便客户从传统平台迁移到我们这款平台。同时对一些非常典型的企业级的工作要求，像这种存储过程，窗口函数，触发器等等，已经实现了一些支持。

第五个是关于多模的一些能力，包括对我们的空间数据实时去响应。其实在阿里内部有很多的这种基于空间的应用，都是构建在这样的一个产品上的。

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**阿里云 PostgresConf.CN & PGConf.Asia 2020**

## AnalyticDB PostgreSQL版简介

高性能、海量扩展、兼容部分Oracle语法生态，快速构筑金融级数据仓库

- PB 级数据** MPP水平扩展架构，PB级数据查询秒级响应；向量化计算，及列存储智能索引，领先传统数据库引擎性能 3x；新一代SQL优化器，实现复杂分析语句免调优
- 稳定可靠** 飞天平台基于阿里多年大规模集群系统构筑经验打造，智能硬件管理，故障监控诊断自恢复，支持MPP数据库实现复杂集群系统高可靠，自运维
- HTAP** 支持分布式事务，支持标准数据库隔离级别。支持高吞吐，高并发事务，TPC-C 交易负载百万 tpmC。
- SQL完备** 支持SQL 2003，部分兼容Oracle语法，支持PL/SQL存储过程，OLAP窗口函数，触发器，视图等，完备功能和生态，实现应用快速适配、或迁移。
- 多模分析** 通过PostGIS 插件支持地理信息数据分析；高性能向量检索算法，支持视频/图像检索以图搜图

CHINA POSTGRESQL ASSOCIATION

下面我会就这个产品的几个功能亮点，单独再去展开一下。

第一个功能亮点是多 Master。Multi-master 是云原生数据仓库 AnalyticDB for PostgreSQL 6.0 版的一个大的增强特性，支持 通过水平扩展协调节点（Master Node）来突破原架构单 Master 的限制。在计算节点（Compute Node）不存在瓶颈的情况下，系统连接数及读写能力可以随着 Master 节点数线性扩展，从而进一步提升系统整体能力，更好的满足实时数仓及 HTAP 等业务场景的需求。

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**阿里云 PostgresConf.CN & PGConf.Asia 2020**

## 技术创新：多Master

APP 提交请求 → ALB → Master Node → Segment

Diagram illustrating the Multi-master architecture:

- The APP sends a request to the ALB (Application Load Balancer).
- The ALB routes the request to one of the Master nodes.
- The Master node handles the request and performs DDL/DML operations.
- The Master node also manages PITR (Point-in-Time Recovery) and GFS (Global File System) processes.
- The Master node is connected to multiple Segments.
- The Segments are represented as blue rectangles at the bottom, showing data flow between them.

Multi-master是云原生数据仓库AnalyticDB for PostgreSQL 6.0版的一个大的增强特性，支持通过水平扩展协调节点（Master Node）来突破原架构单Master的限制。在计算节点（Compute Node）不存在瓶颈的情况下，系统连接数及读写能力可以随着Master节点数线性扩展，从而进一步提升系统整体能力，更好的满足实时数仓及HTAP等业务场景的需求。

CHINA POSTGRESQL ASSOCIATION

第二部分是关于执行引擎这方面。我们纯自研了一个新一代的执行引擎，叫 Laser。通过一些极致的优化，提升了整体的一个性能。新一代计算引擎 laser，消除火山模型碎片化内存分配，采用 LLVM 进行动态代码生成（CodeGen），提升表达式计算性能，利用 CPU 的 SIMD 技术，指令级并行，进一步提升性能。对于客户的价值就在于，客户所写的语句不用有什么大的变化，请求执行层面来讲可能就会有一倍以上的性能提升。



第三块是联邦分析，更加强调我们整个产品的一个开放程度。

对于客户来讲，他不仅有传统的数据仓库，可能还有自己的大数据的平台，如何把这个客户在他内部构建的这么多的数据统一的去管理使用起来，这个时候我们就引入了联邦分析。它的基本原理就是把很多的数据源，包括大数据平台，关系型数据库，或者是对象存储这样的一些平台，作为一个外置的数据源，通过一个外表的形式加载进来。

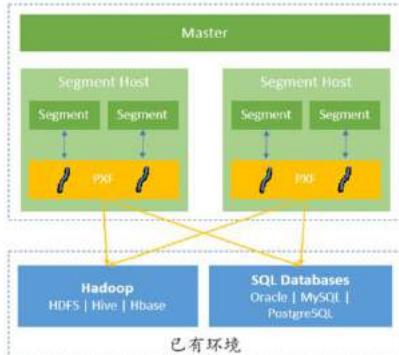
对于客户来说，只要去创建一个外表就相当于打通了 App 产品和外部数据源的一个通路，然后可以在 APP 里边统一的去做数据访问。同时我们还提供了一些性能的加速，把很多的计算能力下沉到下层的其他引擎去处理，这样能更进一步的提升整个的访问效率。

对于客户来说，相当于我们有一个虚拟的数据库访问层，可以访问不同的数据源。

对客户来说，他的体验会非常的好，这就是我们谈到的联邦分析的一个能力。



## 技术创新：联邦分析



通过部署PXF服务进行外表访问：

- Hadoop HDFS(Text | Avro | JSON | ORC | Parquet)
- SQL Databases(Oracle | MySQL | PostgreSQL)

性能加速：

- 列过滤
- 谓词下推

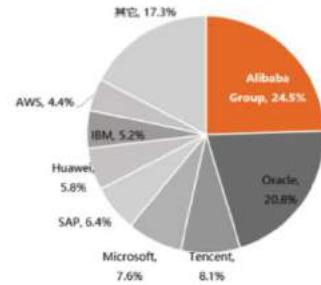
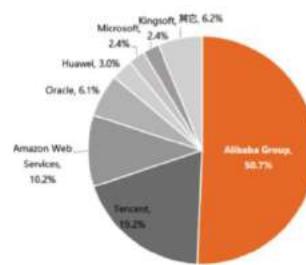
```
postgres# CREATE EXTERNAL TABLE read_pxf_parquet(location text, month text, number_of_
LOCATION ('pxf://data/pxf_examples/pxf_parquet?PROFILE=hdfs:parquet')
FORMAT 'CUSTOM' (FORMATTER='pxfrwriterable_import');
```

前面我们谈到三个不同的技术点，之后我们来看看整个云原生的数据仓库的发展，以及目前阿里云在国内的整体的形势。云原生的数据仓库弥补了传统数仓的一些不足之处，很多的用户也跟我们谈到使用云原生数仓给他们带来了一些非常大的利好。

下图来自第三方机构，可以看到，在左侧整个公有云这块，在国内阿里巴巴目前是领军的一个企业，市场占比将近一半。在下图的右侧，从国内的数据库市场占比来看，阿里巴巴在去年已经超过了传统的数据库的一个巨头，这是一个非常具有代表意义的标志事件。这个事件说明阿里云经过这么多年的发展，在基础设施，特别是云端的基础设施的构建上面，在数据库这个领域已经处于国内的领先地位，并且越来越多的参与到全球的竞争之中。



## 乘云而上，云原生数据仓库AnalyticDB



# 云原生数据仓库 TPC-H 第一背后的 Laser 引擎大揭秘

作者 | 魏闯先 阿里云数据库资深技术专家

## 一、ADB PG 和 Laser 计算引擎的介绍

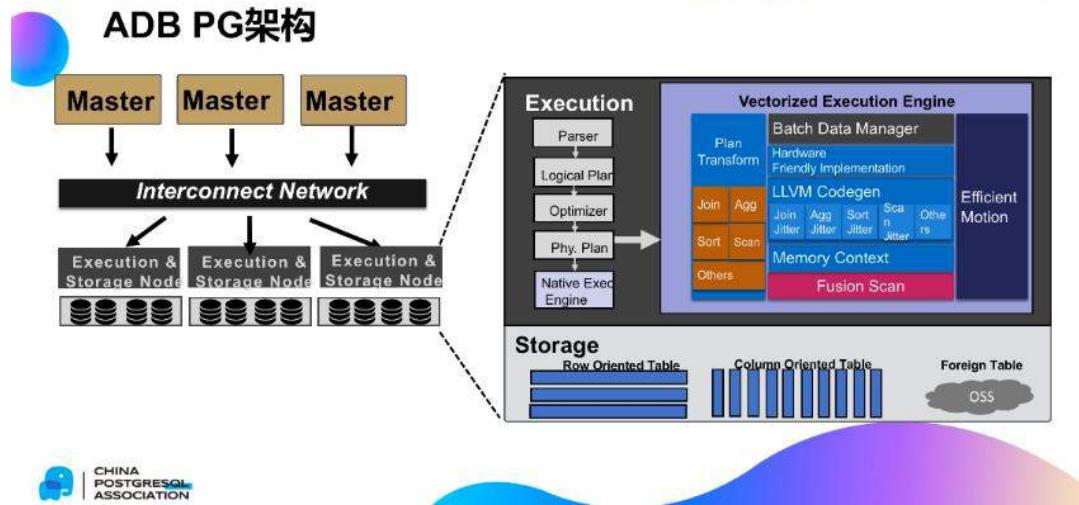
### (一) ADB PG 架构

ADB PG 是一款云原生数据仓库，在保证事务 ACID 能力的前提下，主要解决云海量数据的实时分析问题。它的架构与传统的 MPP 数据仓库非常类似，主要分成两层，第一层是 master 节点；第二层是 work 节点。

master 节点主要承担实时写入和事务的处理，执行计划的生成。与其他的传统的 MPP 数据仓库不同的是 ADB PG 的 master 节点支持线性扩展，可以通过多个 master 提升整体的事务能力、实时写入吞吐能力。

work 节点主要承担两个功能，第一个功能是执行，第二个功能是存储。执行引擎采用的是向量化执行引擎，通过向量列式 Batch 处理，codegen 技术，以及 Fusion Scan 加速列式计算效率，在一些分析场景性能相对于普通的火山模型有了 3~5 倍的提升。

同时它的存储节点不仅支持传统的行表和列表，也可以通过外表方式支持一些开源的表结构，例如 Parquet、ORC 等开源数据结构。ADB PG 可以直接分析保存在类似于像 OSS/HDFS 等分布式存储上的数据，减少数据的导入，可以大幅的降低用户的使用门槛和存储成本。



## (二) 为什么做 Laser 计算引擎

为什么不采用 PG 原生的计算引擎?

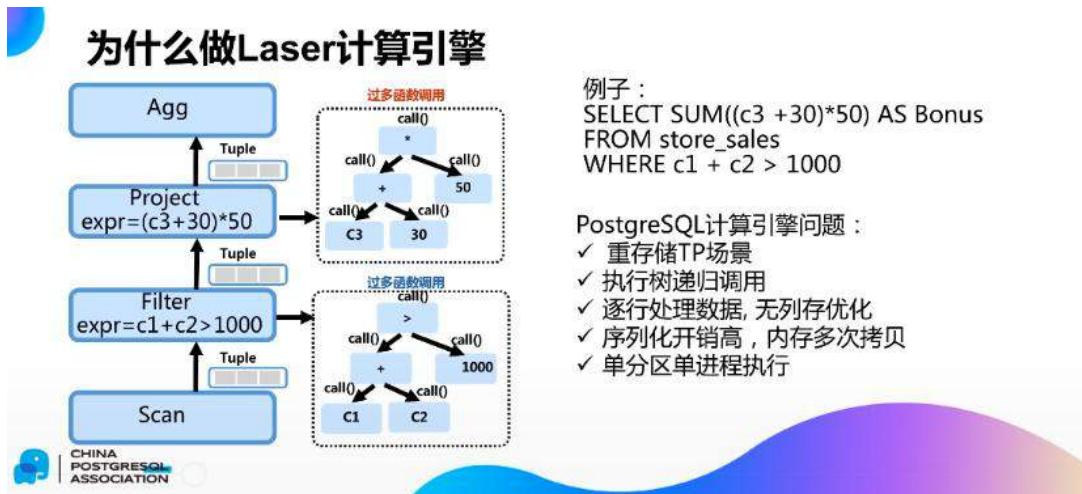
第一，PG 是一个传统的事务型数据库，它主要的优化大多来自于 TP 的事务优化，并没有针对海量数据的分析计算做定制化的优化。

第二，PG 计算引擎采用解释执行的逻辑，复杂表达式采用函数执行树的递归的调用解释执行。如图中的例子所示，每个表达式都被翻译成一个函数并组成一个执行树。每一条的数据都通过以上的函数递归调用去完成最终的计算。它带来的问题就是在海量千万以上的数据计算场景，虚函数调用大幅影响计算的性能。

第三， PG 默认只支持行存存储引擎，并不支持列存，所以它的整个计算执行过程是逐行处理数据，并没有对列存做定制化的优化。

第四， PG 代码具有非常好的扩展性、兼容性，但是在性能上考虑不多。例如说在计算过程中会涉及到内存的多次拷贝，序列化的开销比较高。

最后，PG 采用的是单分区单进程执行模式，无法充分发挥现代多核 CPU 的优势。



### (三) 主流 OLAP 业界方案

在开始 ADB PG 计算引擎的选型过程，我们重点调研了当前主流的 OLAP 业界主流方案。

主要分成两大类，第一类：向量化执行；第二类：编译执行。

- 第一：向量化执行的基本做法，非常类似于火山模型，主要差别来自于它采用的是批量计算，每一批里优先列式计算。

这种模式优势在于可以充分发挥 CPU 的流水线执行和内存的顺序访问，减少 cache 的 miss。缺点是第一实现逻辑比较复杂。第二，它需要批量的数据，并不适合每次计算数据量较小的 OLTP 场景。第三，它需要缓存批量数据，缓存本身带来一定的内存开销。

- 第二：编译执行基本做法是使用 Codegen 技术，将所有的算子编译成函数，通过 PUSH 的模型自下而上通过数据上推完成计算。

优点一：由于每次计算的都是一行数据，执行过程可以将这一行数据保存在寄存器里面，寄存器计算代替内存计算。优点二，它通过 Codegen 技术把复杂的算子编译成一个函数，所以它非常适合计算复杂性非常高的计算加速。

缺点在于，PUSH 模型控制逻辑比较复杂，同时由于采用单条计算，无法做到内存的顺序访问，所以它整体的 Cache miss 率比较高。典型的代表产品有 Spark 和 Peloton。

ADB PG 综合考虑这两类方案的优缺点，最终使用了向量化的方案，主要考虑到 ADB PG 原生 PG 火山模型与向量化模型架构上较为接近，改造成向量计算引擎的逻辑顺畅，改造成本较编译执行小。



#### (四) 业界主流计算引擎对比

我们对业界主流开源计算引擎做了对比，包括 ClickHouse、PG11、Spark、ADB PG。在执行模型上，ClickHouse 和 ADB PG、PG11 都采用的向量执行，Spark 采用的是编译执行。

在内存模型上，ClickHouse 采用的是列存、PG11 采用行存、Spark 采用行存、ADB PG 采用行列混存。行列混存主要应用在类似于 join、AGG 的场景，我们可以将多列 group by、hashtable 数据拼成一列数据，可以充分发挥顺序访问的效率。

在即时编译上，ClickHouse 采用表达式级 LLVM、PG11 采用表达式级 LLVM，Spark 采用 Stage Java 技术，ADB PG 采用算子级 LLVM 技术。算子级 LLVM 技术可以提升算子的计算性能。

在硬件加速上，ClickHouse 和 PG11 都不支持硬件加速，Spark 支持 FPGA 加速，ADB PG 采用 IR 技术，可以通过将 IR 翻译成对应的机器执行代码，从而支持 GPU 加速。



## 业界主流计算引擎对比

产品	ClickHouse	PG 11	Spark	ADB PG
执行模型	向量引擎	火山模型	编译执行	向量引擎
内存模型	列存	行存	行存	行/列存
即时编译	表达式级 LLVM	表达式级 LLVM	Stage JAVA	算子级 LLVM
硬件加速	不支持	不支持	FPGA	支持GPU

## 二、laser 计算引擎的核心技术

Laser 计算引擎的核心技术主要包括 5 大块:

1. 向量计算引擎
2. 行列式内存模型
3. JIT 加速
4. SIMD 指令加速
5. FUSION SCAN

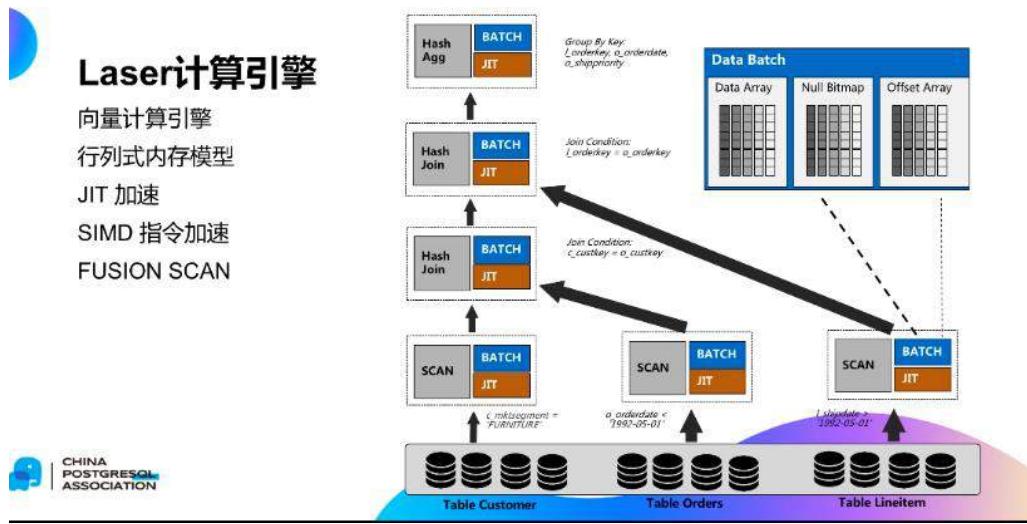
第一,向量计算引擎，传统火山模型中算子之间数据逐条交互，向量化计算引擎之间的交互的是 BATCH 数据，然后在每个算子内部，采用的列式多条数据并行计算，这种逻辑可以充分的发挥 CPU 流水线的作用。

在内存模型上，我们采用的是行列混存内存模型。在算子之间数据传递的是一个 mini batch，mini batch 内部采用的行列混存的结构。由于每列数据在内存里都是连续摆放的，所以每次计算一列都是内存的顺序访问的过程。

第三，针对复杂计算，采用 JIT 技术加速。可以利用 LLVM CodeGen 技术将复杂计算过程转换成 IR 代码，然后再通过 IR 代码翻译成机器码。它的好处是可以避免类型判断，减少虚函数的调用，从而提升计算性能。

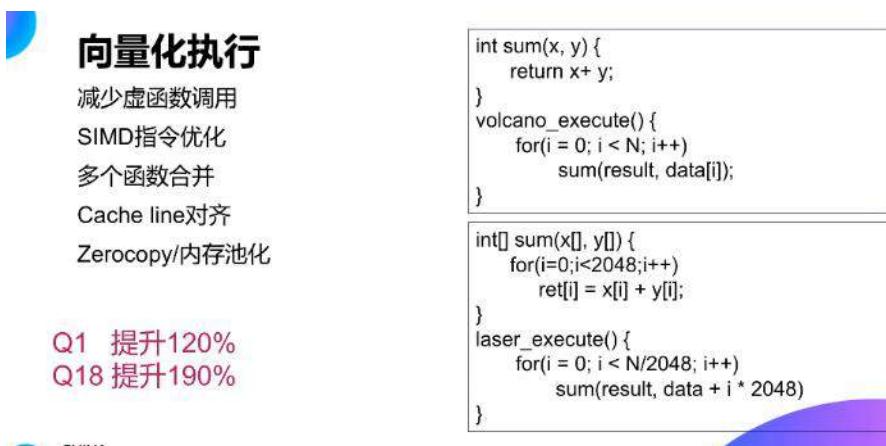
第四，针对一些简单场景（如：聚合场景、固定场景），利用现代 CPU 的 SIMD 指令，实现单条指令多条数据的并行执行，大幅提升这些简单场景的效率。

第五，针对列存场景，可以通过 FUSION SCAN 去减少无用列读取，避免无用内存的中间拷贝，从而提升列表的计算性能。



### (一) 向量计算引擎

下图是一个典型的火山模型下 SUM 算子的计算过程。对于火山模型，如果总共有  $n$  条数据，就会调用  $n$  次的函数调用。右边是向量化执行过程，sum 函数接收输入参数是一个数组，而不是两个变量。整个执行过程，数据按 2048 条分批处理，每 2048 条数据调用一次 sum 函数。



从这个例子中明显可以看出：

第一，Sum 函数调用从原来的  $n$  次变成  $n \div 2048$ ，减少了多次。

第二，在函数内部处理中，由于计算的数据是一个 batch，就可以充分发挥 SIMD 指令加速效果，实现单条指令多条语句的并行计算。

第三，可以针对一些算子，如 AGG 和 JOIN，可以将 AGG 和 JOIN 算子函数合并一个函数，可以进一步的减少虚函数调用，提升系统性能。

由于计算过程中全部采用数组计算，可以将计算过程中的数组使用内存池化技术管理。通过 MemoryPool 可以实现定长数据的内存的复用，避免频繁内存分配和释放，减少整个内存的碎片。在实际的 TPC-H 的测试中，使用向量化引擎后，Q1 提升了 120%，Q18 提升了 190%。

## (二) SIMD 指令加速

针对简单的加速，可以利用现代 CPU 的 SSE、AVX 指令，一条指令可以实现 512bit 数据计算。

我们对 TPCH 测试中的 Lineitem 表做性能的对比测试，在使用 SIMD 以后，整体的性能从原来的 1 秒多降低到了 200 多毫秒，有了 4 倍左右的提升。



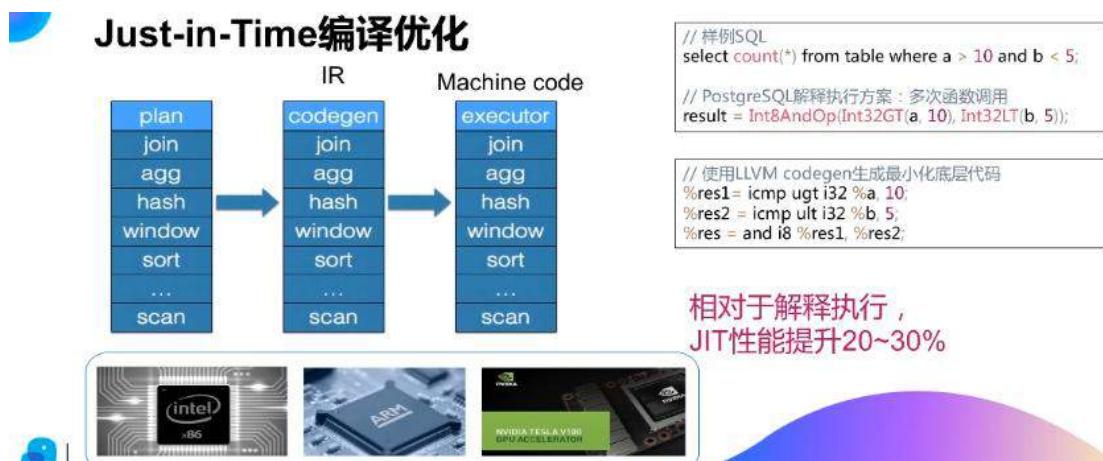
## (三) Just-in-Time 编译优化

ADB PG 不仅支持表达式级的 CodeGen，也支持算子级的 CodeGen。每个 plan 的 node 对应一个 CodeGen 单元和一个 Executor，CodeGen 单元根据 plan node 生成 code(IR)，Executor 根据硬件平台选择不同的后端，将 IR 生成对应的执行文件，并申请资源执行，最终的结果通过 master 返回给客户端。

如图所示，对于这样一个简单的表达式，`where a>10 and b<5`，如果采用解释执行，总共包括三次函数调用，1、`a>10`；2、`b<5`；3、`and` 函数。

如果使用 LLVM JIT 编译优化，上面的三个函数就编译成三条 LLVM 指令，避免了三次函数调用的开销。

在 TPCH 测试场景中，使用 JIT 加速后整体的性能提升了 20%~30%，并且在测试中发现，表达式越复杂，性能提升越明显。



#### (四) Fusion Scan 优化

Fusion Scan 主要优化是减少无用列的读取，并且尽量的减少无用数据的读取和内存的拷贝。

举个例子，如果要查询满足 `a` 大于 3 和 `b` 小于 6 的 `a, c*d` 的值，传统做法是对数据库中的每条数据数据，做 `a` 大于 3 和 `b` 小于 6 的条件过滤并且计算对应列的值。

Fusion Scan 的做法分成两阶段：

- 第一阶段是对过滤列做计算。把 `a` 列和 `b` 列的所有数据读出来，对每条数据进行判断。如图所示，满足条件的只有第一行第三行，我们把第一行第三行号保存在一个 bitset 中，同时把第一行和第三行的 `a` 列值也保存在 mini batch 中；

- 第二阶段是计算 project 列，由于满足这个条件只有第一行和第三行，我们只需要把 c 列和 d 列的第一行和第三行的值读取出来。同时为了避免中间结果的数据拷贝，我们直接去将 c 列和 d 列的值结果相乘，把结果保存在 mini batch 的第二列中。

在计算过程中，我们提前将表达式编译成 IR 代码，可以避免了多次表达式函数的递归调用。

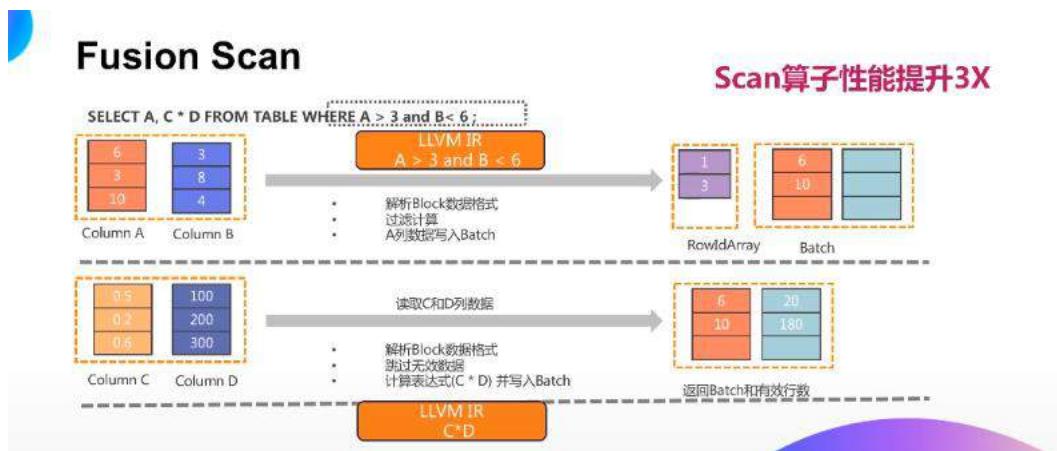
以上过程的主要优化点在于：

第一、避免无用列表 D、E、F、G 列读取；

第二、实现了按需读取，只有满足条件的 c 列和 d 列的里面内容才进行计算和 IO，不需要读那些不满足条件的数据。同时在 c 和 d 计算过程中，直接进行 c 和 d 的读取，无需内存中间结果的拷贝。

在实际执行过程中，Fusion Scan 结合列存块 block 索引做进一步的优化。block 索引中包含了数据块的 min 和 max，我们可以将 min 和 max 值和 where 条件做交集，只有这个交集非空的话，才会去真正读取 block 的值，否则可以直接跳过这个 block。

通过 Fusion Scan 的技术，在列存的场景 Scan 算子整体的性能有 3 倍以上的提升。



## (五) 算子实现-HashJoin

HashJoin 的向量化执行引擎，算法采用 Hybrid Hash Join，HashTable 采用开放链表数据结构。

HashJoin 的实现过程，主要包括：

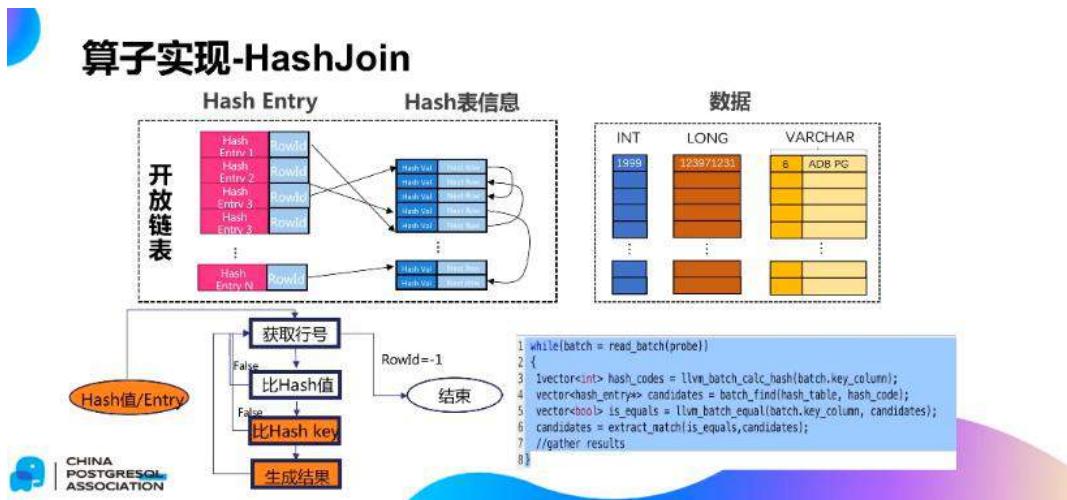
1. 把左表 probe 列的值取出来计算它的 Hash 值。
2. Hashcode 的值去模 entry 个数，获取对应的行号。
3. 对应行号里的所有的 entry 对象，比较它的哈希值，如果哈希值相同，再去比较 join key。
4. 如果 join key 相同，则代表 probe 成功，拼接左右表的对应列并生成最终的结果。

如何将这样的执行过程用向量化实现？

1. 从左表里面读取批量数据。
2. 使用 CodeGen 技术批量计算 hash code 值。
3. 根据 hashcode 值，批量查找 hash table，得到可能的结果集。
4. 批量比较左右表的 join 列值，如果匹配的话，则拼接左右表的对应列并生成最终的结果。

与原来行式的火山模型比起来，向量化执行主要差一点在于：

1. 按列计算；
2. 批量计算。



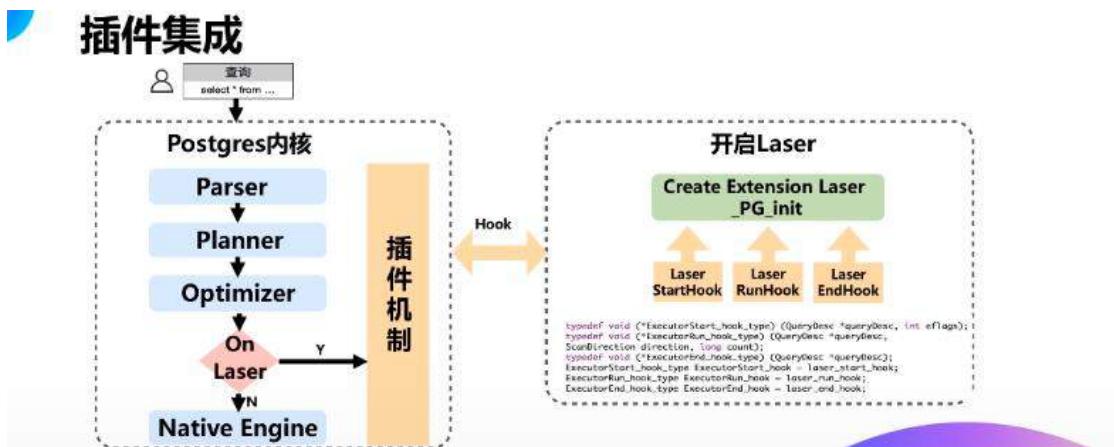
## (六) 插件集成

计算引擎如何与 ADB PG 代码融合?

ADB PG 同时支持 PG 计算引擎和 Laser 向量化计算引擎。优化器会自动根据 SQL 的 pattern 和扫描的数据量来决定是否使用 Laser。如果扫描数据量太少，则使用原生的计算引擎。如果数据量足够多，并且这些 SQL pattern 比较适合使用向量化执行引擎的，就使用 laser 计算引擎。

Laser 引擎的所有代码采用插件模式，代码独立。好处在于代码可以和原生代码之间完全是松耦合关系，不会影响 PG 的原生代码，同时可以复用里面的一些函数和数据结构。插件模式还带来一个好处，就是可以实现热升级。因为采用动态库方式，能够在不重启 PG dameon 进程的情况下，替换插件，完成升级 LASER 引擎。

唯一需要修改的是三个 stub 函数—ExecutorStart、ExecutorRun、ExecutorEnd。



## (七) TPC-H 结果

2020 年 5 月 20 号，我们完成了 TPC-H 30TB 场景测试，拿到了世界第一的成绩。相比于第二名微软 SQL Server 2019，整体性能提升了 290%，且成本只有 SQL Server 的 1/4。

通过性能指标统计，Laser 计算引擎对性能提升了至关重要的价值，相对于 PG 计算引擎，性能提升了两倍之多。细分计算引擎的各种优化，其中大半的性能提升都是来自于

向量化提升。其次是 JIT 加速，主要来源于表达式的计算。第三来自于是 Fusion Scan，针对列存场景下，我们通过 Fusion Scan，减少了内存的拷贝，减少了无用的读取。最后还有小部分来自于 SIMD 指令的提升。

## TPC-H结果

TPC-H - Advanced Filtered and Sorted Results							
Sponsor	System	Scale Factor	Performance (QphH)	Price/QphH	System Availability	Date Submitted	DB Software Name
Alibaba Cloud	Alibaba Cloud AnalyticDB	30,000	5,057,263	1.46 CNY	5/20/2020	5/20/2020	Alibaba Cloud AnalyticDB for PostgreSQL 6.0
CISCO	Cisco UCS C480 M5 Server	30,000	1,278,277	0.94 USD	11/4/2019	11/1/2019	Microsoft SQL Server 2019 Enterprise Edition 64 bit

相比第二名微软 SQL Server 2019，性能提升290%，单位成本仅为1/4



## 三、计算引擎的未来展望

整体的未来转化（以未来一年为计划）：

第一：2020 年 12 月，支持窗口函数，完成全算子的向量化改造。

第二：2020 年 3 月，完成网络 motion 重构。

第三：2020 年 6 月，完成算子并行执行优化，可以充分利用多核能力。

第四：2020 年 9 月，完成优化器适配。优化器不仅适配计划数据书，同时能够根据计算引擎来做动态识别，能够感知到数据的动态变化，再去动态去调整执行计划。

# UniqueKey:让你的查询跑的更快

作者 | 樊智辉 阿里云数据库技术专家

执行计划的好坏对于查询的性能起着至关重要的影响，那么一个执行计划的生成都和哪些因素相关？

首先是用户输入一个查询，数据库会综合表的定义约束信息，表上面的索引、数据的分布特征信息、硬件处理能力信息等等，最终会产生一个执行计划。这篇主要关注的是表的约束信息。



UniqueKey 是在一个结果集中能够唯一确定一条记录的一组表达式，对于一个复杂的查询，这个结果集通常是执行到一定阶段的中间结果集。

比如说有一个 5 张表连接，第一个阶段是每一个结果的单个结果集，在这个阶段每一张表都有自己的 UniqueKey，当第一张表和第二张表建立之后，建立的结果又有一个怎样的 UniqueKey？前三张表建立之后它的 UniqueKey 是怎么样的，整个过程中我们都可以判断它的 UniqueKey 是什么样，基于这些信息，优化器可以做出更加明智的决断！



UniqueKey 是在一个结果集中能够唯一确定一条记录的一组表达式，对于一个复杂的查询，这个结果集通常是执行到一定阶段的中间结果集。

**UniqueKey != Unique Index.**

## UniqueKey! =Unique Index

UniqueKey 和 Unique Index 有很强的关联性，通常 UniqueKey 的起点就是 Unique Index，但它们是不同的，首先我们从简单的例子看起：

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

PostgresConf.CN & PGConf.Asia 2020



### UniqueKey 举例

```
create table t (id int primary key,
    a int, b int not null,           create unique index on t(a);
    c int not null,
    d int);                         SELECT * FROM t;

SELECT * FROM t;                  a 是 UniqueKey 吗？

UniqueKey(id) --> 最简单的UniqueKey
```

**CHINA POSTGRESQL ASSOCIATION**



这种场景是 UniqueKey 的最简单的场景，那么 a 是唯一的 UniqueKey 吗？

答案是否定的，因为唯一索引允许有多个空值存在。A 并不是唯一的，因为 Unique Index 允许多个 NULL。排除 NULL 的因素即可。

如何去保证一个 Unique Index 出来的结果是不包含空的：

第一种场景是我们在定义表结构的时候，列上面就有非空约束，这样我们就可以很清楚的判断它里边一定不会有空值；第二种判断我们会根据用户输入的查询，就是说在表里它可能会存在多个空值，但是对于这个结果集里边 a 它一定不会有空值存在；第三种 select a from t where a in 一个子查询。然后如果 a 是一个空 a in 子查询，一定也不会返回 true。所以 a 它也一定不是空的；第四种就会相对比较不那么直观，它是 select a

from t where func(a)大于 10，并且函数它的 isstrict 的属性是 true，isstrict 的属性 true 它表达的意思是如果它的输入参数 a 是一个空，那么它的返回值一定也是空，那么它的返回值就一定不会大于 10。

我们也可以简单的说，唯一索引再加上一个非空约束，最终我们就可以转变为一个唯一键。例如下图所示：

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**PostgresConf.CN & PGConf.Asia 2020**

## 唯一索引与UniqueKey 的关系

```
c2=# select id, a from t;
 id | a
----+---
 1 | 1
 2 | (null)
 3 | (null)
(3 rows)
```

A 并不是唯一的，因为Unique Index 允许多个NULL. 排除NULL 的因素即可。

```
1. SELECT a FROM t WHERE
   a is not null;
2. SELECT a FROM t WHERE
   a > 10 OR a < 10;
3. SELECT a from t WHERE
   a in (select ... );
4. SELECT a FROM t WHERE
   func(a) > 10;
// func.isstrict = true;
```

我们再看一个组合索引的情况，如下图所示：

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**PostgresConf.CN & PGConf.Asia 2020**

## 精准的UniqueKey

```
create unique index on t(b, c); -- 根据表定义，b 和 c 都是非空。
```

```
SELECT * FROM t WHERE c = 1; -- b 是 UniqueKey ; c 不是必须的。
```

```
c2=# select b, c from t;
 b | c
---+-
 1 | 1
 2 | 1
 2 | 2
(3 rows)
```

```
c2=# select b, c from t
      where c = 1;
 b | c
---+-
 1 | 1
 2 | 1
(2 rows)
```

```
c2=# select b, c from t
      where b = 2;
 b | c
---+-
 2 | 1
 2 | 2
(2 rows)
```

至此为止我们讨论的都是唯一键和单表查询，实际上更复杂的场景来自于表连接。我们来看一个两张表连接的例子，如下图：

**PGConf.Asia 2020** | **PostgresConf.CN & PGConf.Asia 2020**

## UniqueKey和表连接

- `SELECT t.id from t, t2`
- `WHERE t.c = t2.a; -- UniqueKey (null).` T1.c = t2.a 会使得t.pk 一行变多行

```
uniquekey=# table t ;
 id | a | b | c | d
-----+---+---+---+
 100 | 1 | 1 | 1 |
(1 row)
uniquekey=# table t2;
 id | a
-----+
 1 | 1
 2 | 1
```

```
# SELECT t.id, t.c, t2.a from t, t2
WHERE t.c = t2.a;
 id | c | a
-----+
 100 | 1 | 1
 100 | 1 | 1
(2 rows)
```

CHINA POSTGRESQ<sub>L</sub> ASSOCIATION

对于这样的一个结果集，它里边是没有唯一键的，那么对于一个表查询来说，它的唯一键在哪些场景下面是可以继承下来的呢？其他场景会产生 UniqueKey 的例子，如下图：

**PGConf.Asia 2020** | **PostgresConf.CN & PGConf.Asia 2020**

## UniqueKey 举例3

- `SELECT d FROM t GROUP BY d;` UniqueKey (d).
- `SELECT DISTINCT d FROM t;` UniqueKey (d)
- `SELECT d FROM t UNION SELECT d FROM t;` UniqueKey (d).

CHINA POSTGRESQ<sub>L</sub> ASSOCIATION

在现实场景中的 UniqueKey 会隐藏在一个复杂查询中的一部分。我们需要在一个执行计划的产生过程中，在任何一个阶段准确判断出 UniqueKey 的属性，以便优化器产生更优的执行计划。



现实场景中的 UniqueKey 会隐藏在一个复杂查询中的一部分。

## UniqueKey 的用途

- 唯一键的用途？
- 第一减少半连接；
- 第二减少无用连接；
- 第三可以去除一些不需要的 Distinct 操作。

### (一) 减少半连接

如下图所示：

### 减少半连接

```
postgres=# table t;
a | b | c | d
---+---+---+---
 1 | 1 | 1 | 1
 2 | 2 | 2 | 2
 3 | 3 | 3 | 3
 4 | 4 | 4 | 4
 ...
 9 | 9 | 9 | 9
10 | 10 | 10 | 10
(10 rows)
```

A IS PRIMARY KEY.

```
postgres=# table t2;
a      | b      | c      | d
-----+-----+-----+-----
 1 |     1 |     1 |     1
 2 |     2 |     2 |     2
 ...
 1000000 | 1000000 | 1000000 | 1000000
(1000000 rows)
```



PCC POSTGRESCONF  
CN 2020

PGConf.Asia

11.17-11.20

PostgresConf.CN &  
PGConf.Asia 2020

## 减少半连接

```
postgres=# explain analyze select * from t2 where b in (select b from t);
          QUERY PLAN
```

```
Hash Semi Join (cost=1.23..18032.46 rows=10 width=16)
  Hash Cond: (t2.b = t.b)
    -> Seq Scan on t2 (cost=0.00..15406.10 rows=1000010 width=16)
      -> Hash (cost=1.10..1.10 rows=10 width=4)
        -> Seq Scan on t (cost=0.00..1.10 rows=10 width=4)
Planning Time: 0.280 ms
Execution Time: 372.128 ms
(8 rows)
```

为了讨论方便，关闭了一个  
会产生干扰的功能，需修改  
源码

CHINA  
POSTGRESQL  
ASSOCIATION

这里的 t2 有 100 万条数据，我们要把这 100 万条数据全部扫描一遍，根据执行情况，可以看到它执行的时间是 372 毫秒，为了讨论方便，我关闭了优化器的一个干扰功能，大家在自己模拟的时候有可能看到不同的场景，这也是有可能的。

PCC POSTGRESCONF  
CN 2020

PGConf.Asia

11.17-11.20

PostgresConf.CN &  
PGConf.Asia 2020

## 减少半连接

```
postgres=# explain analyze select * from t2 where b in (select a from t); -- (a) is UniqueKey
          QUERY PLAN
```

```
Nested Loop (cost=0.42..85.62 rows=10 width=16)
  -> Seq Scan on t (cost=0.00..1.10 rows=10 width=4)
  -> Index Scan using t2_b_idx on t2 (cost=0.42..8.44 rows=1 width=16)
    Index Cond: (b = t.a)
Planning Time: 0.373 ms
Execution Time: 0.235 ms
(6 rows)
```

CHINA  
POSTGRESQL  
ASSOCIATION

整个过程中我们只对于 t 整个循环了 10 次，而不是原来的 100 万次，它的执行时间也从原来的 370 多毫秒下降到了 0.235 毫秒。

但为什么前面的就不能这样做，后面的这个就可以这样做呢？

原因在于 a 它是一个唯一键，而 b 不是唯一键，在 a 是一个唯一键的场景下面，我们可以做一些查询的改写。

### 半连接的转化过程如下：

本来是 `select * from t2 where b in ( select a from t )` 它可以将这个半连接转换成一个正常的 inner join; `select t2.* from t2, t where t2.b = t.a`, 这里的 `t2.b = t.a`, a 又是一个 UniqueKey 通过这样的教育，并不会让原来的 t2 的一条结果变成多条结果，所以这个语义还是相等的。当把它们转变成一个正常的连接以后，优化器就会同时去看，拿着 t2 去教育 t 生成一些执行计划，也会拿着 t 去教育 t2 去生成另外的一批执行计划。

在这里我们最后优化器会认为拿着 t 去教育 t2 可以得到一个更好的执行计划，也就是我们前面看到的，扫描外表只有 10 行，整个过程中只需要循环 10 次即可达到我们的目的。

这里查询的改写依赖于 a 它是一个唯一键。



### (二) 减少无用连接:

最终出现的这样的一个查询的逻辑分析的一个结果，就是在这种场景下面，t2 里边的任何一个结果集都不会被重复，并且每一条数据都应该被返回，并且都不会被重复，这个结构可以直接去扫描一下，t2 跟 t1 没有任何的关系。

最终我们可以看到对于一个 left join 的这样的一个操作，产生的一个执行计划，仅仅是对 t2 进行了一个全表扫描和 t 没有任何的关系。

那么用户为什么会写一个这样的查询？会写出这种没有用的连接语句呢？

实际上这是一个使用场景，假设我们有一个表，它有很多的属性，这些属性全部放在一起，表就会非常的大，而这个时候我们可以把一个宽表拆分成很多的窄表，然后拆分成窄表以后，我们如果是只去访问其中的个别列的话，其实是访问了一个更小的结果集，它的性能就会更好。

但有一样是不变的，就是说每个人如果想去获得更多的列，他就要去写很多不同的 Join，这个时候其实有一种做法，我们创建一个视图，把所有的表全部通过 left 键把它给连接起来，这个时候用户看到的是一个像宽表一样的，它只有去查询这个视图，去检索它需要的量。如果说它只检索很少的量，优化器就会直接通过这项技术把那些无关的表的 left 那些 Join 全部给去掉，性能也会变得很好，用户也会非常的方便。在 PG 内部的优化视图里边，内置的一些视图里边很多都采用了这样的技术。

**PCC POSTGRESCONF  
CN 2020**

**PGConf.Asia**

11.17-11.20

PostgresConf.CN &  
PGConf.Asia 2020



## 减少无用连接

```
select t2.*  
from t2 left join t on t2.b = t.a;  
( t.a is UniqueKey )  
  
1. Join 会使得一方的结果集过滤掉或者一行变多行。  
2. Left join 保证了左表的数据一定不会被过滤掉  
3. Join 右表的条件又是UniqueKey 所以左表的数据一定不会被重复。  
4. 最终只需要扫描一遍t2 即可，完全忽略掉t1.
```

```
postgres=# explain (costs off) select t2.*  
from t2 left join t on t2.b = t.a;  
QUERY PLAN  
-----  
Seq Scan on t2  
(1 row)  
--  
实用场景：  
将一个宽表拆分成多个窄表，  
然后通过left join 来创建视图，  
应用按需检索列。
```

CHINA  
POSTGRES  
ASSOCIATION

- Join 会使得一方的结果集过滤掉或者一行变多行。
- Left join 保证了左表的数据一定不会被过滤掉。
- Join 右表的条件又是 UniqueKey，所以左表的数据一定不会被重复。
- 最终只需要扫描一遍 t2 即可，完全忽略掉 t1.

实用场景：将一个宽表拆分面多个窄表，然后通过 left join 来创建视图，应用按需检索索列。

## 消除无用的 Distinct 操作

正常的 Distinct 的操作，select distinct a c d 首先要得到一个结果集，然后对结果集进行一次去除，如果它的数据量是很大的，这个去除它本身也可能要花费很长的时间，但是如果说我们可以判断出这个结果集已经是唯一的了，去重操作是可以不做的。看右边的查询，它是 select disstind b c d 那么 a 本来就是唯一的，加上 CD，它自然还是唯一的 distinct 的操作。可以完全忽略这样的操作，省掉了一个代价很高的节点。

这样一个功能的意义在现实生活中，在平时开发应用程序的时候，有可能也会进行很多的表连接，当创建完表连接以后，其实也很难判断它到底是不是唯一的，这个时候我们就会去加一个 Distinct 的操作，同时有一些框架也比较适合去加这种 Distinct 的操作。

如果数据库的内核识别不了这些唯一键，它就必须要去生成一个去除的节点，可能要消耗很大的资源，但是如果能够识别这样的一个信息，这个节点我们就可以直接去除，从而达到很好的性能。如下图所示：

The slide is titled "消除无用的Distinct 操作" (Eliminate unnecessary Distinct operations). It compares two PostgreSQL query plans:

**Left Plan (Original Query):**

```
postgres=# explain (costs off ) select
  distinct b, c, d from t;
```

QUERY PLAN

```
-----
```

HashAggregate  
Group Key: b, c, d  
-> Seq Scan on t  
(3 rows)

**Right Plan (Optimized Query):**

```
postgres=# explain (costs off ) select distinct
  a, c, d from t;
```

QUERY PLAN

```
-----
```

Seq Scan on t  
(1 row)

**Annotations:**

- A pink box highlights the word "distinct" in the first query.
- A pink box highlights the word "a" in the second query.
- A pink box highlights the text "A is UniqueKey" below the second query's plan.
- A pink box highlights the text "现实意义：" and "用户总喜欢去加一个Distinct 去去重尽管。" at the bottom right.

Logos for PGConf.Asia 2020, PostgresConf.CN & PGConf.Asia 2020, and China PostgreSQL Association are visible at the top and bottom of the slide.

阿里云在今年1月份的时候提交了一个patch，patch可以更加系统化的去识别，在一个复杂的UniqueKey，它的唯一键是什么？

相对提供了一个非常完整的监测机制，同时又增强了一个Distinct的操作。将UniqueKey用于Distinct的这种场景下边。这个功能有望会在PGConf和入到PGConf14这个版本里边。

The screenshot shows a feature request page from PGConf Asia 2020. The title of the request is "Erase the distinctClause if the result is unique by definition". The request details are as follows:

Edit	Comment/Review	Change Status
Title	Erase the distinctClause if the result is unique by definition	
Topic	Performance	
Created	2020-01-31 12:42:25	
Last modified	2020-10-01 04:29:37 (1 month ago)	
Latest email	2020-10-12 02:33:17 (3 weeks, 2 days ago)	

At the bottom left, there is a logo for the China PostgreSQL Association.

## PolarDB(兼容 PG/Oracle 语法)

特点：

- 计算存储分离
- 计算节点点瞬间扩容
- 存储节点按量收费
- 高度兼容 Oracle 语法



最后介绍一下 PolarDB(兼容 PG/Oracle 语法)，首先它是一个一写多读的架构，但和传统的主备不同，它采用的是计算存储分离的架构，并且底层是一个共享的文件系统 Polar Store。

这种架构的好处是计算存储分离，计算节点可以做到瞬间的扩容，并且存储节点也可以按量付费。

在语法层面，我们高度兼容 Oracle 语法，用户可以轻易的从 Oracle 数据库迁移到 Polar Store，在读节点和写节点之上，我们还有一个 MaxSQL 节点，用户只需要连接 MaxSQL 节点，就会根据当时的准备延迟情况，将请求转发给读写节点或者只读节点，从而充分的利用硬件资源。

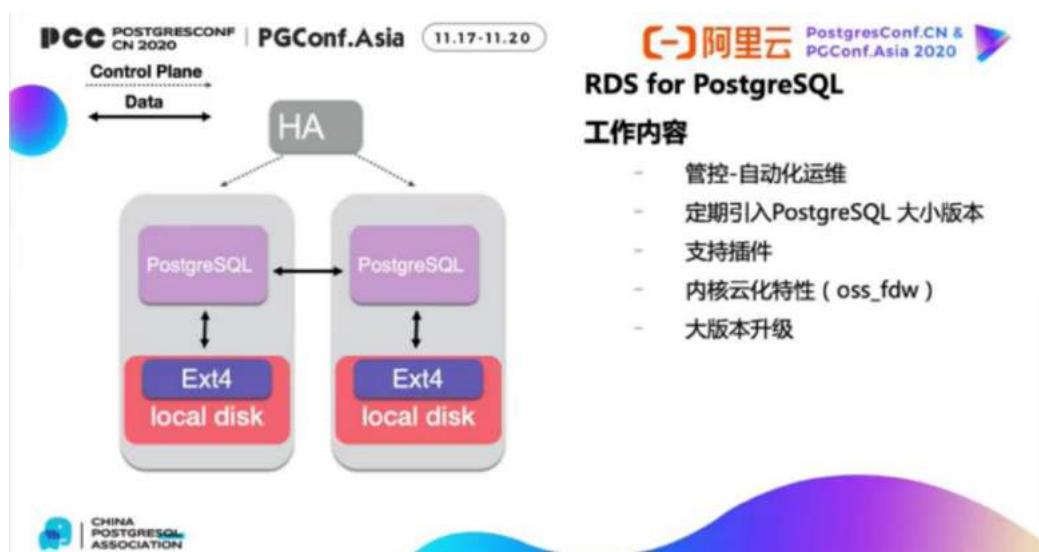
# Implementation of Global Temp Table

作者 | 曾文旌 阿里云数据库高级技术专家

## 一、工作介绍

开发特性过程以及实现特性技术细节：

第一个和 PostgreSQL 相关的云产品是 RDS PostgreSQL，是把主备流复制架构的 PostgreSQL 放到云上，做自动售卖。



内核层面，定期升级 PostgreSQL 的大版本和小版本，根据用户需求支持各类的插件以及开发云上独有的插件。比如打通 PostgreSQL 和其他云产品数据通路的插件(Oss\_fdw)。

这个版本对 PostgreSQL 的内核改动并不大，但和 PostgreSQL 的社区交流较多。

当用户提出的新的需求时，首先会从社区中寻找灵感，如果社区在某个版本中支持了该应用，则会建议用户直接使用这个版本。如果没有，将会做定制化的开发。

2018 年，开始自研了云原生数据库 PolarDB，而 PolarDB 也有支持 PostgreSQL 引擎的版本。



该版本有两个主要的特点：

- 首先，它基于计算存储分离的架构，可以同时支持很多的企业级存储。在公共云上，支持 PolarStore 分布式的存储，在线下也支持传统的块存储。这使得存储上的扩展性有了一个大幅的提高；另一方面，计算节点也支持多读一个架构，它可以实现扩展读的能力，使计算能力可扩展，相较于 RDS PostgreSQL 有了比较大的提高。
- 其次，自研了很多高级的 SQL 特性，使 PolarDB for PostgreSQL 在企业级的应用场景里，或者传统的数据库使用场景有一个高度的 Oracle 兼容性，更加易于在复杂场景中替换 Oracle 应用。

## SQL 功能

SQL 大致分为以下几个类型：

第一部分的工作相对简单，比如支持某类驱动，支持功能性的函数和视图，比如数学函数、字符串处理函数等。

第二部分是 SQL 层的一些特性例如聚集函数，或 PL/SQL 里的特性。

相对复杂的第三部分是表级的特性以及 SQL 优化相关的特性。



## 二、Global Temp table 的实现分享

### 三个部分

- 站在 Postgre SQL 的肩膀上
- Global Temp Table 的设计
- Global Temp Table 在社区 Review

什么是全局临时表？



用普通的表做类比，对数据库中的普通表 a 而言，不同的 application 连接到数据库中查看表 a 的时候，能够看到表 a 中的所有的数据。

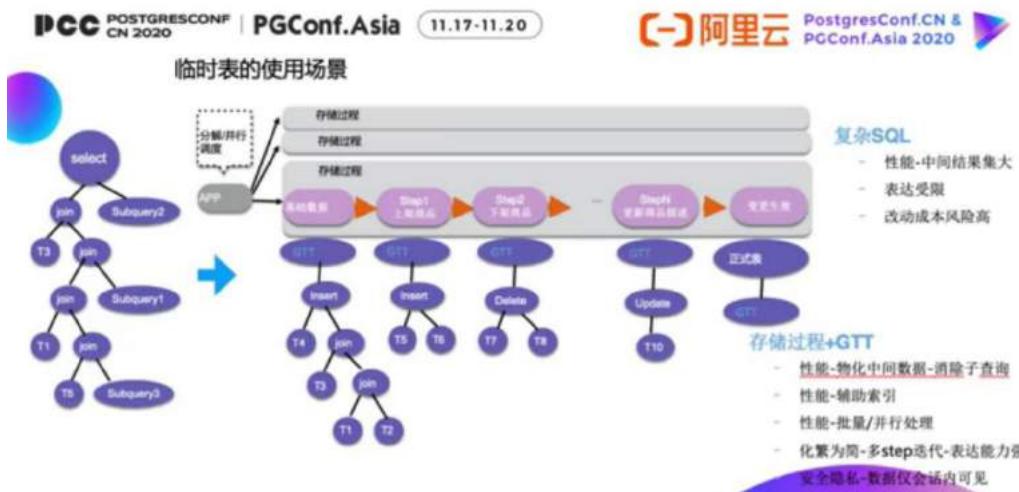
而对于全局临时表而言，不同的连接到数据库中，都能看到临时表 b。但是，对于临时表 b 中的数据，每个连接都只能看到自己的这一份数据，看不到其他连接中的数据，那么这就是临时表的定义。

对于会话中的数据生命周期而言，可以选择绑定事物或者会话，换一种说法也就是我们可以选择这张表中的会话数据，使得在事务提交的时候被自动清理，或者是在会话退出的时候被自动清理。**这就是全局临时表的定义。**

### 全局临时表的使用场景？

一般情况下我们使用多表 join 再叠加子查询的方式来实现对复杂数据场景的数据处理。这种做法业务的开发难度会随着 SQL 的复杂程度的提高而急剧的提高，最终业务 SQL 将转变得不可运维。

自然的，使用存储过程加临时表处理数据的方案，就成为了一个更加合适的选择。



具体来说是把复杂 SQL 中的逻辑进行拆分，拆分成存入过程中的多个步骤，进而分步骤完成，每个步骤之间，使用临时表来做数据的交换。

这样可以使它发挥出临时表的很多的优势。

比如，临时表可以使用辅助索引来做数据处理的加速，由于临时表在会话之间不共享数据，也很容易开发出并行数据处理的逻辑来进一步的加速业务，整体的开发难度不会随着业务的复杂而急剧的提高。

最后，由于临时表中的数据支持在特定的时机被自动化清理，那么开发者不用花费额外的精力去维护临时表的定义信息，维护临时表达索引信息，以及主动的清理临时表中的临时数据。

大量的用户使用 Global Temp Table 处理自己的业务数据。我们调查发现，高达 80% 的 Oracle 数据库用户使用了全局临时表特性，有超过 50% 的客户，使用了存储过程加临时表结合的方式来完成自己的业务逻辑。在调研 Global Temp Table 的现状时发现，该特性还属于 SQL 标准。几乎所有的商业数据库都支持这个特性。

在 PostgreSQL 社区里面，从 2007 年到 2015 年都陆续有多个内核开发者，尝试实现该特性，但终究没有成功。这个特性当前放在 PostgreSQL 的 TODO LIST 里。

DbType	support GTT
Oracle	支持
SQLserver	支持
EnterpriseDB	不支持
PostgreSQL	不支持
MySQL	不支持

## 如何支持 Global Temp Table 特性？

达成一致的结论：

- 相对于社区已经支持的临时表，我们可以叫它 Local Temp Table，Global Temp Table 通过持久化表定义的方式，减少系统表的膨胀。
- Global Temp Table 的一些状态的信息，不用存放在系统表中，可以存放在每个会话的内存机构中。

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**阿里云 PostgresConf.CN & PGConf.Asia 2020**



### 来自社区的观点：

- 持久化和共享临时表定义可以避免系统表膨胀
- 通过设置新的 `relopersistence` 和重用 `local temp table` 的存储形态来支持 `global temp table`. -- **Andres Freund**
- 使用会话内存上下文存储GTT相关的信息 -- **Pavel Stehule**
- 扩展 `relfilenode` 来支持 `global temp table` -- **Craig Ringer**
- 本地数据统计信息是否必要，还没有结论
- `global temp table` 的中数据的事务信息处理方案，没有结论

### 未能达成一致的问题：

- 对于这个特性在 Global Temp Table 的每个会话中的数据，是否应该有一份为它支持的独立的统计信息？
- 对于数据和数据的事务信息应该怎么样存储和处理？是否应该为临时表这么一种特相对特殊的表开发一种全新的更加简洁的存储方案来存放它？

这些问题都暂未达成一致意见。

**PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20**

**阿里云 PostgresConf.CN & PGConf.Asia 2020**



### 计划中的global temp table 和社区已经支持的local temp table对比

	global temp table	local temp table
no data share across session	yes	yes
Support On COMMIT (DELETE ROWS /PRESERVE ROWS ) with same behavior	yes	yes
share table schema across sessions	yes	no
definition deleted after a session exit	no	yes
Support ON COMMIT DROP	no	yes

这里总结了相对于社区已经实现的临时表 Local Temp Table 和我们开发的 Global Temp Table 在需求层面的一些差异点。

可以看到 Local Temp Table 的表定义和数据，它的生存的周期基本上是一致的，但是 Global Temp Table 的表定义和本地数据则不一致，

GTT 的表定义是持久化的，但是每个会话中的数据各有自己生命周期，这一点带来了一系列的问题，即是 Global Temp Table 最大的实现难点。

要实现 Global Time Table 需要解决 4 个维度的难题：

1. 存储层面
2. 元数据层面
3. 统计信息层面
4. 事务层面



解决难度从低到高，这 4 个维度分别又可以展开细化成一些子问题，例如

1. DDL 语句的实现、锁问题、Cache 问题。
2. 统计信息和优化器中的统计信息处理问题。
3. 事务信息的处理问题和数据的可见性问题、vacuum 问题等等。

这 4 个方面的问题，前两点存储层面和原数据层面社区已经有了一个比较明确的方向，实现思路都已经非常明确，但是表中数据的统计信息的处理以及事务信息的处理还没有一个明确的结论。

以四个方面具体的阐述一下。（以下使用 GTT 来简称 global temp table，用 LTT 来简称社区与已经支持的特性 local temp table）



### 设计的第一部分：存储

这部分涉及 GTT 的数据存储的文件格式和用于暂存数据的 Buffer d 的设计。

在这两部分的设计上大体都复用 LTT，但是不同之处是在于同一张 GTT，在不同的会话中都可能有一份独立的存储和 Buffer，而 LTT 只会有一个会话使用。

即在那些已经使用过的会话中有一份存储和 Buffer，没有使用的会话中则没有，那么这就需要使用一套 DDL 语句来管理每个会话中的数据。为了达到使用 DDL 语句来管理 GTT 数据的目的，我设计了一个全局的哈希表，放在了共享内存上，用于追踪和管理 GTT 在每个会话中的存储信息。当用户使用 DDL 管理 GTT 或在每个会话中发生第 GTT 的第一次 DML 时，都会维护这个全局的哈希表。有了哈希表，普通表所支持的 DML 和 DDL 这些

语句都可以在 GTT 上全部实现。

极端的情况下,在一个 DB 中,我们同时管理 1 万个 GTT 和同时建立 1 万个连接,这会产生 1 万个会话,这样的场景最多也只会使用到 12 兆的物理内存,是一个可以接受的开销。

这样的设计保证了用最低的资源实现了 GTT 的整套 DDL。

而为了维护 GTT 这一份元数据和多个会话中的数据的一致性,增加了一个限制,也就是说当多个会话都在使用这张 GTT 的时候,那么它无法对 GTT 做 Drop Table 和 Alter Table。

### 设计的第二部分：元数据的管理

如何给一个 GTT 创建索引?

假设会话 1 在 GTT A 上正在创建一个索引,而其他的会话也可能同时使用表 A,如果会话 1 完成了本地索引的创建并把索引的元数据更新到系统表中,那么其他的会话也能马上看到 A 上的索引。

在会话 3,这时候会话 3 的 GTT 有的一份独立的数据,但这个数据并没有被新建的索引所维护,那么我们应该在触发索引扫描的语句中去新 Build 索引 A 吗?这样 DML 中 Build 索引会存在问题吗?



事实上对于这一个问题，我觉得是有问题的。于是在第一版的设计中就禁止了在多个会话同时使用到表 A 的时候，为表 A 创建索引。

最新的设计：在会话 1 中，当表 A 的索引完成创建后，索引信息更新到 catalog 中，其他的会话会看到索引信息，这时候其他会话会根据自己当前的情况，选择是否立即使用 A 上的索引。

### 举例说明

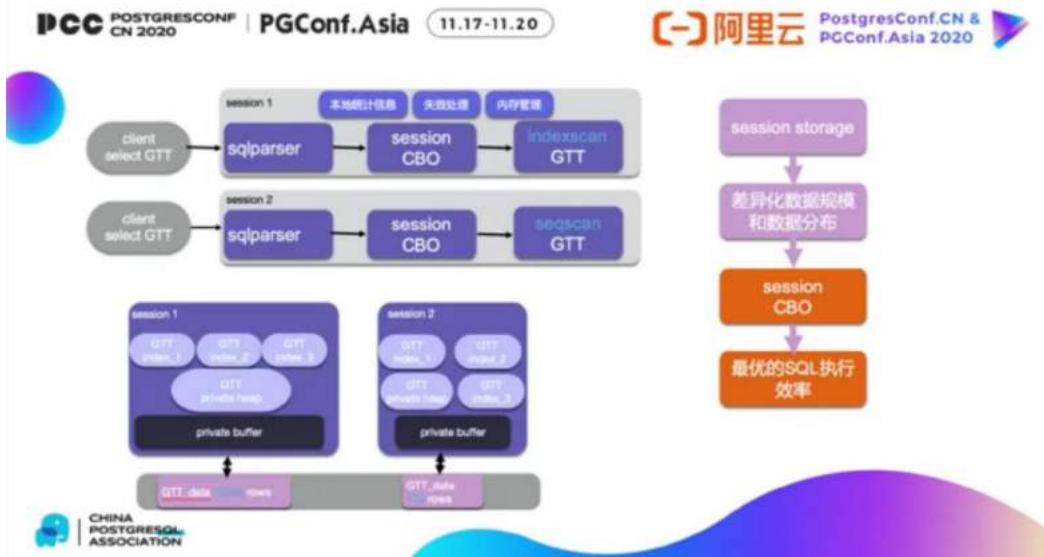
1. 如果会话 2 中表 A 没有存放数据，这时激活这个表上的索引是安全的，在这之后表 A 又来了新的数据，索引是被维护起来的，那么索引是有效的和可用的。
2. 如果会话 3 中表 A 已经有数据了，不可能在一个查询语句或 DML 语句触发 Build 一个索引这样的动作，这是就要失效会话 3 中的索引，当会话 3 中的 GTT 的数据被清理过后，即就是触发了 on commit 子句，在事物提交的时，索引又重新被启用了，再来新的数据，也能及时被索引记录，索引也能用于 GTT 的查询加速。

这个设计相对就是简单粗暴的禁止掉索引创建，是一个比较完善的设计。

### 设计的第三个部分：统计信息

由于每个会话中数据的彼此独立，数据的存取又使用了会话内的 local buffer，这样设计使得没有一个会话有能力看到同样一张 GTT 表的全局数据。同时我们知道 PostgreSQL 的查询使用代价模型，好的查询性能基于准确的数据统计信息，这就使得 GTT 需要为每个会话内的数据提供一份的独立的数据统计信息。

同时这些本地数据的统计信息不需要放到全局的系统表中。

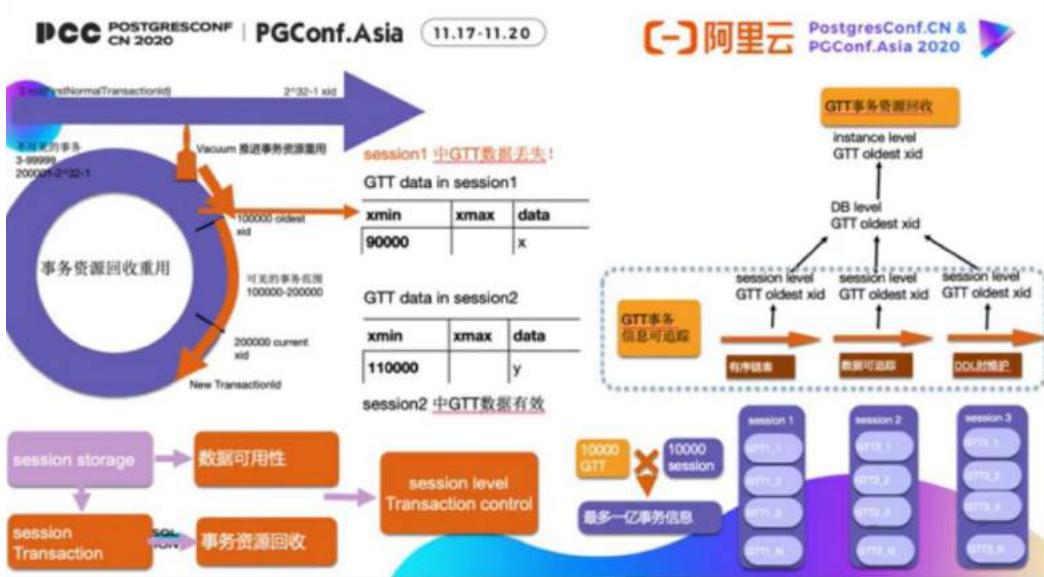


基于这个设计思路，我们设计了一个会话级的本地哈希表，用于保存这些 GTT 表的统计信息，提供给本地会话内的优化器使用，帮助优化器做 GTT 的查询优化，这就使得 GTT 相关的 SQL 有了一个优秀的查询执行路径，保证了 GTT 相关 SQL 的高性能。使 GTT 的查询性能和普通的表抹平了差异。

#### 设计的第四个部分：GTT 事物信息的处理

按照我的计划，GTT 和普通表都将使用相同的存储设计，不需要实现一个新的存储的方式。我们知道 PostgreSQL 的传统行存数据的设计，每一行数据的头部都保存了数据的事务信息。

每个事务号，即 XID 占用 32Bit 存储空间，XID 是一类系统资源，一个实例中只能同时存在 2 的 32 次方减一个 XID。



事务资源的回收通过 Vacuum 机制完成，这是 PostgreSQL 独有的机制。

基于前面讲到的存储部分的设计，GTT 在每一个会话中的数据是独立的，它们自己独立的事务信息，自然需要被维护起来。

数据的可见性，依赖数据内的事务信息，对于具体某个事务的状态（也就是这个事务是否提交或回滚），存储在 CLOG (commit log) 中。

如果事务状态信息，也就是 CLOG，一旦被 Vacuum，GTT 在会话中的数据，由于丢失了事务状态信息，这份数据则不用了，也等于是造成了数据的丢失。

那么，我们将会话中数据的事务信息（这份数据最老的 XID relfrozenxid）也同样保存在本地的哈希表中，同时维护会话级的 GTT 的 Oldest relfrozenxid，并且把它放到全局的共享区中。

这样 Vacuum 在清理 CLog 时，就有机会考虑到 GTT 的事务信息了。

我们用这个设计保证 GTT 数据的完整性。

另一个方面，从全局的角度也能实时看到某个会话中 GTT 中的数据是否包含了老旧的事务信息，需要被清理，也利于做事务信息的全局管理和清理。

# RDS PostgreSQL 管控体系介绍

作者 | 王涛 阿里云数据库高级技术专家

## 一、RDS PostgreSQL 产品架构演进

RDS PostgreSQL 总体上来说，经历过三代架构。

### 第一代架构：

第一代架构是传统的物理机模式，左边是一台 master 的物理机，右边是一台 standby 物理机，它们主从之间通过 Streaming 流复制来做，然后最上面是有 SLB，SLB 是当主节点健康状态下把 SLB 的流量切到主节点，如果当主节点异常的情况下，则把 SLB 切到备节点，这是 HA 的逻辑。

但是纯物理模式有几个缺点：

- 如果备节点，物理机宕机的情况下，数据拷贝的时间是很漫长的。
- 弹性能力比较差。

例如：比如一个变配操作 4C4G 变成 8C16G，如果物理机本地有资源的情况下就可以很快完成变配，但本地物理机没有资源的情况下，就需要一个跨机迁移，进行腾挪，而腾挪的时间非常漫长。



## 第二代架构：

第二代产品的特点是存记分离模式，看左半边这张图，首先有个 master 节点在中间，master 节点会通过 streaming 的复制方式把数据传给两边的计算节点，一个是 standby 节点，一个是 readonly，每个节点下面都挂着分布式存储，保证数据的一致性，独立性。

上面有个 SLB，SLB 连接主节点，还有个 SLB RO 节点，这个主要连着只读节点，这样有几个好处。

首先是底层分布式存储，分布式存储会带来很多运维上的提升，比如说数据的搬迁，数据的克隆，都可以很快的完成。

其次就是计算节点，计算节点由于是 ServerList，所以在迁移的过程中不需要搬数据；能够快速的弹升动作。



右图可以看到是 SLB 挂了一个 master 节点，然后下面是分布式存储，这个架构就是 RDS PostgreSQL 基础版，首先一个计算节点下面挂着一个分布式存储，上面是 SLB，分布式存储有个最大的特点：保证数据的可靠性。

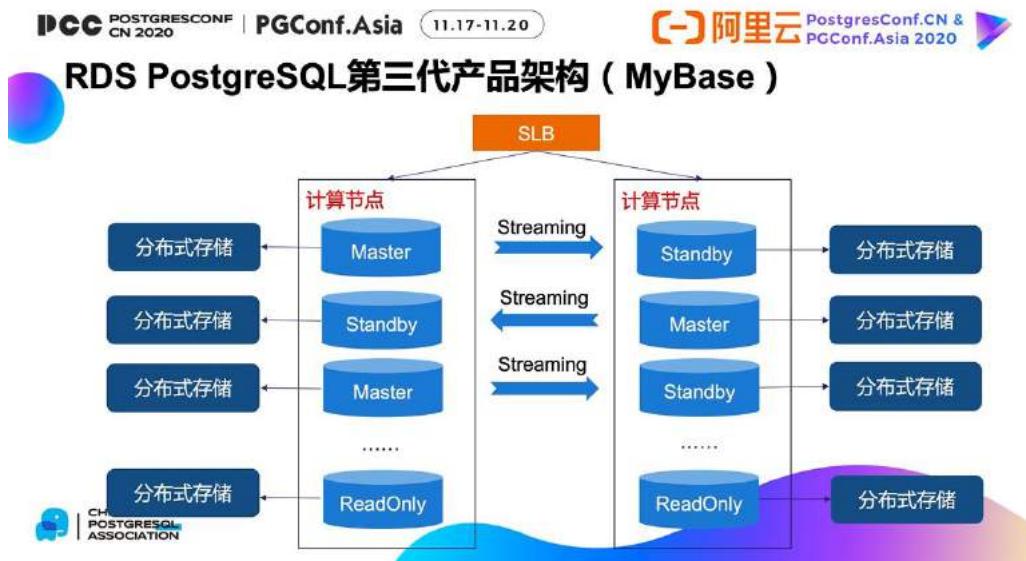
当主节点挂掉的情况下，只要重新找一个计算节点，把数据库和盘关联起来，把盘挂上去就完成了，所以说基础版虽然在 HA 能力会变弱，但是其数据的可靠性还是有保证的。

### 第三代架构 ( MyBase )

MyBas 产品是 2019 年在 My SQL 产品上孵化出来的，今年三月份 MyBase 在 PG 产品上进行了孵化。

大家可以看到一个特点，原来一个计算节点只能挂一个数据库节点，现在一个计算节点可以挂多个数据库节点。

它有什么好处呢？



首先，左边计算节点分布着 master、standby、master、readonly；右边计算节点分布着 standby、master、standby、readonly。

如图所示现在用户可以自定义左边的计算节点挂多少个 master，右边的可以挂多少个 master，这样的话就把资源调拨的能力下沉给用户，用户可以自己调度解决，这是第一点；第二点，用户可以根据业务的压力进行动态的调配。

但是数据库是一个 IO 密集型的计算，所以如果共享一块数据盘，那么共享数据盘可能会出现性能问题。这个可以怎么优化和避免？

首先，每个数据库节点都挂了分布式存储，每个分布式存储都是独立的，这样就保证 I/O 之间是隔离的，不会互相影响。这是 MyBase 产品一大亮点，就是用户成本会更低、更经济，同时具备自主的调度能力。

其次，MyBase 还有个特点就是用户可以自己登陆到计算节点，看到数据库的状态、信息等等，甚至可以自己去写一些小脚本去监控数据库，这样是加大了用户对数据库的把控力。

### RDS PostgreSQL 产品迭代回顾

目前经历了三代产品：

- 第一代是纯物理机模式；
- 第二代是存记分离模式（我们更倾向强调快弹能力，快速的响应能力）；
- 第三代提供的是用户的自主可控能力（比如说自主调配，自主查看数据库）。



## 二、RDS 产品的安全体系



## 第一大块：运维合规

运维合规，首先是白屏化运维，授权审计，这一块主要针对阿里云的后台工作人员来做的，用来保证每个实例操作是用户授权的，并且有审计；

其次是 SQL 审计，这个是开放的功能，是保证 PG 里面的每一条 SQL 都是用户已经执行过的；

另一个是内核支持，内核会更多的做一些加密、支持等等；

## 第二大块：容灾容错

异地容灾：异地容灾指的是异地有可用的 PG 或者 IDS 数据库，并且随时随地可以被激活当主库来用，这就叫异地容灾；

异地灾备：异地备份，当异地发生情况的时候，可在异地灾备中把这个数据库从备份机里面拉出来进行一个复活；

AZ 级容灾：AZ 级容灾就是在购买 RDS 时，会选择可用区，这就是 AZ 级容灾。AZ 级容灾更多的是在可用区相当于 IDC 级别容灾，就是当 IDC 单点发生故障的时候，我们可以进行容灾操作。

### 第三大块：基础备份

基础的备份我们有 RPO 和 RTO 的保障。还有 recycle-bin，就是我们所谓的回收站，很多人在 RDS 官网购买了数据库之后随时间退役忘记续费了，这时候他的回收站就可以使用，比如说超过一个月以后，这个实例就被释放掉，但是不会真的实施释放动作，而会放在回收站里进行静默。

### 第四大块：存储安全

存储安全分为几大块，首先是 KMS，这是密钥服务；然后 TDE，透明数据加密；还有 SGX，是我们的硬件加密，都是存储级别的加密。

### 第五大块：链路安全

链路安全更多的是网络链路上的东西，比如说 VPC，VPC 是相互隔离的，还有一个是白名单，还有一个是防暴力破解，SSL 加密等等。

### 第六大块：账号安全

账号加密和账号的安全，账号的安全分为：DB 账户授权、RAM 授权和安全密码字典。

DB 账户授权：普通账户权限管理。

RAM 授权：是控制台的实力权限，因为阿里云提出的是一个主账号下面 N 多个子账号。

安全密码字典：很多客户可能对数据库设了一个很简单的密码，我们会将弱密码阻绝掉，账户也会拦截掉。

基于安全体系建设我们获得了一些认证，PCI-DSS、SOC、MTCST3、等保四级等一系列安全认证。

说完 RDS 整体安全体系建设概览之后，再给大家再讲一下具体是怎么做的？

## RDS 安全体系介绍—数据面—网络控制

首先这个数据面的网络控制，首先左右两边有两个 VPC，一个 VPC 是应用的 VPC，就是你的 APP；还有一块就是数据库的 VPC，master 和 standby 是通过 Streaming 复制的，在 APP 访问数据库中间我们会经过 N 层过滤，首先第一个是 SSL+证书的安全认证，经过之后到 SLB，SLB 又会做黑白名单控制，大家可以在阿里云官网设置，哪些白名单能访问，哪些黑名单能访问，都会在这些里面去做。然后做白名单控制，控制完之后，才能真正访问到数据库。

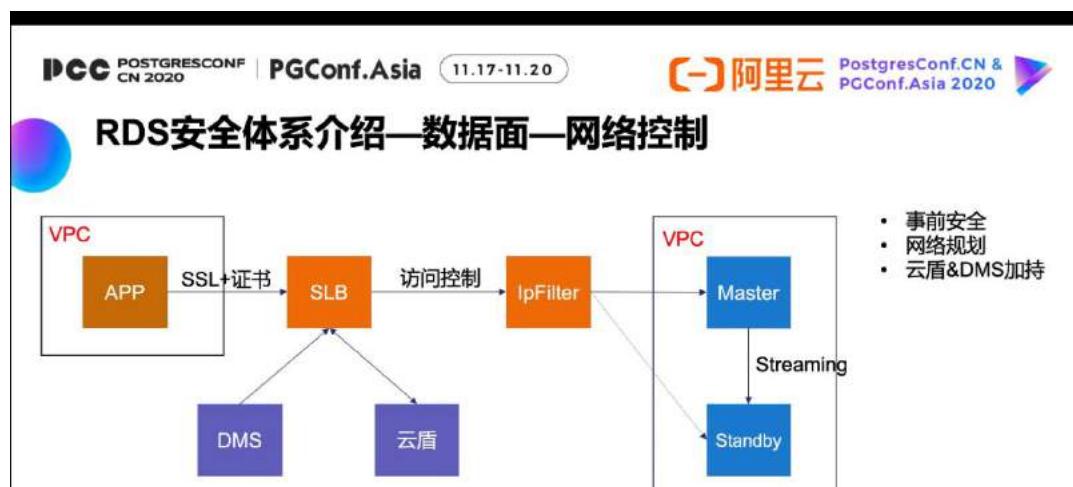
下面还有两个蓝色部分：一个是 DMS，一个是云盾。

这两个主要作用是什么？

DMS 主要解决的问题是客户应用可以访问数据库，但是客户的运维人员和管理人员该怎么访问数据库？

DMS 解决了这个问题，DMS 不但解决了这个问题，还可以做很多的数据库的限制，比如说这个人只能访问这个库这个表，比如说这个列要加密这个列不用加密等等，这些都是 DMS 做的。

还有一块是云盾，RDS 现在是有开公网的，那开公网以后客户的访问就相当于暴露在互联网，有被劫持的风险等等，而云盾都是来做这个事情，就说在安全上，云盾和 DMS 做了数据的加持安全工作。



## RDS 安全体系介绍-数据面-数据控制

安全体系数据面、数据控制这一块，主要分为几大块，首先用户会把密码放在 KMS 服务里面，KMS 把这个密码加密以后传给了 RDS 管控体系，RDS 管控其实根本不知道他的原始密码是多少，然后 RDS 管控把 KMS 的秘钥传给了 RDS 数据库，然后数据库再根据数据读取的情况进行了加密，有两种形式，一种是 TDE 加密，就是分布式存储，更常用的是 TDE 加密；还有一种方案是硬件加密，SGX 加密。

### TDE 加密和 SGX 加密之间的区别是什么？

首先 TDE 加密更多的是数据存储加密，比如说你拿到这个数据文件你再去读的时候是完全读不出来的，因为是加密的；SGX 加密更多做的是数据库读和写的加密，首先写的时候就要加密，然后读出来的时候也是加密的，当你没有秘钥的情况下，每次读出来的数据都是不一样的，这就是 SGX 加密。



## RDS 安全体系介绍-数据面-监控&审计

监控和审计，首先监控目前有十多项监控指标，最长存储 30 天，最小粒度为五秒，这些是数据库 OS 层面以及数据库本身，这些在监控指标后台可以看到。

审计，主要包括 PG 数据库日志审计，并且支持 SQL 审计，就是保证每一条 SQL 都是有记录的；还有一个是支持慢 SQL 审计，用户可以指定慢 SQL 时间，比如说超过多少需要记录；还有支持 HA 日志，比如说数据库为什么进行安全切换了等等，都是可以知道的，都是用户可以通过界面看到的，就是说我们的监控审计都是用户通过控制台可以自主的完成和发现的。

PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20 阿里云 PostgresConf.CN & PGConf.Asia 2020

## RDS安全体系介绍—数据面—监控&审计

- 监控
  - 支持10多项监控指标，最长存储30天，最小粒度为5秒
- 审计
  - PG 数据库日志
  - 支持实例 SQL 审计
  - 支持慢SQL 审计
  - 支持HA 日志

选择时间范围: 2020-11-01 16:46 — 2020-11-01 20:46  
DB: \_\_\_\_\_ User: \_\_\_\_\_ 关键字: \_\_\_\_\_ 更多 关闭SQL审计  
连接数 变更记录 表空间使用情况 SQL 审计 读写分离 实例状态

### RDS 安全体系介绍-实例可用性-HA

安全这个话题必然离不开 HA，实例可用性我们的目标是 99.995%。

RDSHA 有几个优势：

#### 1) HA 自身

首先要有灵活的策略配置。灵活的安全配置就是客户可以自己在控制台设置 HA 策略（比如说检查心跳，心跳超过十秒而且连续三次超过十秒失败，那么我可以发射 HA），这是第一点；第二个是链路的探测能力，很多时候，数据库本身是好的，但是应用访问不了数据库，这多半是链路上出了问题，这是一种情况；还有一种情况是数据库线程池打满，导致外部连接不了数据库，这种情况都属于数据库链路探测；所以说我们的 HA 是站在用户的角度去检查数据库是不是 OK，这就是链路探测能力。

另外是基于 RT 的检测，首先第一点，数据库好不好、数据库运行的健不健康，其实我们不只是要保证数据库是活着，而是数据库的 RT 是多少？

比如说，有个客户每天 0 点-3 点都要跑 DW 任务，这个时候他的 rt 一定很高，平时可能一个 SQL 只要 0.1 秒，如果跑 DW 可能要 4-5 秒，这个时候 HA 认为他这个数据是好还是不好？是要切还是不切？这都是个问题，所以我们引入了 AI 的预测，AI 的预测主要是用来准确的知道这个数据库每天 0-3 点都是在跑数仓任务，那么这个时候就不会去做任何的切换动作，因为他是符合预期的。

## 2 ) 数据一致性

数据的一致性是 HA 的根本，我们在 HA 切之前，会做大量的数据库一致性的校验，保证每一条数据都是从主库传到备库。

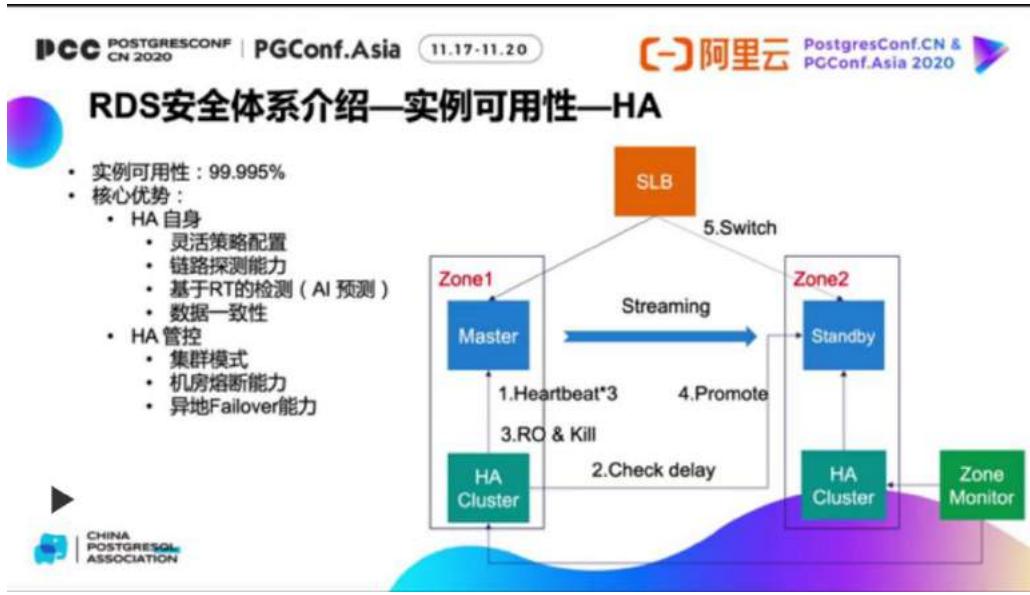
关于 HA 的管控，大家可以看到很多软件，那些 HA 都是不具备集群能力的，都是单点，但是我们的 HA 是 cluster，保证他的可灰度、可验证的，这是集群。

还有一个机房熔断能力，极端情况下，ZONE 1 和 ZONE 2 断网，这时候我的 HA 是切好呢还是不切好？

切了之后，比如说主库是在 ZONE 1，如果把 ZONE 2 的备库激活了好不好？好。

从可运营的角度最好，但是这里还有个问题，我的应用在不断的写 ZONE 1，他不知道 ZONE 2 已经成为一个新的主库了，这时候是会出现双写的情况。所以我们有了一个 zonemonitor 的东西，主要解决的是机房级孤岛和机房级容灾的时候 zonemonitor 可以连接两边的 HA cluster，然后做相应的 action。

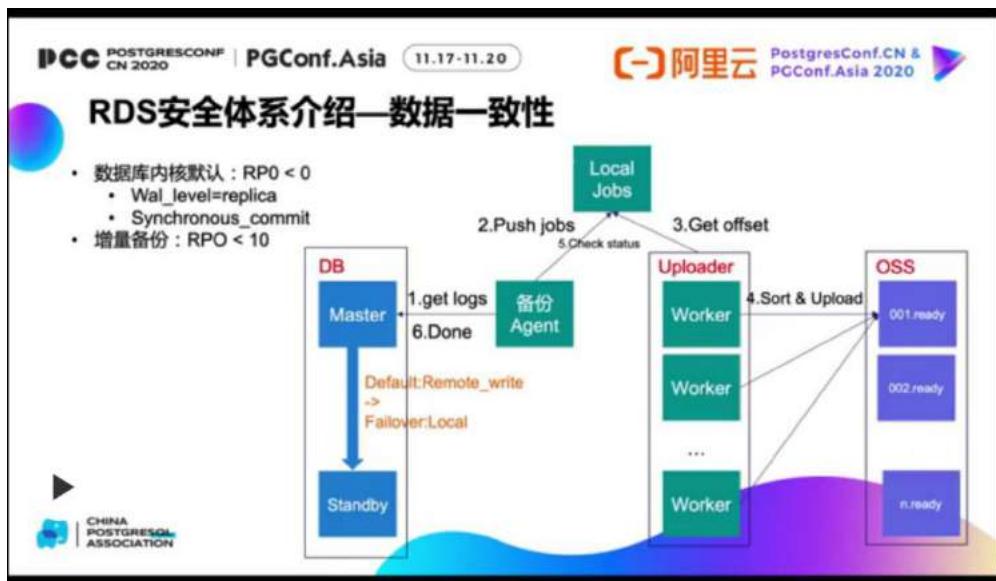
还有一个是异地的 fail over 能力，异地的 fail over 能力更多的是解决 rejoin 级别数据库的可用性问题。



## RDS 安全体系介绍-数据一致性

数据的一致性，一个是数据库内核默认 RPO 小于 0，我们要做到 RPO 小于等于 0。

首先 wal\_level 是 replica 级别，还有 synchronous\_commit 这个我们也做了相应的配置，希望默认是 Remote\_write，只有异常情况下 Local 模式。



数据库自身做了一致性保障，其次是备份如何做数据保障。这里着重讲的是备份的 agent，首先在 DB 里面不断地 get logs，然后不断得到日志，仅仅是得到日志的信息而

不是把日志拿过来，然后我们再把日志的信息放到 Local jobs 里面，放到里面以后会去做 Uploader 任务的拆解与拆分，这更多的相当于是做一个并发，做一个偏移量，然后进行 OSS 上传，这就是备份。备份完以后，会检查状态，最后把 PG 的.wal 文件改成.done 文件。

我们增量的备份要求 RPO 小于 10，这是我们一直在努力追求的方向！

### RDS 安全体系介绍-数据安全性-备份

数据安全性里面的备份包含了大量东西，主要分为几块：全量备份、增量备份、安全性。



第一点是全量备份，我们具备分钟级的备份和恢复能力，由于我们底层使用的是分布式存储，依赖存储的备份能力，我们可以做到分钟级的备份和恢复。

第二点是现在还支持永久备份，比如这个客户实例不需要了，但他需要一个备份，那我们有永久备份的能力；

第三点是库表级备份，库表级备份目前只有 MySQL 上面，PG 上面没有，库表级备份做的更多的事情是，客户想要看 11 月 5 号这个库这个表当时是什么样子，我们现在是支持的；

第四点是增量备份，就是秒级的增量备份；

最后是安全性，很多人问如果数据库备份了，放在备份集上，是不是任何一个人把备份拿走了就可以了，他就可以恢复出数据库，这是不可以的，备份加密我们做了 KMS 和 TDE 的；还有就是异地容灾的备份和备份的有效性验证，备份对有效性验证我们是不断的有定时程序不断地去抽取备份，去做备份有效性验证。

## RDS 体系 VS 用户自建

RDS 体系和用户自建，他们两个区别在哪里？

首先网络上面 RDS 数据支持 SLB/NGLB 多种模式和 VPC 模式，而用户自建一般都使用 DNS 的方式。

还有一种是硬件，我们有专业的硬件团队，每年都会迭代一款硬件，而用户基本上不会去做硬件的更新；OS 我们也有专业的 OS 团队，专业的内核团队去做发布、迭代等等。

数据库我们也有专业的团队做疑难诊断、发布等等，管控我们有运维，备份，告警，HA 等等，我们都有明确的 SLA；而且我们的服务也是 7\*24 专家服务，而用户自建完全是靠 DBA，这是 RDS 体系和用户自建体系的区别。

	RDS数据库	用户自建数据库
网络	SLB/NGLB多种模式，VPC模式	DNS
硬件	专业的硬件团队，每年迭代一款	N/A
OS	专业的内核团队	跟随社区
数据库	专业的内核团队	跟随社区
管控	运维、备份、告警、HA有明确的SLA	跟随社区
服务	7*24专家服务	DBA

## 客户案例介绍

举例子，这是我们某个银行客户，他最后选择了阿里云的 RDS PG 产品，因为这是他对比了其他云厂商的结果。首先阿里云 PG 版本丰富度会高很多，我们支持 PG9/PG10/PG11/PG12 四个大版本，而且我们 Region 开放区有 22 个，与国内云厂商相比是领先的。

还有一个是安全加密，我们有 SSL/KMS/TDE/SGX 加密等多种加密手段；还有一个存储形态，我们现在在阿里云官网可以买到本地盘和云盘两种形式，我们云盘最大支持 32 TB，可以满足很多的业务需求，还有功能丰富度，像刚刚说的监控报警等等功能会比较丰富，所以说这位客户在横向对比自建和其他云厂商以后，最后选择了阿里云 RDS PG。

The screenshot shows a presentation slide with the following details:

**Header:** PCC POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20 阿里云 PostgresConf.CN & PGConf.Asia 2020

**Title:** 客户案例介绍

**Text:** 某银行客户需要购买RDS PostgreSQL产品，横向对比其他云厂商的结果：

**Table:** Comparison table showing metrics for Alibaba Cloud RDS PostgreSQL vs. Other Vendors.

指标	阿里云	其他厂商
版本丰富度	4大版本	3个版本
Region开区	22个	11个
安全加密	SSL/KMS/TDE/SGX	不全
存储形态	本地盘/云盘	本地盘
可扩展性	32TB	2TB
功能丰富度	丰富	一般

**Logos:** CHINA POSTGRESQL ASSOCIATION

## 三、RDS PostgreSQL 未来展望

未来主要通过安全，稳定，经济，智能这四个大的方向来走。



安全前面已经介绍了，稳定上我们要达到 99.995% 的目标，安全和稳定是基础。没有安全和稳定的基础，功能做的再好，再便宜的东西也没有用，这是第一点；

第二点是要做到便宜，就是要做经济，怎么能把数据库做的更经济，这也是我们最大的目标，比如说我们 K8s 化，要做资源磁化等等，就是要做提升性价比的事情；

还有智能，智能是未来的方向，目前 RDS PG 主要做两块，一块是基于 AI 的数据库智能优化；另外基于 AI 数据库智能运维优化，综合来说我们着重的方向还是在安全、稳定、经济、智能这四大块。

# DTS 及其在 PG 数据库生态中的应用

作者 | 王旭 阿里云数据库高级技术专家

## 一、PostgreSQL 数据传输通道关键技术

### (一) 什么是数据传输通道？

数据通道并不是一个很新的概念，很早以前就已经有人或者企业提出了。从数据库的视角来看，数据通道是要去解决各个数据库、数据平台还有信息系统之间的数据连通性的问题，然后来支持数据高速自由的流动。

数据通道建立之后，可以从 TP 数据库流转到 AP 数据库进行分析；或者流转到大数据平台，进行实时计算；或流转到 kv 上去做 cache；或从 AP 数据库再将分析结果流回到 TP 数据库，以便于支撑业务对统计信息的查询。

**在一条数据通道上，流动的数据类型大概有两种：**

第一种，存量数据，存量数据不是实时产生的，比如说一些基础的信息数据。

第二种，增量数据，增量数据是实时产生的数据。

从另一个维度去划分，数据通道上的数据会分成真实的数据和对数据描述的数据。

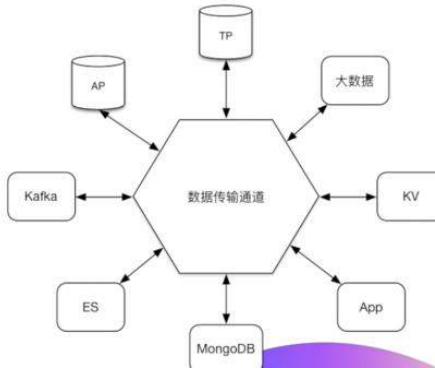
拿数据库来讲，表结构或者说 DDL 信息，以及这张表对应的真实数据，作为数据传输通道，它就像我们的路一样，越快越好，“快”就是数据传输通道的传输效率问题。



## 什么是数据传输通道？

数据传输通道解决各个数据库、数据平台间的连通性问题，是支持数据自由流动的高速公路。

- 存量数据传输
- 增量数据传输
- 数据传输效率
- ETL能力
- 数据质量保障能力



解决了在数据通道上的传输，传输效率之后，我们在数据通道上也应该提供一些 ETL 的能力，ETL 的能力可以做数据的清洗（比如非法的数据，把它清洗成合法的数据，并且写入到这个目标端去）。

或者对一些敏感的数据进行脱敏，再放到大数据的系统里面进行分析。

或者若干张表的数据，在数据通道内部进行多表合并，然后再把它放到 AP 的库里面分析。

做为一条数据通道，在数据质量产生了问题或者数据的链路出现了拥塞的时候，一定要有较强的数据保障能力。

## (二) PostgreSQL 内置数据类型

**PG 内置的数据类型：**

数据通道要解决数据的传输问题，那么就一定要有具体数据。

对于不同的 DB 库来讲会有不同的类型， PG 提供了几种基础类型：数值型、字符型、二进制型、时间型、贝尔型、比特型等。

PG 除此之外还提供了增强的类型，比如地理位置、信息类型、网络型等。

系统 ID 类型——tid， tid 对 PG 来讲，是 PG 的内置字段。在使用上，往往会对 ctid 的错误使用，通过 PK 或 UK 或索引字段，查出来 ctid，紧接着使用 ctid 对表进行操

作。这个使用方式其实是不太靠谱的，因为 ctid 它代表的是这一行数据所属的页以及在页内的偏移。PG 是通过 copy of direct 实现的 macc 机制，在查询出来之后，这行数据的 ctid 可能发生变化，如果我们固化了使用 ctid，就会产生数据不一致的问题。

在大部分的时候，因为我们是通过 PG 的 driver 实现的数据的读取，所以不需要关注各种数据类型内部的存储情况。如果要去看，比如说对 Wal 的日志进行细腻度的分析，或者对一页的数据进行细腻度的分析，可能就会需要了解每一个不同的类型在 PG 内部的存储的格式。



The chart illustrates the built-in data types of PostgreSQL, categorized into several groups:

- 数值型 (Numerical Types):** int2, int4, int8, float4, float8, numeric, interval, time, timetz, date, timestamp, timestampz, bool, bit.
- 时间型 (Temporal Types):** tid, xid, cid, pg\_lsn.
- 布尔型 (Boolean Type):** bool.
- 比特位 (Bit Types):** bit.
- 字符型 (Character Types):** varchar, text, char, bytea.
- 二进制 (Binary Types):** oid.
- 系统ID (System ID):** xid.
- 事务日志号 (Transaction Log Number):** pg\_lsn.
- 地理位置信息 (Geographic Information):** point, box, polygon, circle, line, lseg.
- 网络型 (Network Types):** cidr, mac, macaddr, macaddr8, inet.
- 全文索引 (Full-text Index):** tsquery, tsvector.
- 用户级别事务快照 (User-level Transaction Snapshot):** txid\_snapshot.
- uuid:** uuid.
- json:** json, jsonb.
- xml:**
- enum:**
- range:**
- array:**

Source: <https://www.postgresql.org/docs/11/datatype.html>

### (三) PostgreSQL 分区表

在读取 PG 数据的时候，数据通道里面有两种不同的数据的类型，一种是存量数据，一种是增量数据。

在 PG 的数据库里面，对于存量数据的读取分区表是一个非常独特的 case，分区表从 PG10 开始，有两个截然不同的表现，在 PG10 以前，分区表是基于 PG 的 CHECK-IN HERIT 语法来实现，同时需要手动的在主表上创建 trigger，这种方式实现比较脆弱。

**物理表和逻辑表其实没有明确的区别。**

从 PG10 之后，我们可以看得到 PG 引入了 partition by range 这样的支持，支持分区表的语法，在此之后 PG 的原数据库部分能够明确的区分出来，哪张表是主表，哪张表是物理表。

比如下图的例子，可以看得到 logical table 它的 relation kind 是 p 它代表的是分区表，也就是说主表； physical table 1 它的 relation kind 是 r，代表的是物理表，就是一张正常的、普通的表。

## PostgreSQL分区表

The screenshot shows the PostgreSQL 10 interface with two main sections:

- Left Panel (Code):**
  - CREATE TABLE physical\_table1 (
 CHECK ( logdate >= DATE '2006-02-01'
 AND logdate < DATE '2006-03-01' )
 ) INHERITS (measurement);
  - 创建trigger调用的trigger\_function
  - CREATE TRIGGER \*\*\*
 BEFORE INSERT ON logical\_table
 FOR EACH ROW EXECUTE PROCEDURE
 trigger\_func();
- Right Panel (Table Dump):**

```

<PostgreSQL10>
1. CREATE TABLE logical_table(
    ...
    logdate date not null,
) PARTITION BY RANGE (logdate);

2. CREATE TABLE physical_table1
    PARTITION OF logical_table
    FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');

   relname | relkind
   logical_table | p
   physical_table1 | r
  
```

At the bottom left is the China PostgreSQL Association logo.

### 为什么要区分主表和物理表？

因为在读取全量数据的时候，对于一张分区表，我们希望读取它物理表的数据，这样会读的快，而不去读取主表的数据。

如果不能区分物理表和主表，那就既读取了主表的数据还有物理表的数据，这个数据就会有大量的重复，相当于我们读取了两次。

## (四) PostgreSQL 增量数据获取

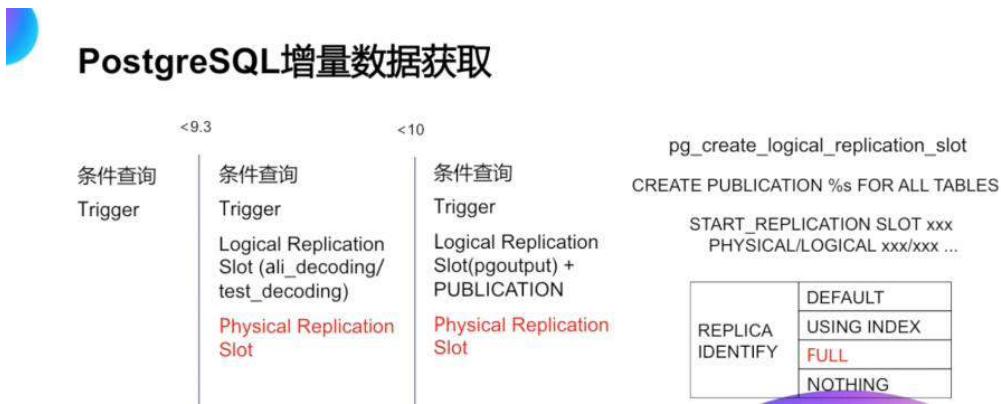
### 怎样获取到增量的数据？

PG 的增量获取方式有三个大的版本：

- 在 PG 的 9.3 之前，它是不支持 Stream Replication 的，一般常用的做法都是基于 trigger 的方式进行增量的获取。当然也可以进行业务的侵入，比如说使用业务的 GMT modify，要求业务有 GMT modify 的字段，通过这个字段，我们可以拿出某一个时间点之后的数据，这个也是一个比较间接的来获取增量数据的方式。
- 从 PG9.3 之后，PG 支持了叫做 stream replication 这样的方式，基于此我们就可以通过这种方式来获取到增量数据。获取增量数据有两种不同的类型，一种叫做

logical slot; 还有一种是 physical slot。在业内目前都是居于 logical slot 的方式进行获取。在 PG9.3 之后，在 PG10 之前，logical slot 的 decoding 是没有一个能够正式应用于生产环境的 decoding 的，所以我们提供出一个叫做 ali decoding 的基于让 logical slot 使用的解码器。通过它我们可以把 PG 的 wal 数据，解码成 string 类型的逻辑数据，再给到增量获取的模块。

3. 在 PG10 之后提供了 PGoutput slot 的解码器，通过它可以实现 ali decoding 相似的功能，在 PG10 之后提供了一个叫做订阅的概念，能够从下图最右边的语句上看出来，在最右边的语句上展示了在 PG10 或 10 以后是怎么样去创建一个逻辑订阅的。



首先会通过叫做 PG\_create\_logical\_reputation\_slot 的方式来创建出逻辑 slot，然后会创建出来一个订阅，这个订阅能够指定订阅哪张表，或是订阅所有的表给逻辑 slot，这里的“%s”是需要填充的上个阶段创建的逻辑 slot 的名称。

对于较少的表，可以指定具体的表名字，如果表很多，就可以写成 FOR ALL TABLES，订阅所有表的增量数据，随后使用 start replication 开启读取订阅数据。

在这里需要注意的是 PG 的 REPLICA IDENTIFY 是表级别的，在这个表级它有不同的级别，比如说 default 是默认的级别； using index 是 PK 或者是 UK； FULL 是所有的； nothing 是什么都没有。

从 PG10 开始支持逻辑订阅，在逻辑订阅中有一个限制，对无主键、无唯一键的表，如果针对这张表开启逻辑订阅，要求这个表的 REPLICA IDENTIFY 是负才可以。否则的话，这张表是不能够进行 delete 和 update 操作的，关于这点要特别注意。

## (五) PostgreSQL Stream Replication

## PG 的 stream, application 内部的实现原理

PG backend 是 PG 的一个服务，接受用户的 TP 数据，形成 WAL 写入到 WAL 日志里面去，并且通知 walsender 读取 wal 的数据，walsender 根据接收到的 signal 的通知之后，读取 wal 的数据。walsender 根据当前 slot 注册情况，如果是 logical\_slot，那就读取一个完整的 Excel record，然后通过 logicaldecode 的方式进行解码，获取到解码后的数据。

如果是 physical\_slot，那就是通过 physical\_decode 进行简单的 xlog 的数据读取，然后获取到待发送的数据之后，最后由 Walsender 将待发送的数据发送给增量的订阅客户端。

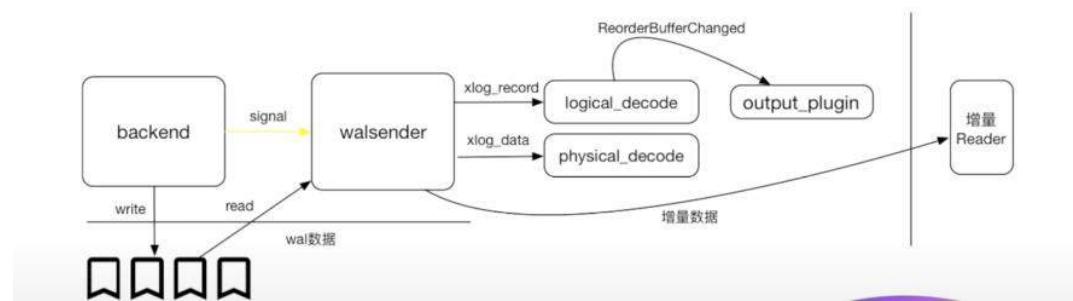
从这儿可以看得到，无论是 logical 的还是 physical 的，其实数据都是 wal 的日志，只不过 logical 的数据它要求是一个完整的 xlog 的记录，而 physical 的只要是一个完整的 xlog 数据块就可以了。

在这里面的关键点，logical slot 它是通过 logical\_decode 这样的一个框架进行解码，框架里面会调用具体的 output\_plugin，进行整行数据的输出与格式的转换。

Logical\_decode 框架，对于每行数据是要求知道这行数据的 relation 信息，要求 PG 知道信息的具体状态。关于这一块如果 logical slot 是以历史的点位进行数据拉取，PG 这边是没办法保证一定能够找得到当时点位所对应的表的结构信息。

所以这就是为什么 PG 的 logical slot 在创建的时候，只能以当前的点位进行创建，而不能以历史的点位。一旦 PG 的 logical slot 创建出来之后，PG 的 backend 会根据各个 logical slot 里历史最早的点位进行数据保存。通过这种方式能够保证所有的 logical slot 在需要 matter 信息的时候，PG 这边都能够得到，都没有被释放掉。

## PostgreSQL Stream Replication



### (六) PostgreSQL 数据导入

如何让链路的效率更快？

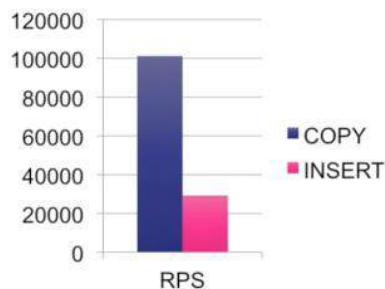
PG 有两种数据的写入方式的：

第一种是 batch Insert，是把一些数据整合成一条 sql，进行插入；

第二种方式就是 PG copy，经过测试，PG copy 的性能要远超于 batch Insert，大概是 batch Insert 的 4 倍左右，是 PG 线路里面一个重要的提升传输效率的一种方式。

## PostgreSQL 数据导入

- Batch Insert
  - `INSERT INTO *** VALUES (..), (..), (...)`
- PG Copy
  - `COPY ... FROM STDIN ....`



## 二、DTS PostgreSQL 数据传输通道实现

理解了 PG 数据库对数据通道能够提供的基础能力后，下面以 DTS 数据传输服务为例来看一下，怎么样基于这些基础能力进行整合，打造出来一条数据通道。

### (一) DTS 是什么——异地多活的数据通道

DTS 是阿里云的服务，中文名称是数据传输服务，DTS 是数据通道的具体的实现。DTS 的一个重要的属性是异地多活的数据通道。

### 什么叫异地多活呢？

比如，解决杭州到北京之间数据库之间的数据传输；多活是杭州和北京之间的两边都支持数据的写入。

DTS 在阿里巴巴集团内部已经是重要的数据数据传输通道，支持了阿里巴巴历年的双 11。

看下图，在 2019 年双 11 的时候，整个的 TP 的数据洪峰大概是 54.4 万米每秒，在 DTS 异地多活的数据通道的强力保障之下，我们做到了毫秒级的延迟，DTS 同时也是阿里云内部、阿里云上客户数据同步的重要的利器。

从下图可以看出，DTS 支撑了全球的数据同步。

## DTS是什么——异地多活的数据通道



## (二) DTS 是什么——用户上云的高速公路

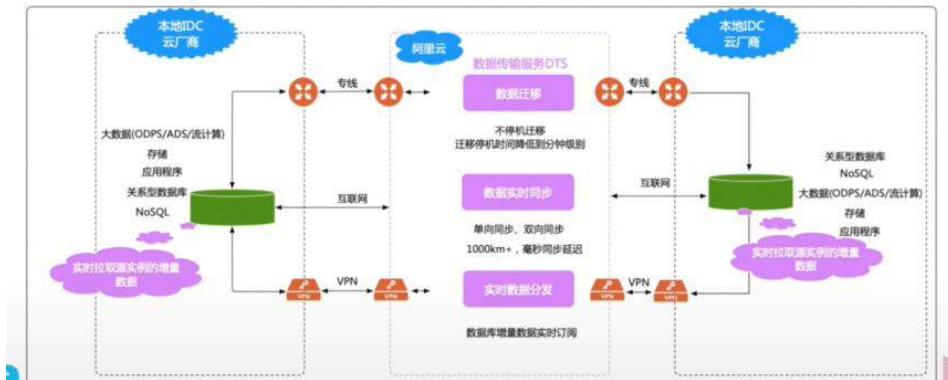
DTS 是云下的用户和其他云厂商的用户上阿里云的高速通道。

云下的用户可以很方便地使用 dts 的服务，将自己云下的数据搬到云上的数据库中，享受不停机数据迁移的能力，并且能够基于 DTS 的双向同步能力，在发现云上数据库出现问题时，能够快速将自己的业务切换到本地。

DTS 支持云下的专线连接，支持公网连接，支持 VPN 连接等多种连接方式。

可以让云下的用户，很方便的享受到云上的数据库能力。比如，云上强大的 IP 数据库的分析能力，大数据的计算能力。

### DTS是什么—用户上云的高速公路

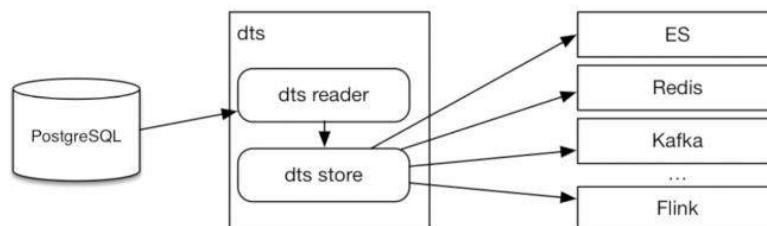


### (三) DTS 是什么——高价值数据的分发源头

DTS 支持各种 TP 数据库的增量数据。

DTS 在获取到这些增量数据之后，可以把这些附加值很高的数据分发到，比如 ES、Redis、Kafka、Flink 等这种大数据平台，进行各种数据计算，将数据的价值最大化。

### DTS是什么—高价值数据的分发源头



### (四) DTS 逻辑架构概览

- DTS 是云上的分布式数据传输通道，在源端的 DTS 支持各种 TP 数据库、Redis、SQL 数据库，以及分布式数据库，在目标端 DTS 可以把这种数据写入到 TP 数据库，mango 或者说大数据平台、Kafka、订阅客户端去。

- DTS 自身给用户提供了两个基础接口，用户可以通过控制台操作 DTS 相应的任务，也可以通过这种 open API 进行批量化的创建与管理。
- DTS 是自身的分成预检查模块、结构迁移模块、全量迁移模块、日志解析模块以及数据写入模块。DTS 通过这个数据校验，提供数据质量的保证能力。
- DTS 的数据都存储在 DStore 中，我们可以通过 DTS 的 ETL 模块，对 DStore 中的数据进行数据清洗、数据转换，并且最终由 writer 同步到目标端，或者由数据订阅的客户端消费。

在原端的 DTS 支持两种数据，一种是存量数据，包括数据还有表结构；另一种是增量日志，这一部分可能包括具体的数据以及 DDL。

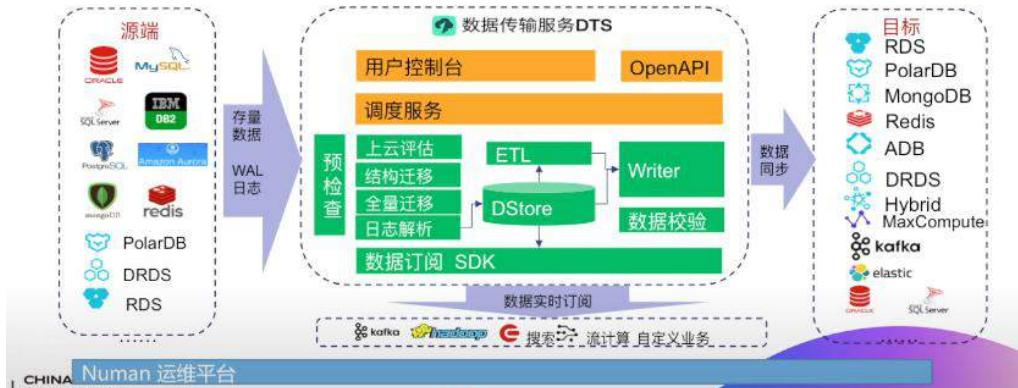
DTS 有三大功能：

- 第一个就是数据迁移，数据迁移这一块主要用来解决用户的数据库迁移；
- 第二个是数据同步，数据同步主要的场景是异地多活的场景；
- 第三个是数据订阅，数据订阅是将增量的数据交给客户端，进行大数据分析，或者用户自定义的业务。

DTS 提供 Numan 运维平台，在运维平台之上可以完成任务的告警监控，可以完成用户的任务管理、任务告警、资源管理以及对源库和目标端的异常情况监控。

DTS 的分布式能力主要体现在它的调度服务。DTS 将资源池化之后，对各个用户的链路提供了 HA 能力，保证了 99.99% 的服务能力。

## DTS 逻辑架构概览



## (五) DTS 高效读写 PostgreSQL

### DTS 如何提升存量数据的传输效率?

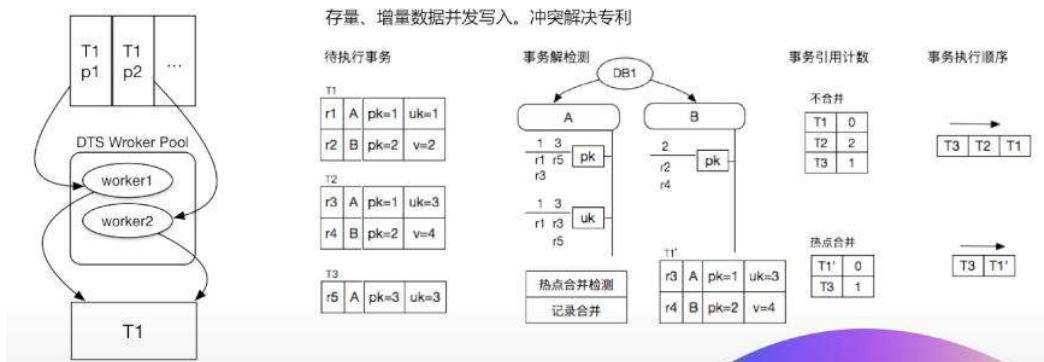
除了使用前面提到的 PG 自身的 PG copy 之外，我们对全量采用了表兼并发的能力。

首先会对一张 PG 表进行切片，将其切分成若干个小片段，针对每个片段使用并发读取、并发写入的方式，将存量数据高效的同步到目标表。

右边是 DTS 针对 PG 增量数据进行的高效写入。对于增量数据，DTS 会将其拆分成原始的一个事物，对于没有冲突的事物，DTS 会采用并发写入的方式。对于有冲突的事物，DTS 会采用串行写的方式。通过这种方式在提升效率的同时，也保证了数据的最终一致性。关于这一块的增量和全量的并发写入 DTS 是有相应的专利的。

还有一部分数据就是针对一张表的某一个字段的频繁的热点的更新，这种情况下因为都是冲突的数据，没办法进行并发写入，DTS 是采用热点合并的方式进行解决。

## DTS高效读写PostgreSQL



## (六) DTS 捕获 PostgreSQL 增量数据

DTS 有着非常丰富的手段获取到 PG 增量数据。

第一个就是 dml trigger，针对 PG 9.3 以前的版本，DTS 可以通过在原库创建 trigger 的方式，拉取到增量数据。针对 9.3 之后的版本，GTS 是通过 logical slot 的方式来获取到 dml 数据。

由于 PG 自身是不支持 DDL 的原始语句写入到 wal 中的，所以 DTS 通过在 PG 源库创建 DDL Trigger 的方式，来捕获到 DDL 的原始数据，再进行同步。

基于 logical slot 的方式有一定的限制，比如说 logical slot 它不能够支持以历史的点位进行数据拉取，logical slot 必须针对无 PK/UK 的表，设置成 IDENTITY FULL，所以现在 DTS 也在公关 physical slot 的增量获取技术，目前已经进入到了测试阶段，在不久的将来能够在 DTS 的云上上线。

## DTS捕获PostgreSQL增量数据



### (七) DTS 解决长链路高 RT 问题

在异地多活的场景里面，要跨 Region 同步数据，必然涉及到高 RT 的问题。DTS 解决问题的基本思想就是近 DB 的部署原则。以增量的数据同步举例，在增量里面，我们的 DTS reader 就是拉取增量数据的模块和源库部署在同一个 Region。

DTS Writer 就是增量数据写入模块和目标库部署在同一个 Region，通过这两个模块近 DB 的部署，最高限度的提升数据的拉取和数据的写入效率。

DTS Reader 是要把拉取到的数据投递到 DTS Store，而 DTS Writer 是要从 DTS Store 里面拉取到同步数据，在这一块 DTS Store 可以部署在源库的位置，也可以部署在目标库的位置，甚至可以部署在中间位置。

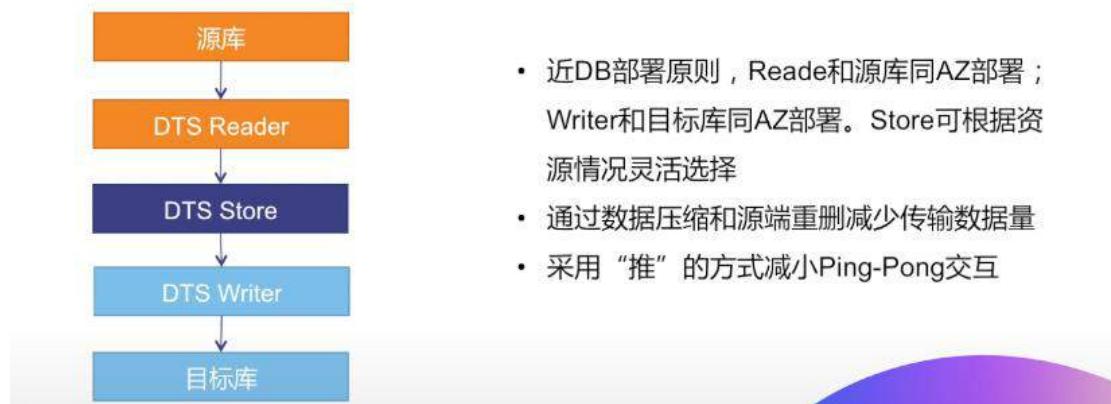
DTS Reader 到 DTS Store，DTS Store 到 DTS writer，数据传输通道是 DTS 自身经过高度优化的，通过高度优化过的一个数据通道，我们就解决了高 RT 的问题。

在这里面的优化点有两个：

第一个，尽可能的减少数据传输量，在这块我们主要是通过压缩，以及源端的数据重删达到的目标。

第二个，因为长链路高 RT 长，但是数据的 Throughpu 是大的，所以我们就尽可能的使用数据推，而不采用 Ping-pong 的方式，以尽可能的降低长链路 RT 的问题。

## DTS解决长链路高RT问题



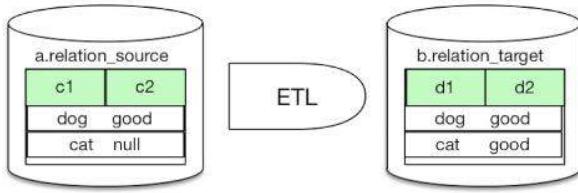
### (八) DTS 提供 ETL 能力

DTS 的数据传输通道里面，它实现了几个 ETL 能力：

第一个就是表级别的，它可以针对库、表、列进行映射。比如，下图例子里面，它是把原库的 Schema 为 a.religion\_source 这个 Relation 映射成 b.relation\_target 这样的一个表。在原表的 c1、c2 列，通过 DTS 可以映射成 d1、d2 列。

第二个维度就是数据维度。举个例子里数据清洗，比如说它对 c2 列 null 字段的清洗成 good。同时 DTS 的 ETL 能力也提供了数据的多表合并。

## DTS提供ETL能力



- Schema、Relation、Column 映射
- 行数据的过滤
- 行数据的改写
- 多表合并

### 三、DTS PostgreSQL 经典案例

在了解了 DTS 基于 PG 的基础能力，所做的数据通道链路之后，基于这个数据通道能够解决哪些问题。

#### (一) 不停机上云

所谓的不停机上云就是源库的业务不需要停，可以很平滑的将源库迁移到云上。

我们有一个比较成功的大型应用案例：

东南亚的一个电商，我们支撑了电商大概 8tb 的核心数据，然后涉及到 11000 个 Oracle 数据库对象的不停机上云。在这个方案里面，我们提供的是双向同步的方案。

先来说正向的方式，首先 DTS 会做结构迁移 1 动作，在这个阶段主要是负责把源库的库、表、列信息以及 PK、UK 信息，迁移到目标库去。在这个阶段完成之后会进行全量数据的迁移，也就是存量数据的迁移。

在存量数据完成之后，我们会进行结构迁移 2 阶段的这样的一个迁移动作。在这个阶段主要是去迁移原库的索引。

**为什么结构迁移要分成两个阶段呢？**

因为经过大量的测试，发现在数据完成之后，创建索引的效率要远高于数据完成之前创

建索引，所以把结构迁移拆成了两个阶段。在数据完成之后，我们使用结构牵引二的方式创建索引。

在结构迁移整个的完成之后，我们就开始进行增量迁移。

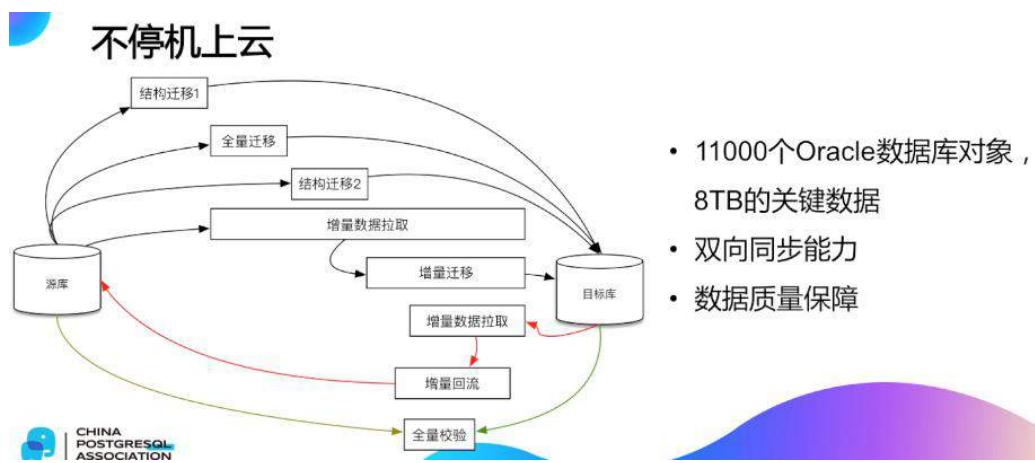
增量迁移的起始位置是从全量迁移之前开始的，这样能够保证整个数据一致性，当增量迁移整个的数据追平之后（就是增量迁移的内容没有延迟），初步的认为这条链路达到了可以切流的阶段。为了进一步的验证数据质量，可以做全链校验的动作。

全链校验是通过拉取源库的数据，以及拉取目标库的数据进行全字段的比较，验证两边的数据是一致的。

当全链校验没有问题之后，就可以进行切流的动作。在切流之前还会搭建出来一条反向链路，所谓的反向链路，我们去启动增量数据的拉取服务，去拉取目标库增量数据，增加增量数据的写入模块，将增量拉取的数据写入到源库。

通过反向链路的建立，如果在切流的过程之中，目标库出现了一些不匹配，或者业务有一些不匹配，我们可以快速的回切到源库。

这个不停机上云的方案，我们在云上已经服务了很多这样的用户了，有着非常广泛的应用。

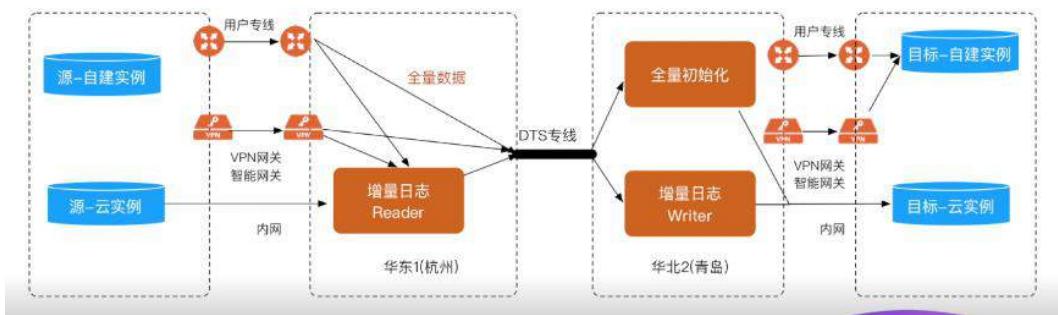


## (二) 异地灾备

来看下图中的例子，这个案例是基于 DTS 数据传入通道来实现的异地灾备。

用户两边都是自建库，但是这两边自建库它是在两地的，通过 DTS 云上的高效数据传输能力，帮助用户搭建了一条从杭州到青岛的灾备链路。

## 异地灾备

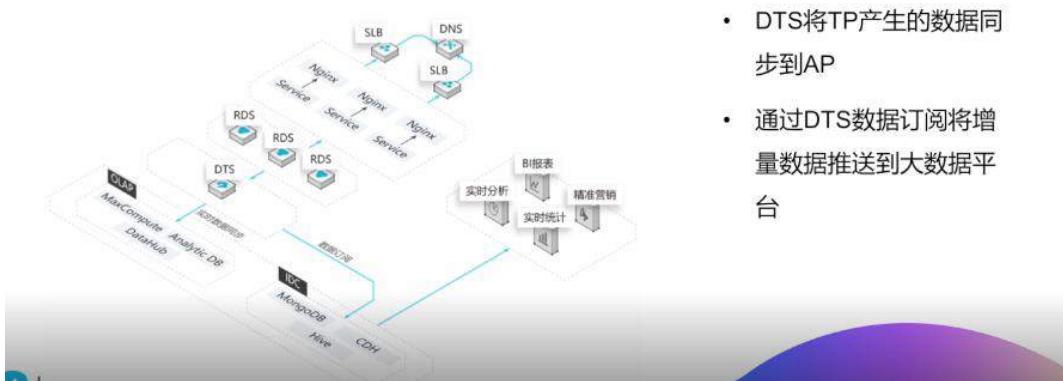


### (三) 数据集成

下图是一个数据集成的例子，DTS 将 TP 数据库，把这样的数据拉取到 DTS 的 DTS Store 里面，通过 DTS 的同步链路，可以写到如 ADB、Kafka 这样的 AP 的数据分析平台里面去。

也支持用户使用数据订阅的客户端，拉取增量的数据出来，按照用户自己的场景进行灵活的应用。

## 数据集成



#### (四) 大数据下游

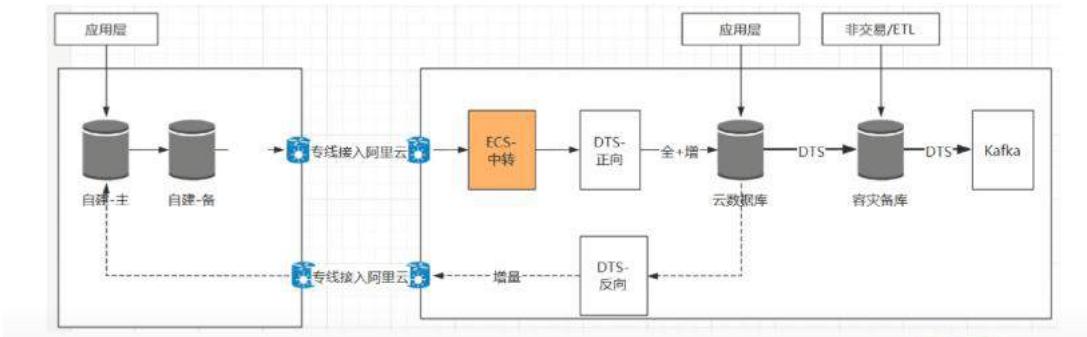
下图是大数据下游的案例，首先用户是在云下，通过专线上云，通过数据同步链路用户在云上的 Ecs 上也有一个数据库，通过专线完成线下到云上的一个同步。

在上云之后，用户在异地又建立了一个灾备库，通过 DTS 完成在灾备库同步。对于灾备库自身，用户为了尽可能的发挥灾备库的价值，又在灾备库上建立了一条订阅的链路。

通过 DTS 把数据同步到 Kafka，然后再由用户大数据的下游，从 Kafka 托举数据进行消费。

同时在云上的库，用户通过 DTS 建立了一条反向链路，通过这条反向链路，再回流到用户云下的数据库里面去。

### 大数据下游



### 四、总结

总结来说，我们以数据传输通道的视角，讲述了 PG 在数据传输通道上的能力，以及以 DTS 为例，说明了基于这些能力，如何构建一条数据传输通道，并且例举了数据上云、数据灾备、数据订阅等经典的使用案例，希望对大家有所帮助。

# PG 数据库生态选型思路与最佳实践

作者 | 樊文凯 阿里云数据库解决方案架构师

## 一、数据库的发展趋势

数据库是 IT 信息技术最基础的一个环节，数据库存在承载了一个企业、一个公司业务系统核心的数据。

### (一) 数据库发展历程

在早期数据库其实仅应用在国防和科学研究院，当时是大型机是在 1950 年开始的，那个时候的数据库仅是一个层次数据库或网状数据库，远远没有形成现在我们这么多的数据库种类；

再往后到 1970 年，随着大型商业处理的需要，处理很多大型商业的一些数据，大型商业处理系统的引入了大型商业处理数据库，这个时候的数据库比较早的有 oracle 和 IBM 的 DB2，是最早的一代关系型数据库，那么这两个关系数据库也一直引用到现在；

到 1990 年的时候，随着 PC 机和 x86 服务器网络的快速发展，数据库也在不断的迭代发展，企业信息化、软件、个人办公、个人娱乐都会使用到数据库，那么这个时候数据库除了原有的关系型数据库库以外，也有了数据仓库用于做历史数据分析，企业数据分析和 PC 单机数据库用来支撑轻量级的应用、轻量级的软件，让个人更好的办公和娱乐；

到 2000 年的随着互联网的发展，媒体搜索、社交以及电子商务娱乐的快速发展，对数据库来说是一个百花齐放的是时代，对数据库的需求也比较大，从关系型数据库到数据仓库，从商业数据库到开源数据库，类似于开源这个时候新兴起来的有 Mysql、PG、Redis、MongoDB 来支撑这些互联网业务和传统企业业务；

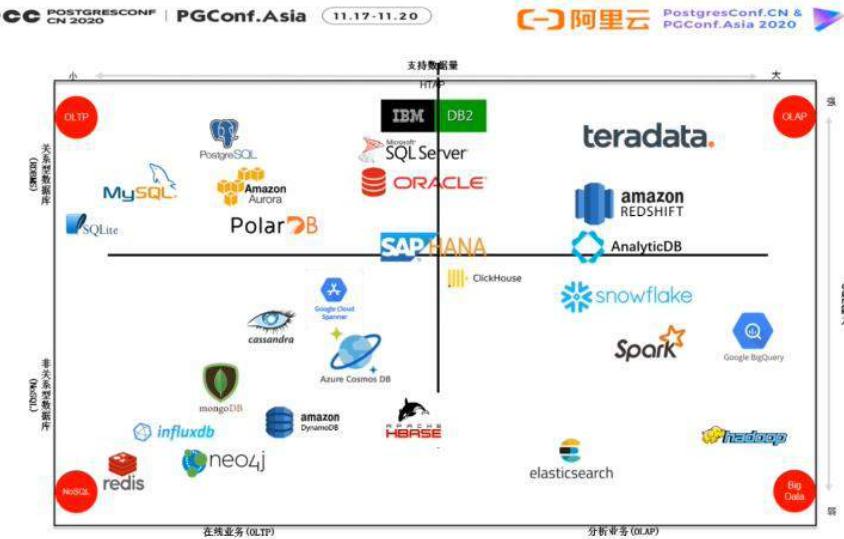
到 2015 年以及到现在，可以称之为是数据库蓬勃发展的一个状态，除了原有的开源数据库、关系型数据库和商业数据库以外，云厂商的发展又涌现出一大批云数据库，比较知名的有类似于阿里云的云原生关系型数据库 PolarDB 及分析型数据库的 ADB、亚马逊的 AmazonRDS、AmazonAurora、AmazonRedshift、微软的 AzureSQL 等等都是比较

流行的数据库，这些数据库和我们各行各业里边所应用到的计算机系统息息相关，我们在研发计算机系统或者在不断的运维发展计算机系统的时候，我们应该怎么样来选择数据库？



## (二) 数据库怎么进行划分

按照数据库的使用用途，把它划分为关系型数据库和非关系型数据库，关系型数据库存放的就是有关系的数据，关系数据库最典型可以把它理解为二维表。非关系数据主要存放一些非结构化的数据，比如打点信息、微博信息和文档信息，这些数据在处理应用业务的时候，有在线事务处理和分析业务处理；在线事务处理主要是处理在线业务的，类似于电商、购票系统，还有一些分析是主要做大数据分析的，比如气象分析系统、股票分析系统，每一种数据库它都可能会有自己的强项和弱项，比如说关系型数据库里边，比较强势的开源数据库有 PG、Mysql，商业型数据库有 Oracle 和 DB2，这两者在分析业务和在线业务里面都比较强势，还有一些其他数据库在选型上来说也有很多，而在真正使用的过程中，我们应该选择哪一个？



### (三) 数据库生产/处理 由量变走向质度 (预判)

对数据库的预判，数据库生产和处理，数据将会由量变走向一个质变的过程，为什么说它是量变走向一个质变？

数据库可以把它理解为是存放数据的一个仓库，这个仓库随着存放的数据越来越多，可能没有办法正常的运载，只能寻求走向质变，来更好的来保存、使用所存储的这些数据，预判到底是什么呢？

预计 2020 年全球数据规模应该会达到 40zb，生产处理要求会智能化，为什么要求智能化？

因为随着现在信息技术的快速的发展，收到的数据，80% 处理的是非结构化占比的一个数据，生产处理更多的要求实时化，现在的数据有在线分析业务系统和离线分析业务系统，在线分析业务系统其实分析的数据量比较少，到 2020 年——2022 年，预计新业务采用实时分析的比例将会达到 55%，原来 t+1、t+2、t+3 或者 t+n 的这种离线的业务分系统，没有办法满足现代商业的一个发展，因为商业瞬息万变。如果能更早拿到这种商业的预判，更可以对为我们的公司或者为我们整个业务的决策提供更有力的帮助。

最后一个数据云化，知名分析机构认为在 2023 年的时候，数据库的云化规模将会达到 75%，那么数据库为什么会云化呢？

主要有两个原因，第一个原因是：云，真的能给我们很好的一个弹性扩展能力；第二个原因是：由传统数据库到云数据库的一个转变，能更好的利用云的一个红利，赋予数据库灵性，让它能更好的来承载现有数据量不断增多，业务不断发展，对数据库要求不断的一个向量变化。



#### (四) 数据库系统架构演进

在早期传统开源也好，商业也好，数据库它是一个单机的数据库，由计算节点加本地磁盘形成了一个数据库业务系统来进行服务，单机它使用到的就是我们的 CPU 和内存加本地磁盘，往后随着单机磁盘容量和单机计算节点的计算资源，没有办法满足现有业务的需要，形成了一个共享的状态，就是共享存储数据库的存储共享的计算节点是多节点，它的优势是易于实现事物的一致性，无需多层复杂管理，那么其优势是怎么实现的？

主要是我们用的是共享存储，我们所并发的事物只要在锁的层次做好之后，那么就可以实现事物的一致性，但是它也有劣势，劣势就是 DB 节点的扩展能力受限，存储扩展能力及 IO 性能可能依赖于更多的高端的共享存储。

简单来说，我们在重组共享状态下的架构，DB 节点要想横向快速扩展的话，它可能会有节点的限制，而这个限制主要在两方面，第一方面是由于使用的是共享存储，节点越多，共享存储的交互就会越频繁，导致成为一个性能的瓶颈；

第二个是我们在共享状态下要保证两个 DB 里边的事物，两个 DB 里各个计算节点的事物和计算节点运行的内容要一致，造成了心跳网络的一个加重。

举个例子，Oracle 数据库最常用的是 Oracle 双节点 RAC 集群，那么如果是 3 个节点、4 个节点、5 个节点、6 个节点，Oracle 是 OK 的，当它发展到 20 个节点，30 个节点，这个时候它的节点间心跳通信就是一个很大的故障点，一般建议 RAC 集群不要超过 4 个节点，超过 4 个节点的话，我们对运维能力、硬件要求就比较高。

第二劣势是要想达到这么大性能的一个要求，可能会对我们的存储的规格要求比较高。你的存储规格越好，你的 IO 和共享能力就越高，这是第二个状态。

到云数据库这个状态，云数据库利用云的一个弹性分布式的架构，实现了数据库分布式的引进，分布式数据库它的优势就是它有横向扩展，这是非常便利的，能形成一个线性的横向扩展。数据库数据是以多副本存储的，无需共享存储，它的优势就是可以不断的扩展，在需要的情况下，每扩展一个，它的 IO、读取，都会呈线性比例的一个增加。

如果是计算存储的能力需要扩展的，那么它的灵活性不会很足，因为这种扩展，需要对应用有侵入，由于其存储是分布式，DB 计算节点也是分布式，要求对数据库分库和分表。在分布式查询的时候，分布式事务处理开销要比集中式的状态大很多，由于数据和计算节点都是分布的，必须要有分布式的处理能力才可以。如果分布式处理能力比较弱，那么这个架构就不适合当前业务应用所需要，只有应用在这个架构上做良好的适配和改造，才可以满足。

这两个架构各有劣势、优劣，在选择的时候可以根据能提供的人力和物力来进行最佳选择。



## (五) PostgreSQL 生态的定位

PG 数据库作为世界上最强大的一个开源数据库，PG 生态的定位是什么？

第一 PG 的核心能力，PG 数据库支持混合负载列式存储，比如说 Greenplum, GPU 并行加速、向量计算，这是一个优势。

最大特点是包容、开放，PG 是支持外部插件的，可以以外部插件的形式开发很多的插件，PG 本身包含很多能力。比如说 PG 对时空、技术、文本是天然支持的，有 postgis 的插件，有其他第三方公司的插件，PG 发展到 11、12 版本时，其对于机器学习、多维计算图谱、向量模拟等，都有很大的能力来做很多的科学计算。

它还有很重大的商用价值。

第一个特点：就是 PG 的 licence，很多的商用数据库很是基于 PG 内核进行开发，比如美国的 EnterpriseDB，中国也有很多基于 PG 内核开发的数据库，由于它的特性，使其成为了传统数据库替换的一个优选目标。

第二个特点：因为 PG 数据库能力非常强，更像商业的一个数据库。比如 Oracle、DB2 在做异构切换的时候，最多推荐的是 PG 分支相关的数据库，因为 My cycle 本身对事物、业务能力要求比较高，PG 由于它的综合能力比较强，针对这些内容，我们是推荐 PG 数据库的。

第三个特点：在云的红利加乘上，它的弹性计算能力是非常强的，比如说在阿里云有 RDSPG, PolarDBPG 都可以借助云的能力来实现快速的 sign-up、sign-out 的扩展。

在社区这块，因为 PG 社区本身是非常强大的，有全球的社区，所以这块有一个非常大的特性——兼容 Oracle，Oracle 数据库是全球第一大商用数据库也是用户最多的数据

库。

近年来 Oracle 数据库的地位一直在不断的受到挑战，最主要的原因就是 Oracle 数据库本身比较贵，也比较成熟，但其实我们并没有真正的用到所有的 Oracle 数据库的 feature。

为了降低成本，同时增加效率，我们会选择其他的数据库进行替代，例如 PG，因为 PG 本身有很多兼容 Oracle 的一些 feature，是可以快速的实现传统数据库和应用到 PG 数据库应用的一个快速切换，就是 PostgreSQL 整个生态的一个定位。



## 二、传统企业级数据库的选型

### (一) 早期传统企业级数据库选型

为什么说是传统企业级数据库的选型呢？

其实在传统企业数据库的选型很早就有定论，在早期传统企业数据库在使用的时候的三大件：Oracle、SQLserver 和 DB2 任选其一保证是没有问题的，那个时候数据库的能力这三家是最大，而且在开发的过程中，我们只要用到了这三大件，基本上不用考虑数据库额外的架构需求，因为这三种商业数据库已经帮我们定义好了，但是它也有一个弱势，就是闭源可扩展性，针对现在的开源数据库和语音数据库来说比较弱，如果传统的企业数据库的 it 信息系统，向新的信息系统的架构演变，会遇到数据库的瓶颈，主要是横向扩展、纵向扩展。

PCC POSTGRESCONF  
CN 2020

| PGConf.Asia

11.17-11.20

阿里云

PostgresConf.CN &  
PGConf.Asia 2020

## 早期传统企业级数据库选型

CHINA  
POSTGRESQL  
ASSOCIATION

## (二) 传统企业客户 IT 现状

怎么样把传统的数据库替换到新的数据库架构？

传统企业的数据库客户 IT 的现状：

传统企业数据库首先它肯定是 IDC 它会有一个成本，这个成本就是我们实际运维的数据库，它都有哪些？我们也做了一个列举，比如说数据库补丁及升级、应用补丁及升级、故障停工性能调整、持续的 it 负担、网络维护与升级等等，都是我们 it 传统企业面临成本。

传统企业的成本总结为三点，第一点就是无法满足现有企业的快速部署，因为传统企业部署就拿数据库来说部署一套数据库，首先要购买硬件、安装操作系统、购买软件、安装数据库操作系统再来部署；

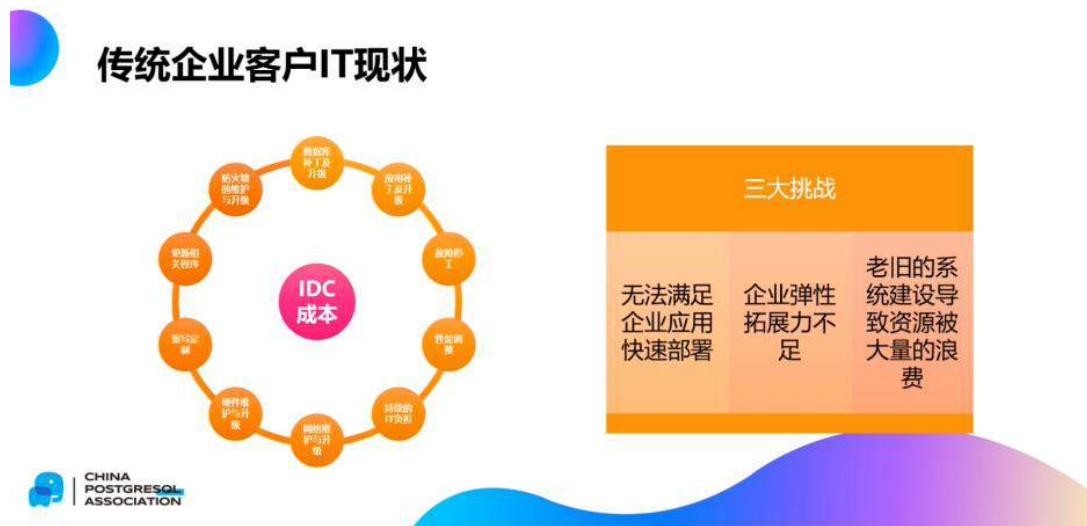
第二点是弹性拓展能力不足，弹性拓展能力不足导致企业每年在做成本规划的时候，必须要考虑到企业最高峰的时候所需要的计算资源。一般来说一家企业它的业务高峰应该是在固定的时间段或在一段时间里边，不会全年都是业务高峰，为了满足这个高峰点，必须要准备高峰值所预估到的硬件，低于硬件的话，是没办法正常运行的；

第三点就是老旧的系统建设导致资源被大量的浪费，很多老旧的系统它依然存在，但是它的存在的已经不是核心系统，却依然要维护，由于维护的成本导致我们现在的基层资源比较浪费，这是整个传统企业 IT 的一个现状。

**PCC POSTGRESCONF | PGConf.Asia** 11.17-11.20

**阿里云**

PostgresConf.CN & PGConf.Asia 2020



### (三) 全球云数据库市场格局

我们可以把它理解为全球数据库市场的格局。在 Gartner 分析象限里面，2019 年国内阿里巴巴营收已经排到了第三位，在亚马逊和微软之后，2019 年的 Gartner 数据库魔力象限里，阿里云也是国内唯一一家数据库公司进入到了挑战者象限。

中国 IDC 的报告中看到，阿里云的市场份额首次超过了 Oracle 市场，排在了中国数据库市场的第一位。当然数据库市场份额是云数据库加传统 IDC 部署的这种模式，如果单只是传统的 IDC 部署的模式的话，还是 oracle 数据库老大，因为毕竟他是 Number one 的商业数据库。从这分析报告里可以看，数据库在不断的往云的方向进行发展，这是一个趋势。



#### (四) 市场洞察与研判: Oracle(One world Conference)

传统的数据库未来会是两个方向，第一个方向，云数据库，比如说云上的各种数据库服务；第二个方向，可以把云部署到我们的机房，就是我们自己做专有云，或者是私有云。私有云和公共云的能力基本是一致的，来为我们提供更好的云数据库服务。

## 市场洞察与研判: Oracle (One World Conference)

## Contrasting Visions of Ideal Database Cloud



## （五）阿里云数据库的特色

云数据库的特色（以阿里云数据库为标准），基本归结为四大特点：

第一点：开箱即用。不需要做任何操作，不需要做任何硬件准备，只要在云服务上进行购买就可以自动使用；

第二点：更快。一般来说，传统的数据库在使用的时候是需要有一个 DB 来进行运维的，来帮我们数据库运行更快。使用云数据库本身云平台就会对内核和参数有很好的优化，来保证我们所购买的数据库是非常快的，同时有云的快速弹性拓展能力；

第三点：更稳。在云上会保证云数据库的可用性。云数据库本身对于硬件来说，有一个稳定的硬件选型和压测，不需要对硬件来进行考虑。

第四点：更安全。传统企业数据库里，我们都要求数据库是要有备份的，没有备份肯定是对的，云数据库是有很多的备份手段和审计手段，以及很多传输的加密手段来帮助我们。就拿阿里云数据库来说，阿里云数据库它有多种类型，包括 PG MySQL 开源类型在内的也有很多商业的、阿里云资源的数据库，比如说 PolarDB、ADB 等等，这些数据库都是可以使用的。

阿里云里有中国最强大的数据库研发团队，提供了丰富的数据库选型，100%进入生态，客户无需更改业务，如果我们是使用 PG 的话，是可以 100%迁移到阿里云上面的，所有的内核都是经过了双 11 高并发和稳定性考验的。双 11 高并发和稳定性考验不只是阿里内部的生态，阿里外部也有很多历经双 11 的企业在使用阿里云的数据库，也是经历了考验的。

**PostgresConf.CN & PGConf.Asia 2020**

**阿里云**

**阿里云数据库**

**10万客户  
40万实例**

- 中国最强的数据库研发团队
- 提供了最丰富的数据库选型，MySQL8.0、SQLServer2019、Redis6.0、Mongo4.2等
- 100%兼容生态，客户业务无需修改，可以直接上云
- 所有的内核都是经过了双十一高并发与稳定性考验

开箱即用	更快	更稳	更安全
<ul style="list-style-type: none"> <li>• 1分钟创建实例</li> <li>• 自动参数调优配置</li> <li>• 内置监控告警</li> <li>• 一键数据迁移</li> </ul>	<ul style="list-style-type: none"> <li>• 内核优化</li> <li>• 参数优化</li> <li>• 性能至少提升30%</li> <li>• 快速弹性扩展</li> </ul>	<ul style="list-style-type: none"> <li>• 可用性SLA：99.99%(业界第一)</li> <li>• 稳定硬件选型与压测</li> <li>• 管理了数十万实例的HA系统</li> <li>• 快速内核bug修复</li> </ul>	<ul style="list-style-type: none"> <li>• 内置备份恢复，避免副本跟丢</li> <li>• 可以恢复到一小时内任意时间点</li> <li>• 回归站，避免误操作</li> <li>• SQL操作审计</li> <li>• 传输与存储加密：TDE, SSL</li> <li>• PCI DSS/ISO/SOC等安全认证</li> </ul>

**CHINA POSTGRESQL ASSOCIATION**

## (六) 传统企业数据库迁移 ( Oracle )

### 数据库迁移为什么比较难?

数据库迁移它不仅仅是把一个数据库里面的数据迁移到另一种数据库里面的数据那么简单，它还要考虑到数据库迁移之后，原有的应用是否能在现有的数据库里更好的来使用。这个问题要考虑到不同数据库之间它可能会所有对象结构不兼容、SQL 不兼容、架构不稳定。

那么怎么样能快速的做到或者是有一个标准化的产品能做到?

阿里云推出了一个数据库迁移的一个方法论和工具集，来帮助我们来快速的进行迁移。

这个方法论就是借助阿里云数据库应用迁移和分析产品，来快速的帮我们做传统企业数据库，类似于 Oracle、DB2 做一个业务调研和可行性评估。业务调研和可行性评估以自动化能力分析我们原有的数据库有哪些特性，在目标数据库里面是否支持，还会给我们一个可迁移的可行性报告，我们根据可行性报告有一个 poc 验证，poc 验证完了之后，就可以做选型决策。

选型决策完成我们适配改造数据迁移、测试迁移、隔接上线，上线之后护航保障方案沉淀，整个的一个标准化的一个体系，来帮助我们快速的把传统数据库到目标数据库做一个选型。

目标数据库可能是 OLTP 场景，也可能是混合场景，也可能是一个 OLAP 场景，我们在阿里云上都对应的数据库，那么针对开源来说，我们有对应的数据库，比如说阿里云上有 RDS-PG，我们可以来进行选择，那么在阿里云自研有 PolarDB PG，也是支持 PG 的，在所有的解决方案里是支持公共云、专有云和混合云的。

这套工具方法论我们和中国信通院联合多家数据库厂商也做了分析，我们联合做出来一个数据库及应用迁移指南，整个操作阿里云也有原厂服务和合作伙伴服务，来支撑我们快速的完成整个的一个沉淀度的迁移。



## (七) 企业级数据库 PG 选型

对比企业级的应用，在 PG 这块有哪些选型？

PG 可以把它简单分成两个分支：

第一个分支是：关系型数据库，就是开源的 PG；

第二个分支是：分析型数据库，分析数据库比较流行的或者是大家比较熟知的 Greenplum 是基于 PG 来进行研发，可以提供大数据并行处理的能力，可以把它构建成一个很好的数据仓库。

还有其他商业数据库，也是支持 PG 的，比如说阿里云的 PolarDB-PG、ADB-PG、RDS-PG 都是原生百分百兼容 PG 的，那么其他还有 EnterpriseDB 或者是其他商业性数据库都是可以的。那么针对 PG 生态，我们在做传统企业数据库选型的时候，完全可以借助 PG 这些分支来替换现有的数据库和现有的技术架构。它首先能帮助我们降低成本，因为 Oracle、DB2、SQLServer 它们的成本相对来说是比较高的，如果我们把一个商业的数据库用开源的数据库替代的话，这个成本来说是巨大的，我们的收获是比较高的。



### (八) ORACLE 数字化迁移方案的“四步法”

整个的选型，我们把它分成“四步法”，以 oracle 为例，更简练来说，就是评估、决策、实施、优化：通过自动化工具来评估，评估完成之后，根据报告来进行决策，决策完成之后，就可以组建团队，组建方案来进行实施了。

整个的 Oracle 的迁移“四步法”，可以把它理解为三件套的工具：ADAM+POLOA RDB+DTS 三件套



目标数据库的选择，就是 POLARDB，POLARDB 是阿里云自主研发的通用关系型数据库，100%兼容 MySQL、PG 数据库更高度兼容 ORACLE 语法，这个是我们数据库的一个能力。它是基于分布式架构的，它和普通 PC 服务器提供商用数据库的能力相当，但是成本只有 10%，具备高性能、高可用性、高可耗，高耐用、高安全、高应用性，这都是 PolarDB 可提供的能力。

**POLARDB侠**

分钟级弹性伸缩的阿里云 POLARDB

5分钟弹性扩容  
存储容量最高可达 100TB

高可用性和高耐久性  
可用性达99.99%  
具有3个数据副本  
实例故障转移小于30S  
具有容错及自我修复能力

高性能和高可扩展性  
3倍于标准MySQL的性能  
2倍于PostgreSQL性能  
最多15个只读节点  
单实例可达100TB

高安全性  
支持VPC网络隔离  
支持TDE&SSL加密  
支持等保、SOX及GDPR  
法规要求  
支持跨机房数据实时同步

高易用性  
数据库智能化运维  
数据库100%兼容MySQL  
和PG，高兼容ORACLE语法  
数据库存储按需收费  
数据库补丁一键升级

## (九) PolarDB 架构的优势

第一大优势：智能代理。能提供自动读写分离的能力，它是一个主节点，多个从节点，它有智能读写自动分离的能力，可以自动化的把写 SQL 路由到主节点上，只读 SQL 路由到只读节点上，不需要我们做任何应用改造，在路由的时候它有一个负载均衡，会平均的把这些读业务分配到各个词组节点上。

第二大优势：计算节点。计算节点是存储与计算分离的，在计算节点有两个保证，第一个是一写多读支持很好的一个线性扩展，这个线性扩展按量按需扩展的，当我们在业务量比较大的时候，我们就可以进行扩展。当我们业务量不是很大的时候，我们就可以缩回来，降低配置，就是按量付费。

怎么样能保证主节点和备节点数据读写是一致的呢？有一个物理日志能实现把主节点更新的数据快速的同步到重节点上面。

第三大优势：存储节点。用的是多副本的分布式存储，来保证整个数据是一致性的。存储和计算节点我们用的是 RDMA 网络，是非常快速的。

## PolarDB 架构

### 智能代理

- 读写分离：智能分析SQL，并根据SQL自动做读写分离
- 负载均衡：自动在多节点做负载均衡并支持自定义业务负载

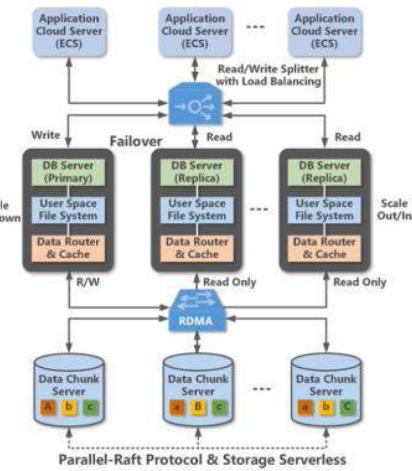
### 计算节点

- 计算与存储分离：一写多读，支持线性扩展，最大可扩展至15个只读节点
- 物理日志复制：主备数据复制延迟<1秒，数据分析及时有效
- 并行查询：充分利用多核CPU性能，加速业务数据处理
- 高可用检测：自动判断节点可用性，主备切换小于30S，会话自动转移，确保数据一致性

### 存储节点

- 分布式多副本：单可用区3副本，双可用区6副本并通过分布式算法ParallelRaft 实现多副本强一致，确保RPO=0
- 软硬一体高性能：网络、Bypass Kernel的用户态协议协同确保数据库25Gb RDMA高性能
- 秒级备份不加锁：利用自研Polar Store存储引擎实现秒级存储数据备份

阿里云



## (十) PolarDB Box 一体机

快速的把 PolarDB Box 一体机带到我们的机房里面，有三大优势：

第一大优势是：兼容。高度兼容 Oracle 语法，降低我们高达 95% 迁移人力，1000+ 项兼容性涵盖特色，我们有一个数据库的开发手册；

第二大优势是：易用。就相当于你把阿里云的 PolarDB 搬到了我们的机房，实现了开箱即用，在阿里云上怎么操作 PolarDB，在机房就怎么操作 PolarDB；

第三大优势是：创新。在 PolarDB 上增加了 Ganos 插件，它可以支持我们 5G+IoT 全新 SQL 查询，专用于数据定位，能实现人、时、地、物、事件综合的一个处理。在 10 亿级数据查询，查询“十万位点+事件”信息的时候实现达到秒回。这个能力已经运用到很多企业地图，比如说像恒力中恒、高德都在使用。

- 兼容：高度兼容 Oracle 语法：降低高达 95% 迁移人力
  - 1000+ 项兼容性涵盖 PL/SQL、函数
  - 配合 ADAM 实现 DB、应用、压测、上线全链路可回归迁移
  - 主攻 88C/710G/40TB 以下 OLTP 场景：
    - INSERT、SELECT(OLTP 查询) 性能与 Oracle 持平
    - DELETE、UPDATE 性能高 3 倍
- 易用：将云原生数据库带回家：阿里巴巴数据库管理最佳实践
  - 开箱即用、易运维、灵活扩展
  - 与阿里云公有云相同的控制台及 OpenAPI 操作体验
  - 自带 PolarDB 数据库，无需单独购买软件 License
- 创新：5G+IoT 全新 SQL 查询：特有 Ganos 时空 SQL 能力
  - 专用数据类型及函数，人、时、地、物、事件综合处理
  - 十亿级数据中查询十万“位点+事件”毫秒返回

### (十一) PolarDB-PG 提供完整的生态工具

- 第一个生态工具是 DMS 可以支持数据开发、数据管理、数据安全；
- 第二个生态工具是 cloudDBA 能实现自动化的监控运维；
- 第三个生态工具是 DTS 能快速的做数据全量迁移；
- 第四个生态工具是 DBS 是做数据备份的；
- 第五个生态工具 DAS 和 DMS 是数据安全的考虑。在 dms 上也有一个数据审核、数据脱敏、SQL 注入检查的一个能力；
- 第六个生态工具 DAS 数据自治管理，可以自动优化 SQL，自动 SQL 限流，根据数据库的复杂来弹性伸缩。这是整个 PG 生态。

## 三、最佳实践

### (一) ORACLE 架构迁移标准化解决方案

我们把传统的数据库以 Oracle 为代表分为两类，前一类是已有业务，对事物是强制性的，存在很多表的撞演和复杂的 SQL，这个时候迁移，如果我们使用的 oracle 特性比较少的话，只是用了关系型数据库特性，推荐直接签到 PT 上面；

如果我们用到的 oracle 特性比较多，可以选择阿里云的 PolarDB PG、PolarDB Oracle 兼容版来实现快速的迁移，因为它本身对 oracle 做了很多的兼容，PolarDB 还有一个特色，就是它的单库支持 100TB。

如果超过 100TB 的话，我们有两个解决方案：

第一个解决方案是可以把这些历史数据放到外部表里，外部表存放的位置 OSS 存储，这个是非常廉价的；

第二个解决方案是可以专门构建 ADB for PostgreSQL 的数据仓库，这个数据仓库可以快速的实现实时数据处理。它可以直接实时的把 PolarDB PG 或者 PostgreSQL 里的数据实时同步到 ADB 里边，如果我们要做业务分析的话，实时传回来。

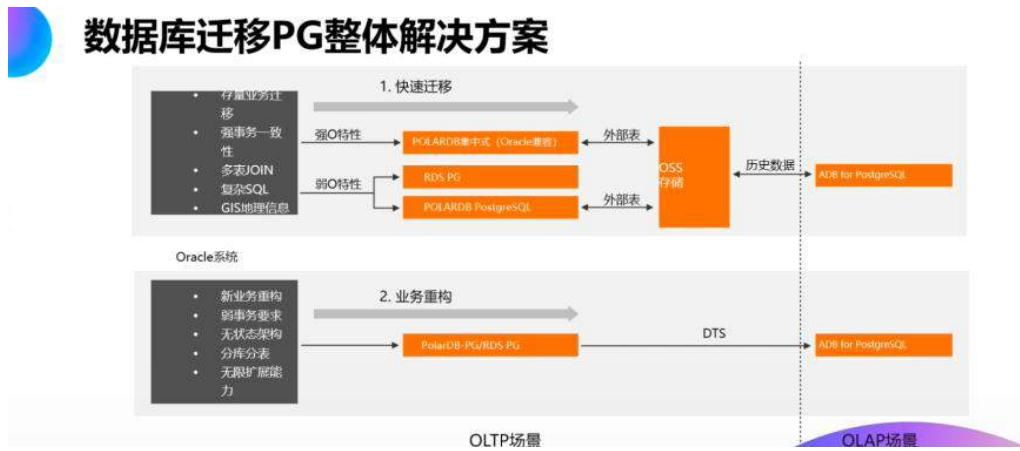
当然了我们也可以选择新的分布式架构，这个业务可能是我们新业务要重构，或者是它的事物比较弱，没有状态的架构也需要分布，可以选择阿里云的 RDS PG 加分布式服务来实现一个全分布式的架构。

这是我们提供的 Oracle 架构前移的一个标准化的解决方案。



## (二) 数据库迁移至 PG 整体解决方案

混合性的场景，就是存量业务迁移强事务一致，有多表 JOIN，有 GIS 地理信息，还有一些分析型的内容。这里还是以 oracle 为例，如果我们用到的 oracle 特性比较强，我们叫强 o 特性，我们可以选快速迁移的话，可以直接迁移到 POLARDB ( Oracle 兼容板 ) 上面。



如果是弱 O 特性的话，对性能要求不是很高的时候，我们可以迁移到 RDS PG 上面，但是如果我们对性能要求比较高，需要有很强的动态扩展能力的话，我们建议是迁移到 POLARDB PostgreSQL 数据库上面。如果数据量比较巨大的话，用外部表或者是直接构建一个数据仓库。

针对 OLTP 场景，如果这个业务是新业务重构，事物要求不是很强，是弱事物的，而且很多表它是无状态的架构，也可以很好的支持分库分表，但是它要求有无限的扩展能力，那么这个时候建议迁移到 PolarDB-PG 和 RDS PG，但是 PG 可能根据我们的业务量发展，我们有分表需求的话，我们也可以在上面加分布式数据库的服务。

最后由于它的分布式业务，做 OLAP 是不行的。如果我们需要做 OLAP 类的业务场景，建议单独的构建数据仓库，来为我们提供实时的数据分析。

很多业界科学家认为未来是从 BIGDATA 转向 FASTDATA，未来大数据和现在大数据是有一些不一样的。现在的数据虽然也是大数据，但是它在分析数据时可能存在一个时间差，而这个时间差会对我们的业务产生一点点影响；比如说如果是一个实时分析我们可以马上定位、决策，一个公司的商业能力，要是有时间差存在，可能会产生影响。FASTDATA 就是实时传输、实时分析给予我们一个实施回馈，只要分析模型 OK，就能做出一个正确的选择。这就是我们对未来大数据的研判。

# PostgreSQL CDC 的最佳实践

作者 | 王建 阿里云数据库高级运维工程师

## 一、CDC 应用场景

### CDC 的概念？

CDC 是 Change Data Capture 的缩写，是去捕获一个数据的变化情况，刚才提到主题是 PostgreSQL CDC 的一个最佳实践，其实也就是说 PG 数据库的变化事件的捕捉，理解为数据库内容中产生的一系列行为信息，比如说 insert 或者 delete 影响一些数据，它把这些变化的数据去捕获出来。

### 捕获出来做什么？

#### 第一个场景：不同数据源间的同步

左右两边是一个对比，左边的可能是一个购物网站，这个购物网站它可能有几个功能。

- 一个就是说我们日常的库存的扣减、库存的存储，我们会存储在传统的一个数据库中。
- 另一个是有一些用户经常频繁去点击访问的一些页面，放在数据库中的话，很难扛住压力，所以一般会把这部分数据放在缓存中。
- 第三部分因为是一个电商网站，所以说它肯定会有去搜索商品的一个功能。

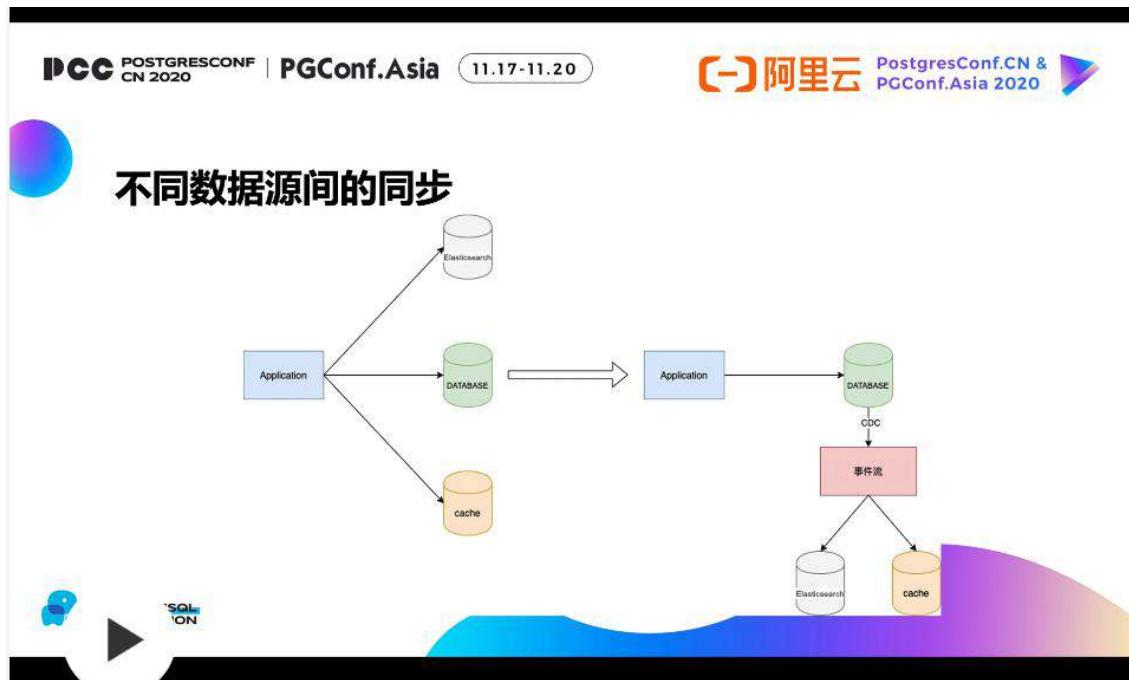
如果我们要用传统的数据库去做搜索，是很难的，它只能支持一个前缀模糊匹配，所以对于一些前后都模糊的一些情况，它是做不到快速响应查询的，那么这种场景下我们一般都会用到一些搜索引擎，比如 ES，我们会把商品的一些信息存在 ES 里面，然后在 ES 中进行一个快速的匹配和查找，这是我们一个传统的架构。

### 这个架构到底有哪些痛点？

数据是需要去维护多份的，比如说需要去维护一份缓存的，维护一份数据库的，还要去再维护一个搜索引擎的，这样其实对于应用来说，就会有一个很大的改造难度和一个工作量。

如果说我们要用到一个刚才说到的数据库变化事件的捕获的一个功能的话，那就会变成一个什么样的效果？

比如说右边这个图，数据首先是写入你的 PG 的数据库，然后通过变化事件的一个捕获变更，就把数据库内变化的一些情况解码成一个事件流，然后事件流的再去你的搜索引擎和缓存中进行一个消费，这样其实很大程度减轻了我们应用需要写多份数据的情况，可以减少开发的进度和维护的成本，这是第一个场景。



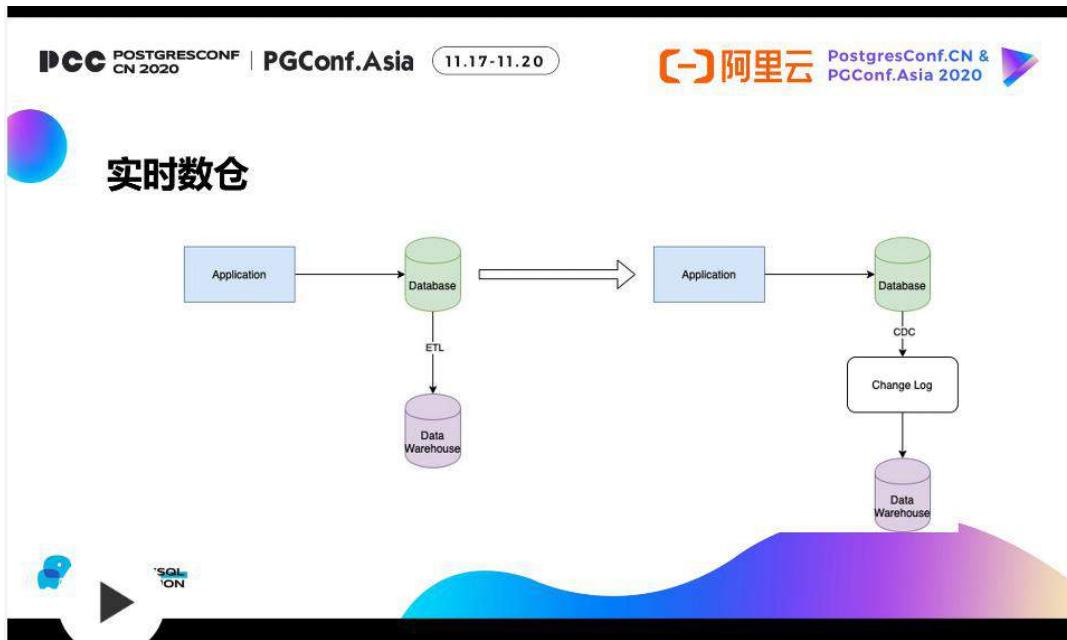
## 第二个场景：海量数据存储对于传统数仓情况

现在是一个大数据的时代，有很多海量的数据是需要去存储的，但是对于传统的一个数仓是一个什么样的情况？

一般都是来说是将前一天的数据，通过一些 ETL 的程序跑出程序，然后再同步到数仓中，这样带来一个很大的问题，就是时效性，另外还有就是我们需要去定制化的开发一些 ETL 程序的工作。

如果用到 CDC，就会方便和快捷很多，比如说先写入数据库，就是传统数据库中，然后再通过数据库捕获这些变化的事件，最后再到我们的数仓中进行一个消费。

这样一是增强了时效性，另外也简化了中间做 ETL 动作，这是第二个场景。



### 第三个场景：微服务间的数据同步

从传统的一个集中式的服务，慢慢向一个微服务的方向去演进，为什么？

还是刚才购物商城的例子，原本可能商品用户订单大家都放在一个数据库上跑着，这个时候会有哪些问题？

- 第一个，如果说当用户量上涨很大的情况下，就会导致性能会出现瓶颈，并发不足，比如说我表太大了，然后索引也很大，就导致性能跟不上。因为垂直向上的扩容能力是有限的，毕竟物理服务器的一个 CPU 数量，还有内存的大小是有上限的，所以说会达到上限的天花板，导致你没法再向上扩展，这是第一点。
- 第二点是在应用开发的一个过程中，也会有很多问题，比如你在商品应用做了一个发布，这个时候大家应用都在一起，此时其实需要你所有的都要去进行一个升级。每做一个发布，所有应用都要去做升级，商品发布对于用户来说，其实是没有强相关的，照理来说它不用去单独升级。

- 第三点开发的效率也会降低，因为耦合很严重的情况下，有可能大家做的内容是有交叉的，所以说这部分也是需要单独的人去维护的。

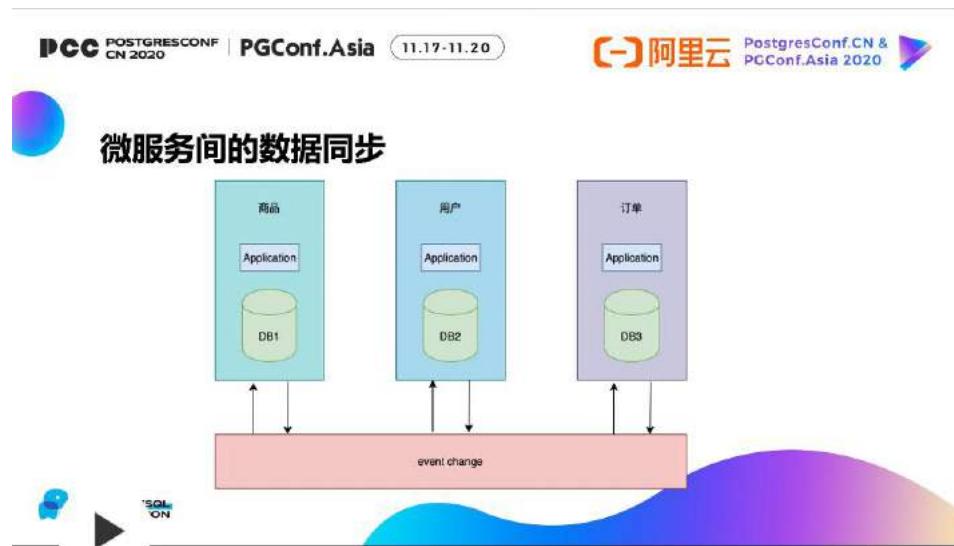
所以说开始逐渐向微服务演变的情况，就是我们去按照一些模块去拆分。

其好处是，一个是可以支持更大的用户量和访问，另一个好处是每个团队其实只需要负责自己这个模块，中间就不会有一个互相耦合的情况。

但是做了一个拆分之后，又会出现一个新的问题：

第一个是因为变成了一个分布式服务的情况，这个时候会带来运维成本，第二点是在传统数据库中，对于两个表之间的访问，在同一库中是可以用 join 来做到的。

比如说一个用户他想查一个商品的明细，如果都在一起，就用 ID 对这两个表进行一个关联，就可以得到一个数据，但是对于非服务人员，因为数据在不同的一个库中，是没法做同库之间的效应的，那么就要通过我们 CDC 能力做一个数据同步，这样的话用户能快速的访问到我们商品的一个明细，这就是微服务间的数据同步。

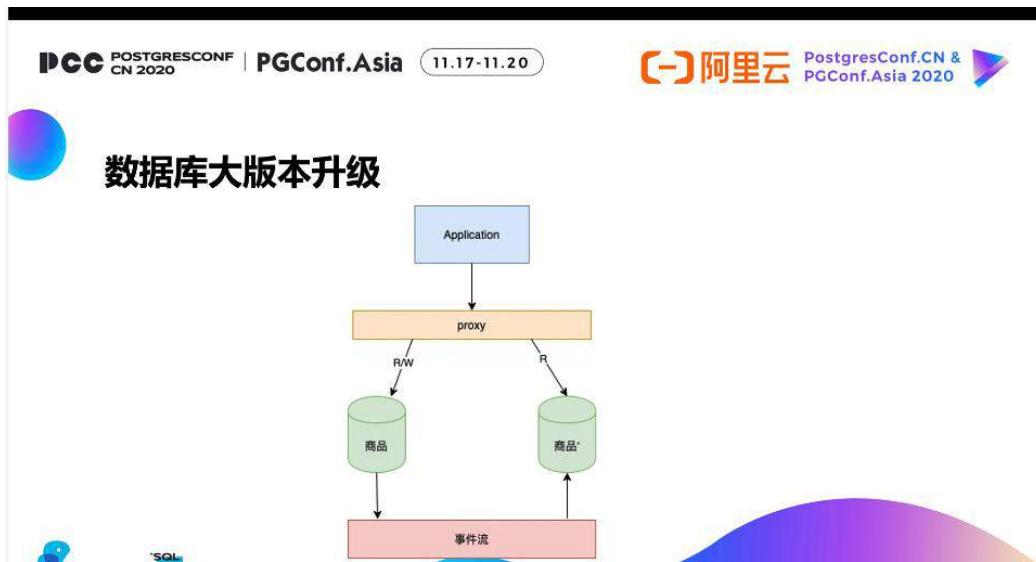


#### 第四个场景：数据库大版本升级

数据库是需要一个一直能在线提供服务的能力，如果说要尽量缩短它不可用的时间，就需要我们做到一个无缝切换的能力，需要有实时同步能力。

实时同步怎么做，其实跟之前一样的道理，就是相当于我们实时去捕获一个数据的变更，然后同步到新的版本的数据库上，然后等数据达到一致的时候再进行一个切留。

描述的过程很简单，但其实中间有很多很复杂需要去注意的关键步骤，但是当你有了CDC的能力，是能帮助你去做无缝升级，数据库大版本无缝升级的一个利器。



## 二、CDC 的原理

PG 的 CDC 是怎么提供的？出现原理？

这个其实类似于一个事物的日志，因为数据库它是一个需要保证数据是不会丢的，不可能说写条数据进来，如果数据库挂了，数据就会丢，这就是不符合数据库的一个能力的。所以说第一点是要保证一个数据写入一定是不会丢的。

第一部分，看一个具体的 case，例如插一条数据，数据库到底发生了什么？首先一个是 shared buffer pool，shared buffer pool 是数据库的一个共享缓冲区，相当于我们会分配一块很大的内存，这部分用于缓冲数据，可以极大增强我们访问数据的速度。

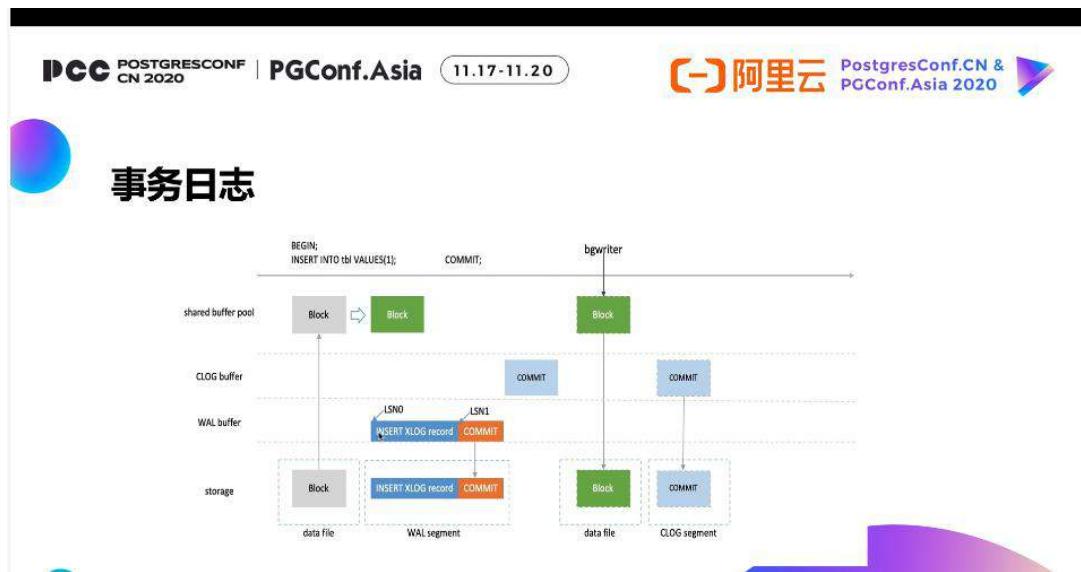
第二部分是 Clog buffer，指的是记录一些事务日志提交的状态信息，比如说一号事务是 commit 或者二号事务是 Block，记录这些的信息，Clog buffer 其实就是记录暂时存储 Clog 的一个地方。

还有一个 Wal buffer。Wal 就是刚才提到的事务日志临时放在 Wal buffer 中，然后在一定的条件下会进行刷盘的 1 个动作，会把 Wal buffer 中的数据写入到一个存储中，存储一般就是指传统硬盘数据文件。刚才说到的一个例子，首先是从一个数据块中读一个空号，然后会在内存中将数据写入数据块中。

这时候这个数据块是发生过变化的，这个时候先写一条 Insert 的日志，写到日志的 buffer 里面，当触发 commit 的时候，它就会把刚才写的日志落盘。

这样如果说数据库发生了崩溃的话，我们是可以通过日志来找到这条数据的。

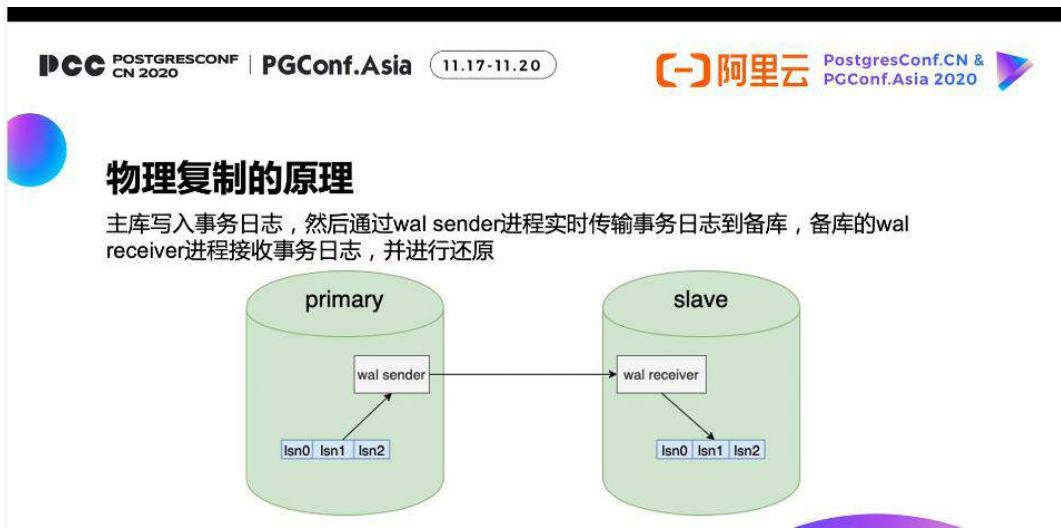
那么数据怎么去落盘？数据化会通过一个 BG writerde 进程，或者说是通过 check PAD 这种进程，定时将脏页刷写到磁盘中，这就是一个事务日志的功能和原理。



## 物理复制的原理

当我们有了日志，它不仅能产生一个备库去读取变化的日志进行一个重放，可以做到高可用的能力。

它的原理其实也很简单，比如说我们有一个主库，主库它会实时产生刚才提到的事务日志，然后事务日志通过 wal sender 的进程传输给 wal receiver，备库拿到日志，然后再进行还原当初一个最新的数据状态，这样就做到了主备同步的能力。



## 物理复制的局限

日志其实记录的是：比如说一个事物日志的头，事务日志标识的是往表中插入一条数据，然后这个是实际变化数据块的头，然后是实际写入的一条数据块，它其实有很多个这样的收支来组成一个 wal 的文件。



默认 wal 文件是十六兆，它记录的内容是实际的数据块内的变化信息，因为它记录的是数据库块内变化信息，这就会导致它复制的不灵活。当然物理复制也有自己的好处，它不需要等待一个事物提交，可以实时将这些数据块的变化做一个同步，对于 DDL 或者大事物这种它都不会遇到复制延迟的问题。

当然它要依赖我们数据库的版本一致，操作系统一致，这样才能保证数据块变化的情况是一致的。

## 逻辑复制

逻辑复制和物理复制的具体区别在哪？

下面是一个例子，比如说这一块是我们有三张表 tableA、tableB、tableC，然后对这些表做一些变更，会产生一个刚才提到的事务日志，事物日志再通过 output plugin，output plugin 相当于对事务日志做一定的解码。

比如说刚才看到的是物理块变化的情形，其实就很多 0101，但是我们就要 0101 按照一定的方式进行解码，比如默认解码成 text coding，可以按照一个 Json 的方式解码，解码之后会通过 publication（就是发布端），比如说我们只发布 a 表和 b 表，然后订阅端订阅发布，这个时候就会把订阅到 a 表和 b 表的一个数据变化同步到目标端的一个数据库的表中。

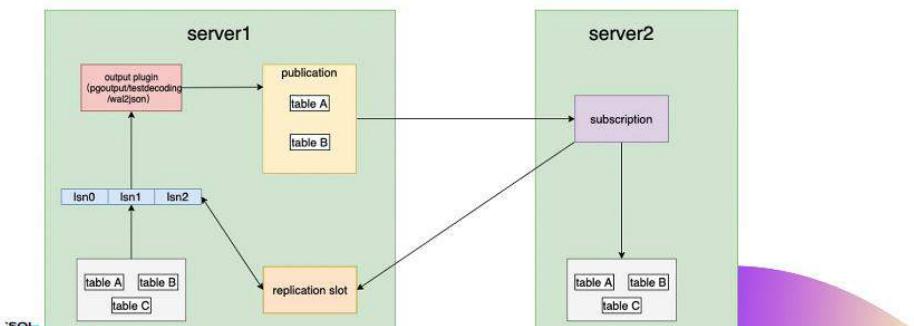
然后还有一个功能是 replication slot，就是复制槽的一个能力，复制槽是做什么的，比如下图这几个箭头代表了什么？

其实就是当订阅的时候，消费的这段数据，我们会实时通知复制槽，复制槽的话来控制事务日志是否被清理和一个位点的推进。

**PostgresConf.CN 2020 | PGConf.Asia 11.17-11.20 | 阿里云 PostgresConf.CN & PGConf.Asia 2020**

## 逻辑复制

通过解码插件对事务日志进行解码转换为sql语句，通过发布端发送到订阅端进行消费



再举个例子，如果不通知的情况下会怎么办？

将会造成日志累积堆压，就是导致你的 wal 日志可能会把你的磁盘写满。

如果说不通过复制槽来保证数据不被清理掉的话，会发生一个什么样的现象？

比如说我日志写得很快，因为 wal 日志是会循环使用的，这个时候我订阅端消费的比较慢，就导致日志没有被消费，但是日志会复用掉，就会导致这个数据丢掉，所以在此串联了一个这样的问题。

所以 replication slot 就是要解决我们事务日志被清理掉的问题。

## 逻辑解码

刚才讲到逻辑复制的原理，那么它到底是怎么去做解码的？

比如说我们创建一个复制槽，刚才说了我们使用逻辑复制一定要有逻辑复制槽，否则逻辑变化数据可能会丢。

The screenshot shows a PostgreSQL terminal window with the following content:

```
postgres=# SELECT pg_create_logical_replication_slot('logical_slot', 'test_decoding');
pg_create_logical_replication_slot
-----+
(1 row)

postgres=# insert into tes select 1;
INSERT 0 1
postgres=# SELECT * FROM pg.logical_slot_peek_changes('logical_slot', NULL, NULL);
 lsn      | xid      |
-----+
 2/FBB631A0 | BEGIN 1747612
 2/FBB631A0 | 1747612 | table public.tes: INSERT: id[integer]:1
 2/FBB632B0 | 1747612 | COMMIT 1747612
(3 rows)
```

Below the terminal window, there is a decorative footer with the PostgresConf.CN & PGConf.Asia 2020 logo.

刚才知道那个插件的一个名称，比如用了 test decoding，这时候我们去对表插入一行数据，通过命令来捕获复制槽内的数据变化情况，就看到一个 begin 和一串数号，然后

对这个表做一个插入操作，然后是 ID 是 integer 类型，值是一，再做一个 commit，就发现 insert 语句被解码出来了，原本直接看 wal 日志的信息时，是根本看不懂的，不知道他其实是对这个表做了一个插入，插入 ID 等于 1 的一个情况。

### 逻辑解码还有哪些问题？

比如 DDL 为什么不支持 replication slot，我们通过一个例子给大家讲清楚，它的原理到底是什么，为什么它会不支持 DDL？

The screenshot shows a PostgreSQL terminal session with the following commands and output:

```

test=# SELECT pg_create_logical_replication_slot('logical_slot', 'test_decoding');
pg_create_logical_replication_slot
-----+
(1 row)

test=# alter table test add column id2 int;
ALTER TABLE
test=# SELECT * FROM pg_logical_slot_peek_changes('logical_slot', NULL, NULL);
 lsn | xid | data
-----+
 2/FB8828F8 | 1747617 | BEGIN 1747617
 2/FB8828F0 | 1747617 | COMMIT 1747617
(2 rows)

```

At the bottom left is the logo of the China PostgreSQL Association.

比如说有专业的复制槽 test decoding，我们对表做一个 out of table 的动作，这个时候我们捕捉复制槽的变化，但是发现他其实只有一个 begin 和 commit，它是没有 DDL 语句的解码的，所以说它不支持 DDL 语句解码，但是 DDL 像这种同步的能不能做？也能做，后面会提到这种 DDL 同步是怎么做的。

我们看一个例子，先对这个表对于表先做一个 replica identity default，这个是做什么呢？是说我们对这个表设置一个默认复制，它默认的话就是走主键，就是说它会记录主键的一个前镜像。

## 逻辑解码

逻辑解码无法或者无主键表的前镜像

```
test=# alter table test1 replica identity default ;
ALTER TABLE
test=# delete from test1 where id=1;
DELETE 1
test=# select * from pg_logical_slot_get_changes('logical_slot',NULL,NULL);
 lsn   | xid    |          data
-----+-----+-----
 2/FBB9C6E8 | 1747633 | BEGIN 1747633
 2/FBB9C850 | 1747633 | COMMIT 1747633
 2/FBB9C888 | 1747634 | BEGIN 1747634
 2/FBB9C888 | 1747634 | table public.test1: DELETE: (no-tuple-data)
 2/FBB9C8F0 | 1747634 | COMMIT 1747634
(5 rows)
```

这个例子中，`delete from test1 where id=1`，比如说我们把 ID=1 这行数据要删除掉，ID=1 其实我们删除后之前的前镜项。那么就捕获一下变更。

我们发现它删除了以后，是一个 no tuple date，因为没有数据，所以不知道删的是哪一行。因为我们用的是一个无主键表，命令说的是 default，这样它是没法去捕获到这种无主键表的一个前镜像的，所以说会出现这样的情况，这个时候我们下游拿到了 delete，这个语句它其实没法做删除的，因为他不知道删的是 ID=1 的值。

所以说这个我们为什么逻辑复制的过程中一定要设置主键，就是这个原因。

再看设置主键后的一个例子，比如说我们对这个表加一个主键 ID，然后这个时候我们去删 ID 等于 2，我们就可以看到它这个时候就把具体的一个 ID 类型 L 值列了出来。

## 逻辑解码

设置表的主键后，逻辑解码可以看到数据主键的前镜像

```
test=# alter table test1 drop constraint test1_pkey;
ALTER TABLE
test=# alter table test1 add primary key (id);
ALTER TABLE
test=# delete from test1 where id=2;
DELETE 1
test=# select * from pg_logical_slot_get_changes('logical_slot',NULL,NULL);
 lsn   | xid    |          data
-----+-----+-----
 2/F8B9C928 | 1747635 | BEGIN 1747635
 2/F8BABA80 | 1747635 | COMMIT 1747635
 2/F8BAC088 | 1747636 | BEGIN 1747636
 2/F8B8B288 | 1747636 | COMMIT 1747636
 2/F8B8B248 | 1747637 | BEGIN 1747637
 2/F8BC2980 | 1747637 | COMMIT 1747637
 2/F8BC29E8 | 1747638 | BEGIN 1747638
 2/F8BC2AE8 | 1747638 | table public.test1: DELETE: id[integer]:2
 2/F8BC2AE8 | 1747638 | COMMIT 1747638
(9 rows)
```

如果设置了主键的话，我们就可以看到数据主键的前进项，这样就可以帮助下游做一些 delete 或者是 update 的动作，update 也是同样的一个道理。

设置 replica identify full 是指我们会保存整个事物日志的一个前进项，全字段的前镜像。

**PGConf.CN 2020**

**PGConf.Asia (11.17-11.20)**

**阿里云 PostgresConf.CN & PGConf.Asia 2020**



## 逻辑解码

设置replica identity full，可以看到删除行的全字段前镜像

```
test=# alter table test1 replica identity full ;
ALTER TABLE
test=# delete from test1 where id=3;
DELETE 1
test=# select * from pg_logical_slot_get_changes('logical_slot',NULL,NULL);
 lsn   | xid   |          data
-----+-----+-----
 2/FBBC2C08 | 1747639 | BEGIN 1747639
 2/FBBC45E0 | 1747639 | COMMIT 1747639
 2/FBBC4618 | 1747640 | BEGIN 1747640
 2/FBBC4618 | 1747640 | table public.test1: DELETE: id[integer]:3 id1[integer]:3
 2/FBBC4718 | 1747640 | COMMIT 1747640
(5 rows)
```

我们看看这个例子，就是我们删个 ID 等于 3，这个时候的话会出现看下面这个值它会出现一个 Delete: ID[ingeter]:3，然后 id1[ingeter]:3，这样就会记录一个删除这一行的前进项。当然也会带来一个问题，为什么不默认都设置这个就好了？因为设置这个的话，它相当于你的事物日志中会记录所有的一个前镜像，就导致你的事物日志会变得很大。

所以说这是我们强调要作一个主建，用主键的话首先性能高，因为能复制。

如果没有主键就设置 replica identify full，带来的问题就是数值变大，然后原本的吞吐会下降，另外一点就是逻辑复制的速率也会降低。

还有一点，对于序列的话，我们的逻辑解码也是无法支持的。

比如说我们创建一个表，这个表是 serial 类型，就是一个智增序列的意思，这个 serial 它还有一个 int 型，它会自动默认帮你把这个列变成一个自增列，然后并且还会自动创建一个序列。



## 逻辑解码

对于序列，逻辑解码也是无法获取序列的变化情况

```
test=# create table test2(id serial,id1 int);
CREATE TABLE
test=# select * from pg_logical_slot_get_changes('logical_slot',NULL,NULL);
 lsn   | xid   |      data
-----+-----+
2/FBBC4868 | 1747641 | BEGIN 1747641
2/FBBD7A70 | 1747641 | COMMIT 1747641
(2 rows)

test=# insert into test2(id1) select 1;
INSERT 0 1
test=# select * from pg_logical_slot_get_changes('logical_slot',NULL,NULL);
 lsn   | xid   |      data
-----+-----+
2/FBBD7A8 | 1747642 | BEGIN 1747642
2/FBBD7B10 | 1747642 | table public.test2: INSERT: id[integer]:1 id1[integer]:1
2/FBBD7B80 | 1747642 | COMMIT 1747642
(3 rows)
```



这个变化的情况，其实这个是本身没有的，插一条数据看一下变化情况，发现是一条 insert，对这个表有具体变化的一个值，这是符合预期的。

但是它少了什么、在哪？你不知道。

因为我们没有指定具体的已知序列，因为它是自增的，它 ID 是会加一的。但是我们在事物日志中是没有看到一个序列变化的情况的，所以说对于逻辑解码，我们也无法获取一个具体的序列变化。

这些，会让我们很清楚的明白逻辑复制的原理是什么，逻辑复制有哪些局限，为什么它不支持 DDL，为什么不支持序列，为什么我们一定要有主键，其实就是这些原因。

## 三、CDC 的主流工具

第一个，就是 PG 自带的一个逻辑复制，这个能力是 10.0 版本之后才提供的能力。

比如说我们先创建一个普通的表，然后创建一个发布端发布这个表，然后对这个表插一下数据，在订阅端再创建一个相同的表，做一个订阅，订阅的连接串指向我们发布端的地址，发布端名字是什么。

创建完订阅端之后，再去查这个表，就发现数据有了，这样就很方便快捷，就达到了我们需要的数据捕获的能力，这个时候我们实时往里插，它也会一直同步，这就做到了表值的同步能力。

```

发布端
test=# create table t1(id int primary key,name int);
CREATE TABLE
test=# create publication pub1 for table t1;
CREATE PUBLICATION
test=# insert into t1 select 1,1;
INSERT 0 1
test=#

```

```

订阅端
test=# create table t1(id int primary key,name int);
CREATE TABLE
test=# create subscription sub1 connection 'host=127.0.0.1 port=5434 dbname=test user=postgres password=123456' publication pub1;
NOTICE:  created replication slot "sub1" on publisher
CREATE SUBSCRIPTION
test=# select * from t1;
 id | name
----+---
   1 | 

```

它有哪些优点？

- 第一点：方便易用；
- 第二点：具备一个物理复制不具备的能力，支持跨一个数据库的大版本；
- 第三点：实时同步。

它的缺点是什么？

- 第一点：不支持异构数据源
- 第二点：冲突导致订阅中断
- 第三点：DDL 维护需要提前创建，比如说我们刚才如果要对第一个表加一个字段，如果加上字段之后，我们再去插三个字段，这个时候订阅端同步过来是有三个字段的，订阅端的提议表它是写不进去的，之后发生冲突的话，就会导致整个复制后面之后数据都是没法同步的。所以我们需要提前对 t1 表去加一个字段，比如加上刚才说到新的字段，然后再对发布端加字段，需要注意顺序的。
- 第四点：必须是 PG 的一个 10.0 及以上的版本，因为 10.0 版本才默认带了发布和订阅的能力，所以说的试用场景是一个数据库的大版本升级，就很便捷的表决同步，微服务间的数据同步就很方便。

**PostgreSQL 逻辑复制**

**优点**

- 方便易用
- 支持跨版本
- 实时同步

**缺点**

- 不支持异构数据源
- 冲突导致订阅中断
- DDL维护需要提前创建
- 需要PostgreSQL 10.0及以上版本

适用场景：数据库大版本升级，微服务间的数据同步

## pglogical

上面说到 PostgreSQL 逻辑复制只有 10.0 及以后的版本才有内置的逻辑复制能力，之前怎么办？

9.3 版本之前是没有逻辑复制能力的，之前都是通过吹杆触发器的方式来做。这种文化对原端有很大的一个性能影响，因为要做触发器，然后每变化一次就要触发一次，就会造成源端性能的大幅下降。

但有了逻辑复制的话，就完全就解决了这样的问题，因为是实时解码的日志，其实对源库基本上是不会有影响的。

所以 pglogical 就是最早去做逻辑复制的能力，它是 9.4 的 PG 版本，支持了一个逻辑复制的接口，pglogical 就是基于这些逻辑复制定义的一些接口，使我们拿到这些数据，然后再组装成 SQL 到下游去消费（从订阅端解析日志，然后再去下游消费），是做这样一个过程。

但如果你有一些开发的能力的话，也可以自己做解码解析，再做订阅。

还有一些像 outjson 这种，其实他相当于已经帮你做了很多事情，它把发布端解码的能力已经做了，比如说拿到日志，就已经是一个 Json 的格式了，然后再通过 Json 进行转码转成 SQL。还有一些创意，比如 default buffer，它可以直接帮你解码成一个 SQL。



再看一下 pglogical 的使用方式，它也是创建一个发布端的 note，这时我们选择一个发布的副本级，其实设置的所有表都会发布到默认 default 副本集中，然后我们对 text logical 去查一查数据，订阅端也是同样的创建一个节点，设置为 noad name，是一个订阅端的节点，再创建订阅端，这时订阅地址就是发布端的地址，然后我们这配置了一个 structure: =true，这就相当于我们会同步表结构。

可以注意一下之前在 pg 自带逻辑复制中的话，其实我们是提前创建了这个表结构，然后在这里是完全没有创建表结构的，我们直接就查这个表，会发现数据已经过来了，而且表结构也同步好了，所以说它有这样的一个能力。

## Pglogical 额外的功能

### 1. 支持 DDL 多个节点同时执行

刚才提到了 pglogical 逻辑复制，它其实对于 ddl 是很不友好的，因为我们要提前在源端做，但是它有一个能力叫 replicate ddl command，它可以同时支持 DDL 在多个节点同时去执行，这就免去了我们需要重复劳动的成本，只需要去执行一下命令，然后带上 DDL 语句就好，也不会引起复制的中断的问题。

### 2. 序列值的同步

原生的逻辑复制是不会去捕获事件变化的情况的，但是它可以做到序列式同步，就是你可以在你的源端状捕获到你的序列 synchronize，然后再将它更新为目标端的一个最新值就好了。

### 3. 支持行条件过滤

有些数据是需要被过滤掉的，而这种同步也是能做到的，所以很灵活。

The screenshot shows a slide from a conference presentation. At the top, there are logos for PostgresConf.CN & PGConf.Asia 2020, with the date 11.17-11.20. Below the title 'pglogical' is a bulleted list of features:

- 额外的功能
- pglogical.replicate\_ddl\_command 支持DDL多个节点同时执行
- pglogical.synchronize\_sequence 序列值的同步
- 支持行条件过滤
- 支持4种冲突策略，保留本地数据，应用源端数据，保留最新版本，保留最老版
- 支持PostgreSQL 9.4版本及以上
- 支持指定延迟复制
- 支持结构初始化迁移

### 4. 支持四种冲突策略

- 1) 保留本地数据
- 2) 应用源端数据
- 3) 保留最新版本
- 4) 保留最老版本

### 5. 支持 PostgreSQL9.4 版本及以上

支持 9.4 版本及以上，比如说 9.4 版本、9.5 版本的都可以通过 pglogical 的能力进行大版本的升级，这样就很大降低了平均迁移的时间。

### 6. 支持指定延迟复制

因为 PG 的物理复制是支持这个能力的，但是逻辑复制是不支持的，所以说 pglogical 是支持延迟复制。比如说有延迟备库的情况，这样就方便了在一些误操作时，导致我们数据删除，也可以通过延迟备库去来找回来。

## 7. 支持结构初始化迁移

优点：

- 支持冲突检测，健壮性更好
- 支持行过滤，灵活性更高
- 维护 DDL 方便

缺点：

- 不支持异构数据源
- 需要主键，不支持 replica identity full
- 不支持不同字符集间的数据库逻辑复制

**pglogical**

优点

- 支持冲突检测，健壮性更好
- 支持行过滤，灵活性更高
- 维护 DDL 方便

缺点

- 不支持异构数据源
- 需要主键，不支持 replica identity full
- 不支持不同字符集间的数据库逻辑复制

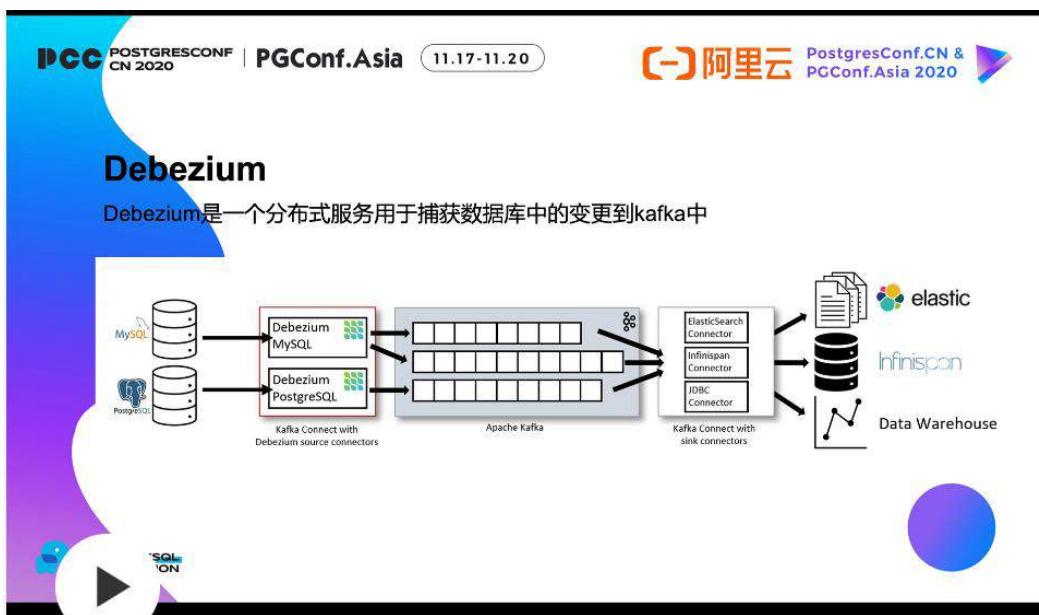
适用场景：数据库大版本升级、微服务间的数据同步

## Debezium

Debezium 其实是一个分布式服务，是捕获一个数据库变更到 kafka 中的，它这个是怎么做的？

它源端是支持多种数据源的，比如 MySQL 是解析它的并 log，得到具体的 SQL 变化情况，写入到 kafka 中，然后 PG 解析 wal 日志，解析到变化的情况后写入到 kafka，然后如何消费呢？

其实 kafka 的话，它有一些 JDBC connector 或者 ES connector 或 Infinispan connector，就可以直接写入到你的目标库中。



它的优点是什么？

优点：

- 支持异构数据源
- 分布式服务健壮性高
- 直接写入到 kafka 中，生态好

缺点：

- 依赖 kafka，需要熟悉 kafka
- 无界面化，配置项复杂
- 组件多，运维成本高

适用场景：

不同数据源间的同步、微服务间的数据同步、实时数仓。

**Debezium**

优点

- 支持异构数据源
- 分布式服务健壮性高
- 直接写入到kafka中，生态好

缺点

- 依赖kafka，需要熟悉kafka
- 无界面化，配置项复杂
- 组件多，运维成本高

适用场景：不同数据源间的同步、微服务间的数据同步、实时数仓

## DTS

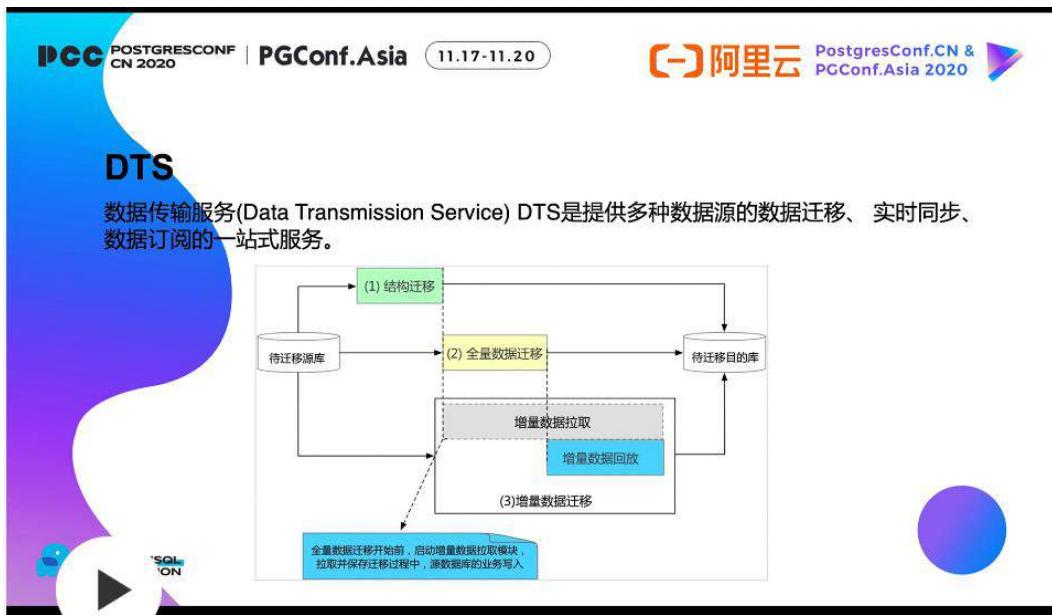
DTS 是数据传输服务的能力，它提供多种数据源的数据迁移、实时同步、数据订阅的一站式服务。

它的原理分为三步：

第一步：结构迁移；

第二步：数据全量迁移，数据全量迁移的时候，会启动一个增量数据的拉取模块，然后解析，当全量迁移完成之后，再去做一个增量数据的回放。回放到目标库后就完成了数据的实时迁移；

第三步：实时同步。



## DTS

优点：

- 支持 DDL 同步
- 支持行过滤，灵活性更高
- 支持 MySQL、PolarDB、ADB 等目标端迁移
- 界面化、操作简单

缺点：

- 不支持 kafka 生态
- 操控感较差，用户使用相对不透明

适用场景：

数据库大版本升级、微服务间的数据同步、实时数仓。

The slide is titled "DTS" in large bold letters. It features a blue-to-purple gradient background with abstract shapes. At the top left is the "PCC POSTGRESCONF CN 2020" logo. To the right is the "PGConf.Asia" logo with the date "11.17-11.20". On the right side is the "阿里云" logo with "PostgresConf.CN & PGConf.Asia 2020". Below the title, there are two sections: "优点" (Advantages) and "缺点" (Disadvantages), each with a bulleted list. A note at the bottom states "适用场景：数据库大版本升级、微服务间的数据同步、实时数仓". The footer contains a "SQL ON" logo.

**DTS**

优点

- 支持DDL同步
- 支持行过滤，灵活性更高
- 支持MySQL、PolarDB、ADB等目标端
- 界面化，操作简单

缺点

- 不支持kafka
- 用户使用相对不透明

适用场景：数据库大版本升级、微服务间的数据同步、实时数仓

SQL ON

这 4 个主流的工具，对于它们的一些适用场景、使用方式、优缺点，已经有了一个大致的对比，以后可能需要真正的去要落地用到我们刚才说的那些能力，比如说大版本升级，或者说实时数仓、用户通过，我们就可以考虑这些 CDC 的一些工具一些方案，来支撑我们的一个业务的发展和能力。



微信关注公众号：阿里巴巴数据库技术  
第一时间，获取更多技术干货



阿里云开发者“藏经阁”  
海量免费电子书下载