# OceanBase1.0 分布式技术架构

2017-11-29 日照 技术琐话

OceanBase作为金融级分布式数据库一直备受瞩目，特刊OB团队中坚日照同学的架构解析一文以飨读者。

---

**OceanBase 1.0项目从2013年初开始做总体设计，2014年开始编码、测试，2015年底正式上线并无缝迁移部分集团MySQL业务，直到2016年中才正式上线蚂蚁核心业务，包括会员视图、花呗、账务，等等，最后“丝般柔顺”地通过了2016年双十一大考。**

从技术架构的角度看，一个分布式数据库主要就是两个部分：一个部分是怎么做存储，怎么做事务；另外一个部分是怎么做查询。首先我们看第一个部分，主要是三个关键点：可扩展、高可用以及低成本，它们代表了OceanBase的核心技术优势。

分布式存储&事务

第一我们怎么理解数据，如何把数据划分开来从而分布到多台服务器？这个问题其实传统关系数据库已经帮我们解决好了。无论是Oracle还是MySQL，都支持一个叫做两级分区表的概念。大部分业务都可以按两个维度划分数据：一个维度是时间，数据是按照时间顺序生成的；另外一个维度，对于互联网业务来讲，往往就是用户。不同的用户生成不同的数据，不同用户之间的数据相关度比较低，而同一个用户的数据往往会被同时访问。

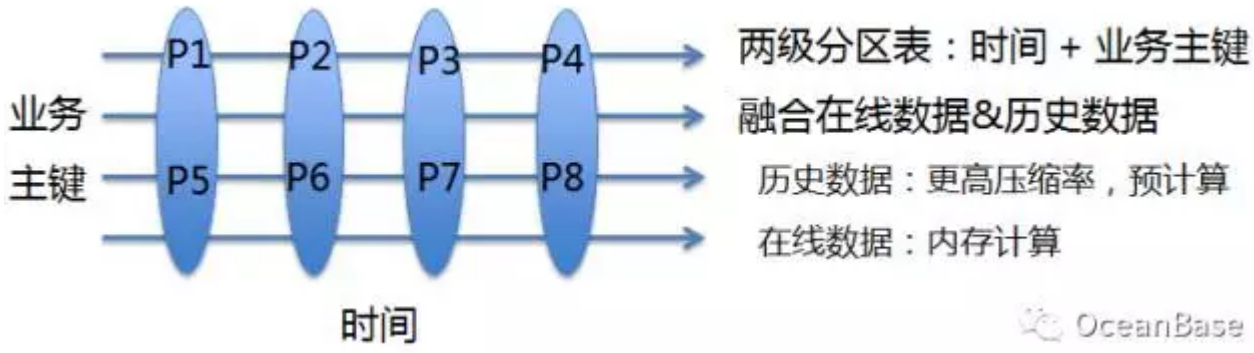


图1 OceanBase数据分布

如图1，通过时间和业务主键两个维度将表格划分为P1~P8总共8个分区。OceanBase跟传统数据库不一样的地方在哪里呢？传统数据库所有的分区只能在一台服务器，而OceanBase每个分区可以分布到不同的服务器。从数据模型的角度看，OceanBase可以被认为是传统的数据库分区表在多机的实现。对于常见的分布式系统，要么采用哈希分区，要么采用范围分区。OceanBase的数据划分方案和这些系统有较大的差别，通过两级分区表，我们可以把不同的用户，以及在不同时间点生成的数据全部融合到统一的表格里面。无论这些分区在在多台服务器上是如何分布的，甚至可

以对在线数据采用内存计算，对历史数据采用更高压缩率的压缩算法或者执行预计算，整个系统对使用者呈现的都是一张表格，后台实现对使用者完全透明。当然，这里面还会有很多的工作要做。

第二点是我们底层的分布式架构。

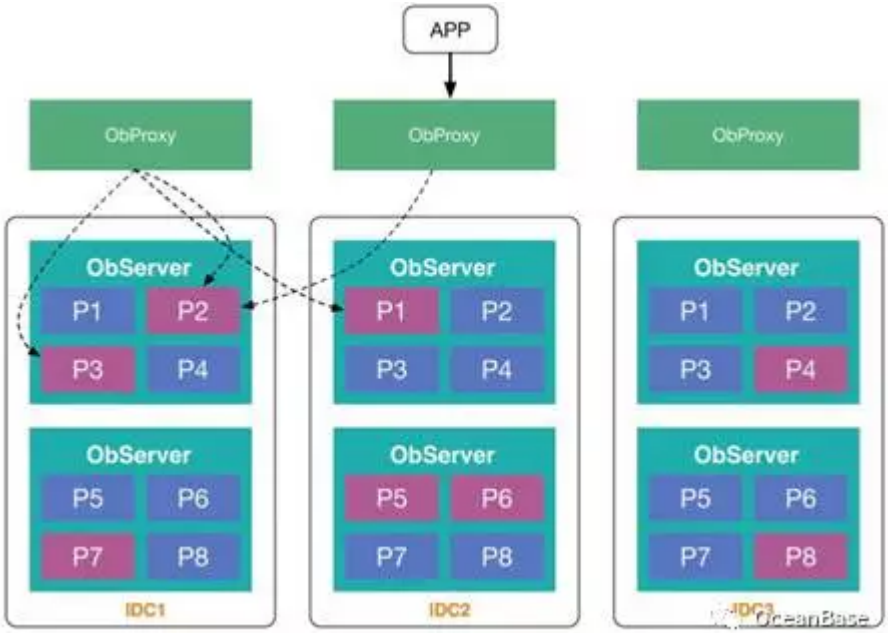


图2 OceanBase整体架构

图2是OceanBase整体架构图。OceanBase架构图往往都是三个框框。为什么这样呢？这是基于高可用考虑的。为了实现机房故障无损容灾，OceanBase需要部署到三个机房。每个机房又会有很多服务器，每台服务器又会服务很多不同的分区。如图2， P1到P8代表不同的分区，每个分区有3个副本，分布在三个机房内。用户的请求首先发给ObProxy。ObProxy是一个访问代理，它会根据用户请求的数据将请求转发到合适的服务器。ObProxy的最大的亮点在于性能特别好，我们对它做了非常多的针对性优化，使得它可以在非常一般的普通服务器上达到每秒百万级的处理能力。ObServer是OceanBase的工作机，每台工作机服务一些分区。

与大多数分布式系统不同的地方在于，OceanBase这个系统没有单独的总控服务器/总控进程。分布式系统一般包含一个单独的总控进程，用来做全局管理、负载均衡，等等。OceanBase没有单独的总控进程，我们的总控是一个服务，叫做RootService，集成在ObServer里面。OceanBase会从所有的工作机中动态地选出一台ObServer执行总控服务，另外，当总控服务所在的ObServer出现故障时，系统会自动选举一台新的ObServer提供总控服务。这种方式的好处在于简化部署，虽然实现很复杂，但是大大降低了使用成本。

接下来我们看整个数据库最核心的部分。

数据库最基础，也是最核心的部分是事务处理，这点对于多机系统尤其重要。

如果只是操作单个分区，因为单个分区一定只能由一台服务器提供服务，本质上是一个单机的事务，OceanBase的实现机制和Oracle、MySQL这样的传统数据库原理类似，也是多版本并发控制。不同点在于，OceanBase做了一些针对内存数据库的优化，主要有两点：

1. 日志同步。因为要做高可用，一定要做日志的强同步。OceanBase之所以既能做到高可用，又能做到高性能，是因为我们做了很多针对日志强同步的优化，包括异步化、并行，等等；
2. 内存数据库。OceanBase借鉴了内存数据库的设计思路，实现了无锁数据结构、内存多版本并发控制、SQL编译执行等优化手段。

如果操作多个分区，这又分成两种情况：

1. 多个分区在同一台服务器。由于多个分区在一台服务器，本质上也是单机事务，实现方案与之前提到的单分区事务类似。
2. 多个分区分布在多台服务器上。由于多个分区跨ObServer，OceanBase内部通过两阶段提交实现分布式事务。当然，两阶段提交协议性能较差，OceanBase内部做了很多优化。首先，我们提出了一个表格组的概念，也就是说，我们会把多个经常一起访问，或者说访问模式比较类似的表格放到一个表格组里面。与此同时，OceanBase后台会将同一个表格组尽可能调度到一台服务器上，避免分布式事务。接下来的优化手段涉及到两阶段提交协议的内部实现。两阶段提交协议涉及多台服务器，协议中包含协调者、参与者这两种角色，参与者维护了每台服务器的局部状态，协调者维护了分布式事务的全局状态。常见的做法是对协调者记日志来持久化分布式事务的全局状态，而OceanBase没有这么做。如果协调者出现故障，OceanBase通过查询所有参与者的状态来恢复分布式事务。通过这种方式节省了协调者日志，而且只要所有的参与者都预提交成功，整个事务就成功了，不需要等协调者写日志就可以应答客户端。

OceanBase里面高可用是基于Paxos协议实现的。Google Chubby系统的发明者说过一句话，这个世界上所有的高可用强一致的协议都是Paxos或者Paxos的变种，我们也认为一切不是Paxos协议实现的高可用机制都是耍流氓。


图3 OceanBase高可用原理

如图3，OceanBase最底层的技术就是通过Paxos协议实现的分布式选举和多库多活。Paxos协议的好处在于节点是多活的，当服务节点出现故障时，其它正常的节点可以在几秒内替代这个故障的节点，很快恢复服务，而且完全不丢数据。Paxos协议同时实现了高可用和强一致，这是很牛的。

当然除了最底层核心的Paxos协议，OceanBase还会通过分布式调度将同一个分区的不同副本调度多多个机房，而不是在一个机房或者一个机架里面。当服务器出现故障，OceanBase客户端能够很快自动感知到，并把服务切到没有故障的服务器上。通过这一系列组合拳，OceanBase真正做到了持续可用。

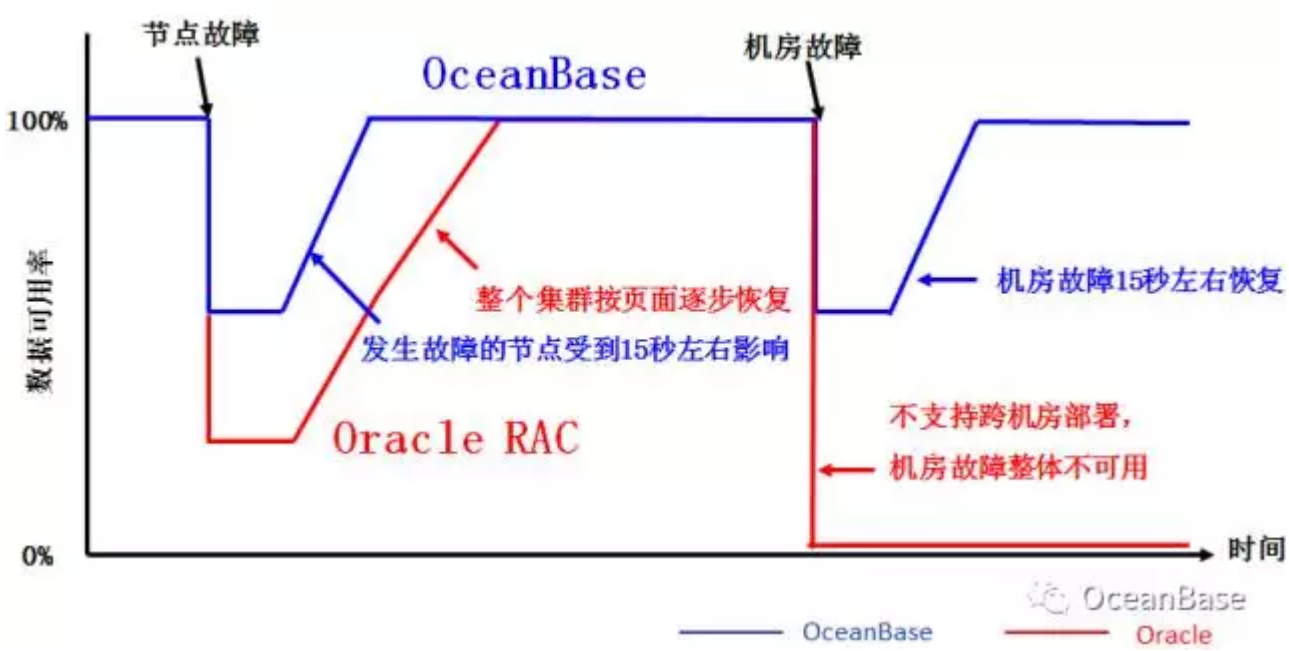


图4 高可用：OceanBase VS Oracle

图4是OceanBase和Oracle的高可用能力对比。

1. 单个节点故障，相比Oracle RAC，OceanBase单个节点故障的影响会小一些，为什么？因为OceanBase的集群规模往往会比较大，单个节点的故障只影响很小一部分数据。另外，OceanBase的恢复速度也会更快。OceanBase采用基线加增量的存储引擎，最热的增量数据都是全部在内存的。当主库出现故障，备库只需要把最后一部分未完成的日志回放出来就能够提供服务，非常快。而Oracle底层存储是基于页面的，它需要逐步恢复页面，这个过程相对更长。
2. 机房整体故障。OceanBase在蚂蚁都是三机房部署的，当一个机房出现故障，OceanBase也能够做到几十秒恢复，和单个节点故障的恢复速度基本相当。但是，传统的关系数据库，即使是Oracle也做不到，Oracle RAC不太可能部署到多个机房的。

接下来我们看看OceanBase的性能。长远来看，性能取决于底层的引擎如何设计，而OceanBase的引擎跟传统数据库的引擎有较大的差别。

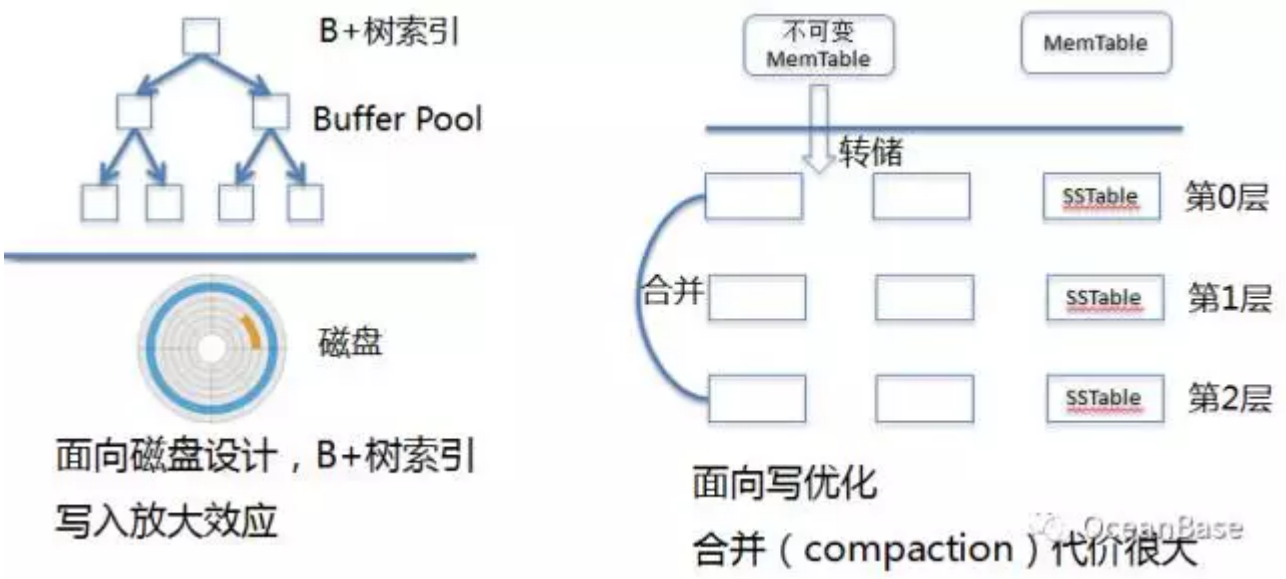


图5：两类数据库引擎

在互联网里面能够看到各种各样的存储系统，引擎非常多，我认为基本可以分为两个大类。如图5，第一类是传统关系数据库引擎。关系数据库本质上是面向磁盘设计的，它把数据分成很多很多的页面，通过页面缓存机制来管理页面，底层的索引是B+树。这种引擎发展得非常成熟，但是写入性能差，原因是关系数据库的写入放大效应。用户数据是按行写入的，但是关系数据库却是按页面管理的，有时只写了一行数据，却不得不把整个页面刷入磁盘。另外一类互联网公司流行的引擎是LSM树。Google里面有一个很有名的开源系统LevelDB，Facebook基于它又开发了一个类似的系统RocksDB。这种引擎采用基线加增量的设计，数据首先以增量的形式写入到内存中，当增量达到一个阈值时统一写到磁盘。这种做法的好处是解决了关系数据库写入放大的问题，但是它的合并代价会比较大，可能出现合并速度赶不上内存写入的情况。

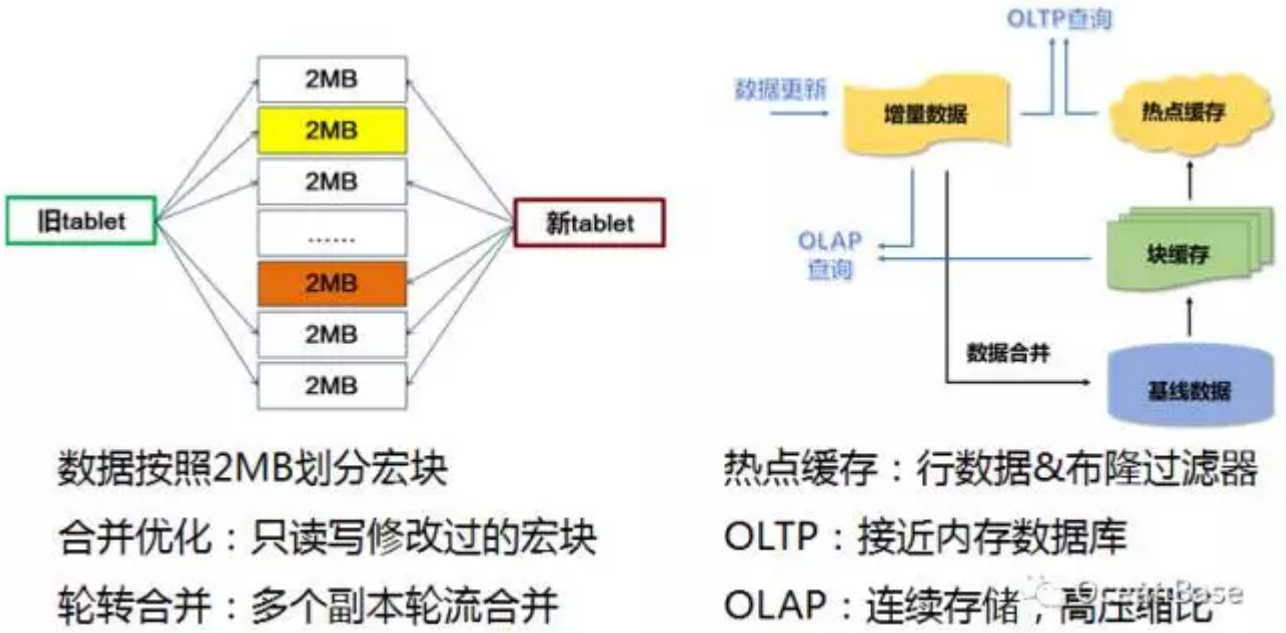


图6 OceanBase存储引擎

OceanBase本质上是一个基线加增量的存储引擎，跟关系数据库差别很大，但是我们借鉴了传统关系数据库的优点对引擎进行了优化。如图6，第一个优化就是合并性能的优化，传统数据库把数据分成很多页面，我们也借鉴了传统数据库的思想，把数据分成很多2MB为单位的宏块。执行合

并时，如果只有一部分数据修改，那么，只需要合并那些修改过的宏块，而不是把所有没有修改的宏块也一起合并。通过这个方式，OceanBase的合并代价相比LevelDB和RocksDB都会低很多，这是第一点。第二，我们会利用OceanBase的分布式机制做优化。在多机系统中，数据一定是存在多个副本。OceanBase实现了一个称为轮转合并的机制。当主副本提供服务时，其它副本可以执行合并；等到其它副本合并完成以后，原来的主副本接着合并，并把流量切到已经合并完成的副本。通过这种方式，OceanBase把正常服务和合并时间错开，使得合并操作对正常用户请求完全没有干扰。

由于OceanBase采用基线加增量的设计，一部分数据在基线，一部分在增量，原理上每次查询都是既要读基线，也要读增量。为此，OceanBase做了很多的优化，尤其是针对单行的优化。OceanBase内部把很多小查询的结果缓存在内存里面，以行为单位，而不是以块为单位。行缓存包括两个部分，如果这个行存在，缓存这个行的原始数据；否则，缓存布隆过滤器。OLTP业务大部分操作为小查询，通过小查询优化，OceanBase避免了传统数据库解析整个数据块的开销，达到了接近内存数据库的性能。另外，由于基线是只读数据，而且内部采用连续存储的方式，OceanBase可以采用比较激进的压缩算法，既能做到高压缩比，又不影响查询性能，大大降低了成本。

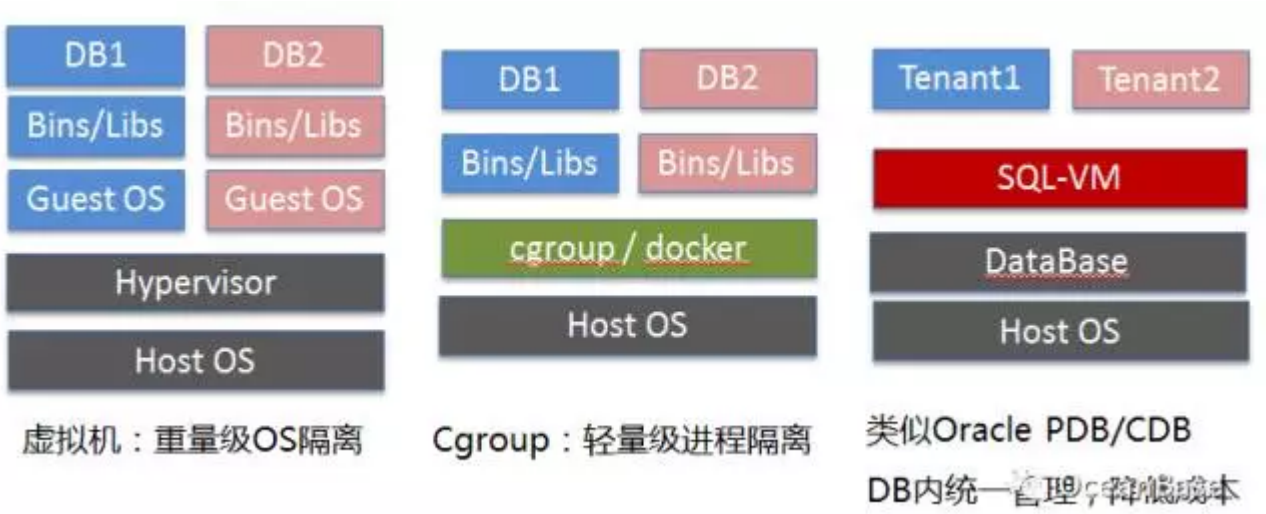


图7 SQL-VM

多租户隔离有不同的实现方式，这些实现方式的效果是类似的。最传统的实现方式是虚拟机。这种方案比较成熟，在云环境也经常使用，然而，这种方案比较重量级，开销比较大。另外一种最近比较流行的技术类似Google Cgroup这样的轻量级隔离技术，或者一些Cgroup衍生的技术，比如Docker。OceanBase的隔离方式很不一样，我们采用的是在数据库内部做了一个SQL虚拟机。

为什么这么做呢？这种做法在业内也是有例可寻的，例如Oracle 12c以及Azure SQL Server。它的好处是DB内把很多业务统一的管理，会把整个管理机制做得对用户特别透明。另外，隔离的开销比较低，单台服务器可以服务更多的租户，降低云服务的整体成本。

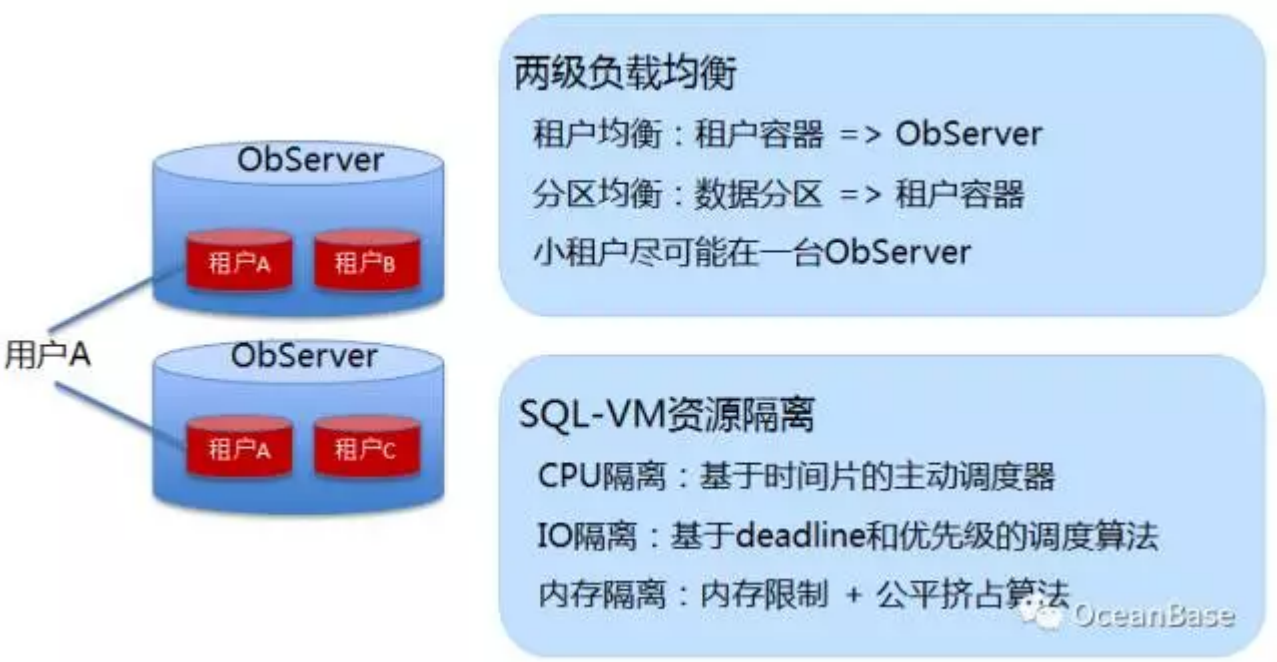


图8 OceanBase多租户原理

OceanBase是分布式系统，多租户除了单机层面怎么做隔离，还涉及到在多机层面如何做调度。如图8，在OceanBase中，有的租户比较小，位于一台服务器上，例如租户B或者租户C；有的租户比较大，位于多台服务器上，例如租户A。图8中的租户A在每台服务器上有一个资源容器，限制了该租户能够使用的CPU、网络、内存、IOPS等系统资源。OceanBase的负载均衡分为两个层面：第一个层面是租户负载均衡，它的原理就是把每个租户的资源容器分布到很多台ObServer上面去；第二个层面是分区负载均衡，如果租户只在一台服务器，第二个层面是没有必要的。如果租户在多台服务器上，需要把这个租户的分区均匀地分布到它的资源容器中。OceanBase内部会尽量使得小租户只在一台服务器上，避免分布式事务。当租户需要的资源逐步增加时，OceanBase也能做到自动扩展，对用户是透明的。

SQL-VM是OceanBase实现的资源隔离方案，分为三个部分：CPU、IO还有内存，网络目前还不是瓶颈，我们做得比较少。CPU隔离是基于时间片的主动调度，跟操作系统的调度比较类似。IO隔离我们借鉴了虚拟机隔离方案，采用基于Deadline和优先级的隔离算法。这里最大的难点在于既要把资源的利用率提高，又要做到尽可能公平，这两个目标本身是矛盾的。内存隔离基本上是两个机制，一个机制是怎么做内存的限制，另外一个机制是怎么做公平挤占算法。

分布式查询

用户的请求首先会发送到一台称为ObProxy的代理服务器，它的功能就是一个透明转发的代理服务。ObProxy会解析SQL，识别SQL操作哪个分区，然后把该SQL转发到它所操作的分区所在的服务器。

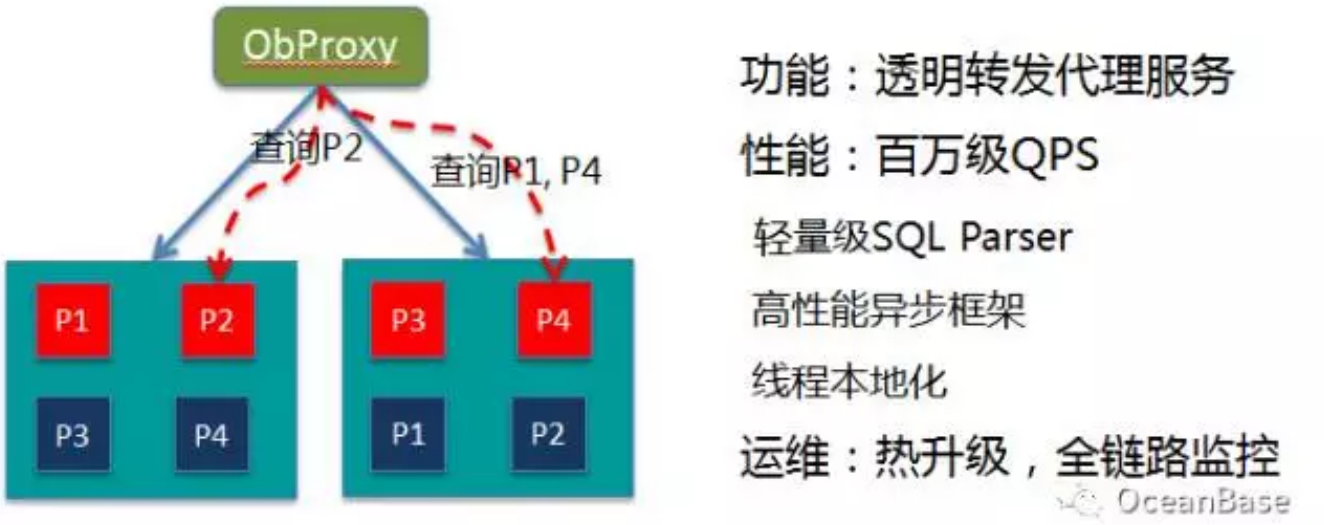


图9 ObProxy原理

如图9，可以看到，如果SQL请求操作分区P2，ObProxy就会转发到左边的服务器；如果涉及到多个分区，例如同时操作P1、P4，由于它们位于不同的服务器，ObProxy就会随机或者根据负载选择一台服务器。

ObProxy最大的亮点还是在于它的性能。世面上有很多数据库代理服务方案，ObProxy的性能是其中的佼佼者。我们实现了大量的优化技术，包括：

1. 轻量级SQL Parser。如果采用传统的数据库SQL解析框架，性能肯定是不高的，OceanBase做了大量的手工解析工作。
2. 异步框架。ObProxy是全异步框架，无论是SQL转发，还是网络收发包，所有操作都是全异步的，从而把CPU利用到极致。
3. 可扩展性。ObProxy内部会使得对一个请求的转发尽可能做到每一个线程内部，避免跨线程访问或者访问全局的数据结构。我们基本上做到每个请求都是在一个线程内完成的，ObProxy可以做到随着CPU核心完全线性扩展。

除了性能卓越之外，ObProxy还实现了在实际生产系统非常关键的运维特性。第一个运维特性是热升级，ObProxy的升级对使用者是无感知的。升级过程中，ObProxy内部会有两个版本，新的请求会由新的版本提供服务，老的请求由老的版本提供服务，过了很长时间才会把老版本退出。另外一个运维特性是全链路监控，ObProxy作为用户使用OceanBase的入口，和服务端一起联合实现了全链路监控功能。

当SQL请求到了ObServer服务端，经过SQL解析、重写、优化等一系列过程后，再由SQL执行器来负责执行。分为三种情况：

1. 本地计划。如果ObProxy请求转发对了，转发的位置就是分区所在的ObServer，这是个本地请求，其执行方式和传统的单机数据库相同。

2. 远程计划。SQL请求只操作一个分区，但是ObProxy应用缓存的信息失效，或者出现一些BUG，把请求转发到另外一个服务器。OceanBase会自动发现这一点，并把SQL当成远程计划来处理，即内部把SQL执行计划发到另外一台有数据的服务器，等那台服务器计算出结果再返回。
3. 分布式计划。如果SQL请求涉及的数据在多台服务器，需要走分布式计划。这种请求涉及的数据量往往比较大，OceanBase后台会做并行处理，比如说对任务做拆分，对每个子任务的结果做合并、处理并发、限制并发数，等等。基本上可以这么讲，在并行数据库或者OLAP分析型数据库看到的东西，在OceanBase的分布式计划中都有。

经验&思考

第一点关于高可用。首先，我认为强一致加高可用是未来云数据库的标配。我们以前做应用架构和存储选型的时候会谈很多东西，比如数据库异步复制提高性能，或者CAP理论导致一致性和高可用不可兼得，等等。然而，通过OceanBase的实践我们也已经得出一个结论，实现强一致相比实现弱一致性能开销不是那么大，性能的损耗比较低，而且可以获得非常多的好处，这一定是以后的趋势。另外，实现云数据库级别的高可用底层一定要用Paxos协议，回避该协议的实现方案本质上都是耍流氓。目前我们已经从各大互联网公司，包括Google、Amazon以及Alibaba的云数据库实践看到了这一趋势。

第二点关于自动化。以前传统的关系数据库规模往往比较小，只有少数几台机器，有一个DBA运维就够了。然而，在云数据库时代，一定要做规模化运维，一个DBA对几千甚至上万机器，达到这个量级一定要做自动化。强一致是自动化的前提，没有强一致很难自动化，主库故障备库会丢数据，自动化是很难的。虽然每次宕机概率很低，但是规模上来概率就高了。

另外云数据库设计的时候也会有很多不一样的考虑，尽可能地减少人工干预。例如数据库配置项，或者比较流行的SQL Hint，在云数据库时代一定需要尽可能减少。系统设计者需要在服务端自己消化，不要把灵活性留给运维的人员，因为运维的人要做的就是规模化运维。

最后一点关于成本。成本是云的关键，很多用户上云就是为了节省成本。首先，性能不等于成本。性能虽然很重要，但是成本需要考虑更多的因素，性能、压缩比、利用率、规模化运维，等等。以利用率为例，云数据库的一个成本节省利器就是提高机器利用率。在云数据库中，可以通过分布式方案把整个利用率提高来，即使单机性能类似，但是利用率提上来以后，成本会特别的低。

最后是OceanBase的存储引擎跟传统数据库有很大的区别，是基线加增量引擎，这种引擎有特别多的好处，能够比较完美地解决OLTP和OLAP业务融合的问题。如果实现得当，把这种引擎的缺陷规避地比较好，能够收获大量的好处，相信也是未来的趋势。

1. 独立完成中小型项目的系统分析、设计，并主导完成详细设计和编码的任务，确保项目的进度和质量；
2. 能够在团队中完成code review的任务，确保相关代码的有效性和正确性，并能够通过code review提供相关性能以及稳定性的建议；
3. 参与建设通用、灵活、智能的业务支撑平台，支撑上层多场景的复杂业务。
岗位要求:
1. 扎实的java编程基础，熟悉常用的Java开源框架；
2. 具有基于数据库、缓存、分布式存储开发高性能、高可用数据应用的实际经验，熟练掌握LINUX操作系统；
3. 具备良好的识别和设计通用框架及模块的能力；
4. 热爱技术，工作认真、严谨，对系统质量有近乎苛刻的要求意识，善于沟通与团队协作；
5. 具备大型电子商务网站或金融行业核心系统开发、设计工作经验者优先。

**Java技术专家-成都**
职位描述
引入多元的思考、平台化能力、创新解决方案，一起面向蚂蚁金服整体场景解决高可用，稳定性，支付技术等方面问题，具体工作职责如下：
1、独立完成较复杂的系统分析、设计，并主导完成详细设计和编码的任务，确保项目的进度和质量；
2、能够在团队中完成code review的任务，确保相关代码的有效性和正确性，并能够通过code review提供相关性能以及稳定性的建议；
3、参与建设通用、灵活、智能的业务支撑平台，支撑上层多场景的复杂业务。"
岗位要求
1、5年以上大规模、高吞吐量的系统开发实践经验；
2、精通分布式系统和架构，对高性能、持续可用架构的最佳实践以及设计原则有深刻理解；
3、对技术有激情，喜欢钻研，能快速接受和掌握新技术，有较强的独立、主动的学习能力，良好的沟通表达能力和团队协作能力；
4、3年以上研发和架构经验，对JAVA技术有较深刻的理解；
5、热爱技术，工作认真、严谨，对系统质量有近乎苛刻的要求意识，善于沟通与团队协作；
6、具备大型电子商务网站或金融行业核心系统开发、设计工作经验者或有复杂系统的数据建模能力的人才优先。

**数据研发专家**
岗位描述:
1.参与资金、财务、监管相关的数据集市规划、设计、分析与挖掘；
2.参与资金风险相关项目的数据模型设计与研发。
岗位要求:
1.从事数据仓库或挖掘领域至少5年以上，熟悉数据仓库模型设计与ETL开发经验 ，掌握Kimball的维度建模设计方法，具备海量数据处理经验；
2.熟悉数据仓库领域知识和技能者优先，包括但不局限于：元数据管理、数据开发测试工具与方法、数据质量、主数据管理；

3.有从事分布式数据存储与计算平台应用开发经验，熟悉Hadoop生态相关技术并有相关实践经验着优先，如MR、Hive、Hbase、Spark、Storm；
4.熟练掌握一门或多门编程语言，并有大型项目建设经验者优先，如Java、Python、Shell；
5.熟悉图数据、图算法者优先；
6.良好的语言沟通与表达能力和自我驱动动力。

**高级算法工程师/算法专家**
具体职责包括但不限于：
1、负责机器学习、特征提取领域的技术和算法研发工作，包括但不限于强化学习、迁移学习、主动学习、维度降低、降噪算法、特征提取、推荐、随机优化等的算法和系统研发等；
2、负责机器学习、降噪算法落地和算法优化，并落地应用于支付，账务，汇兑，清算，核算，金融网络，资产负债管理，金融风险分析等系统平台实践中；
3、研究分析业内智能算法平台产品。优化技术方案，改进产品功能； 将能力高效应用于技术风险分析、业务分析和智能决策等业务产品中。
岗位要求:
1、在机器学习或数据挖掘方向有较强的积累，熟悉经典的算法并有实践经验，包括LR、SVM、GBDT、DNN、LSTM等；对算法优化有一定经验，有效提升准确率；
2、精通至少一门语言，Java/C++/Python/Matlab/R等，具有扎实的代码功底和实战能力；
3、对数据敏感，分析数据、抽象问题、理解并解决问题，对使用机器学习解决金融系统问题有热情；
4、有深度学习、迁移学习经验更佳；
5、较强的沟通能力和团队协作能力，善于学习新知识、乐于接触新场景、关注前沿新技术。

**高级测试开发工程师/质量专家**
岗位描述:
1. 参与软件项目的需求分析，关注项目需求的合理性，可测性；
2. 参与重大产品需求和架构设计评审，保证产品设计与架构的合理性；
3. 设计合理的测试用例，并能参与到具体的测试执行工作中，跟进缺陷的fix、质量复核，能引入比较好的思想和方法，保证产品的质量；
4. 参与自动化用例的编写，设计&参与产品的性能、稳定性测试，解决测试过程中的复杂技术问题。
5. 根据产品研发的实际情况，改进or设计软件质量保证体系，并推动落地实施；
6. 通过测试相关流程、策略、方法、数据、技术、工具等创新，提升测试的质量和效率；
7. 能结合行业发展趋势，制定合适的质量技术发展规划，促进团队质量保障效率和团队技能的提升，指引团队测试技术的发展方向。
岗位要求:
1. 具有2年以上软件开发和测试经验，精通测试用例设计方法及常见自动化测试技术，独立承担过中大型项目测试负责人；
2. 熟悉java，熟悉java常用框架，能完成测试工具、测框架开发要求；
3. 有撰写自动化测试工具以及搭建自动化测试平台的实战经验，有自动化测试经验，设计用例并编写代码实现自动化测试；

4. 性格开朗乐观，责任心强，工作积极主动，具备良好 的沟通能力和团队协作能力；
5. 在某一测试领域比如性能、稳定性、自动化、数据化等具备很强的专业技能者优先;