# 背景

关系数据库架构的杰出代表是我们常说的IOE架构：I指的是IBM小型机、大型机；O指的是Oracle数据库；E指的是EMC的存储。IOE基于专用软硬件，本质上是一个封闭体系，这个体系的最大问题在于不够开放，产生了两个问题： 一是价格非常昂贵。摩尔定律大幅降低了开放的X86硬件的成本，而专用硬件无法享受技术红利，一套专用硬件往往需要上千万甚至上亿人民币；  
 二是几乎无法横向扩展，只能支持垂直扩展，通过购买更好的硬件来提升系统的处理能力，从而错过了云和分布式时代的技术变革。  
 传统集中式数据库面向的是封闭体系下的Mission Critical业务场景，处理的是少量用户最为关键的基础数据和事务数据，比如账户、交易数据；而互联网是开放体系，处理的是大量用户的全部行为数据，这些行为数据不仅仅包含基础数据和事务数据，也包含交互数据，也就是海量用户并发访问过程产生的日志。  
 这将导致两类问题：

一是现象级应用带来的可扩展和高并发需求。如天猫双十一零点峰值的流量是平常的几十上百倍，需要数据库系统具备快速弹性扩容的能力；  
 二是海量数据带来的分布式架构。集中式架构把硬件故障当成异常处理，而大规模分布式架构把硬件故障看成正常现象，这是一种新常态，需要在软件层面实现自动容错，且做到低成本存储。

# 技术挑战

# 概述

参考：

<https://blog.csdn.net/seteor/article/details/10532085>

<https://www.cnblogs.com/xiaotengyi/articles/3585465.html>

<https://www.eygle.com/archives/2017/03/oracle_sharding.html>

# 发展

纵观分布式数据库的发展，我认为分布式数据库经历了三代：

第一代分布式表格系统是NoSQL系统，代表作是Google Bigtable；

第二代分布式数据库支持可扩展的SQL处理，代表作是Google Spanner；

第三代分布式数据库是透明扩展的企业级数据库，兼容传统集中式数据库，支持HTAP混合负载，且单机性价比很高，代表作是OceanBase。

注：这里的分类将分库分表中间件的方式没有算在分布式数据库架构中，这个还是属于传统的一种架构，不是NewSQL。

## 第一代分布式数据库NoSQL



第一代分布式数据库其实是NoSQL系统。为了实现可扩展，第一代分布式数据库牺牲了SQL和事务，不支持SQL语言，不支持跨机事务，只支持单行事务或者单机事务，部分NoSQL系统甚至牺牲一致性。11年到13年之间NoSQL系统很流行，当时很多人认为数据库性能瓶颈的根源在于SQL，只要牺牲SQL语义，就能够一劳永逸地解决高并发场景下数据库面临的可扩展性和单机性能问题。当时有一些流行的说法，比如"SQL is dead"，"one size does not fit all"，走向了另一个极端，但是做着做着却发现 NoSQL 系统中用到的核心技术大多都源于关系数据库。OceanBase团队12年也有争论，到底是走NoSQL路线还是坚持做关系数据库，最终我们还是决定做关系数据库，这才走到了今天。  
 最流行的两个NoSQL系统分别是Amazon Dynamo和Google Bigtable。



他们的底层存储分别是一个分布式哈希表和一棵分布式 B+树。Dynamo采用去中心化的设计，引入一致性哈希来做数据分布，并采用NWR协议，要求写入的副本个数W加上读取的副本个数R大于总副本数N。Dynamo系统最大的问题在于牺牲了一致性，需要用户处理冲突，这个做法最后证明是失败的，Amazon后期设计的分布式存储系统DynamoDB就没有沿用这个做法，而是在内部通过Paxos协议保证强一致性。  
 Google Bigtable构建在GFS之上，实现了强一致性，自动将表格划分为子表 tablet，实现了tablet的自动分裂与合并。Bigtable只支持单行事务，之所以只有单行事务，我认为根本原因还在于分布式事务实现过于复杂，不容易做到高效。Bigtable有两个开源的模仿者，一个是Hypertable，另一个是今天非常流行的 HBase。

## 第二代分布式数据库

第一代NoSQL系统对用户很不友好，关系数据库的性能和扩展性问题也并不是因为支持SQL，于是进一步产生了第二代分布式数据库。



第二代分布式数据库以Google Spanner为代表。Spanner支持大部分SQL，支持分布式事务，但不兼容SQL标准。Spanner通过Truetime实现了分布式事务全局时间戳，保证了强一致性，但每次事务提交延时很高，牺牲了单机性能。Spanner是第一个全球级的分布式数据库，很好地解决了Scalable SQL问题，但牺牲了关系数据库最核心的性价比等企业级特性。Spanner适合应用在对扩展性要求特别高的特定应用场景，例如Google本身，但不适合应用在传统行业的 Mission Critical业务场景。

## 第三代分布式数据库

第三代分布式数据库是透明扩展的企业级数据库，以OceanBase为代表。

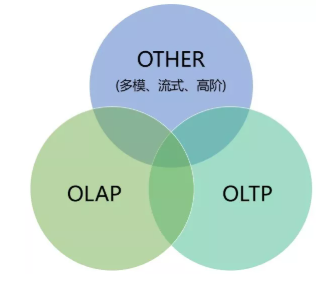


第三代分布式数据库的设计理念是把复杂留给基础设施，把简单留给数据库使用者：业务开发人员和运维人员。  
 它的底层是可扩展的分布式架构，从而享受分布式技术的红利，如高可用、可扩展，兼容传统企业级数据库的功能，在同一套数据库引擎中支持HTAP混合负载，并同时，追求极致的单机性价比。  
 追求极致性价比这一点将从根本上影响分布式数据库的设计，举几个例子： 1、Google Spanner系统通过Truetime机制获取全局时间戳，这个方案导致事务延迟太高，需要改变；  
 2、分布式数据库可以划分为SQL层和存储层，Google Percolator系统的事务和并发控制机制构建在存储层之上，采用松耦合的设计，影响性能。为了追求极致性价比，需要和传统关系数据库一样，把事务层放到存储层内部，采用紧耦合的设计。  
 3、为了追求性能极致，需要扣实现细节，尽可能优化读写流程的CPU指令数，采用C/C++语言实现，手动执行内存管理。

# 分类

## 数据应用类别

根据数据的使用特征，可简单做如下划分。在选择技术平台之前，我们需要做好这样的定位。



### OLTP

联机事务处理OLTP（On-Line Transaction Processing），OLTP是事件驱动、面向应用的，也称为面向交易的处理过程。其基本特征是前台接收的用户数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果，是对用户操作的快速响应。例如银行类、电子商务类的交易系统就是典型的OLTP系统。其具备以下特点：

直接面向应用，数据在系统中产生。

基于交易的处理系统。

每次交易牵涉的数据量很小；对响应时间要求非常高。

用户数量非常庞大，其用户是操作人员，并发度很高。

数据库的各种操作主要基于索引进行。

以SQL作为交互载体。

总体数据量相对较小。

### OLAP

联机实时分析OLAP（On-Line Analytical Processing），OLAP是**面向数据分析**的，也称为面向信息分析处理过程。它使分析人员能够迅速、一致、交互地从各个方面观察信息，以达到深入理解数据的目的。其特征是应对海量数据，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。例如数据仓库是其典型的OLAP系统。其具备以下特点：

本身不产生数据，其基础数据来源于生产系统中的操作数据

基于查询的分析系统；复杂查询经常使用多表联结、全表扫描等，牵涉的数量往往十分庞大

每次查询设计的数据量很大，响应时间与具体查询有很大关系

用户数量相对较小，其用户主要是业务人员与管理人员

由于业务问题不固定，数据库的各种操作不能完全基于索引进行

以SQL为主要载体，也支持语言类交互

总体数据量相对较大

OLTP VS OLAP



### OTHER

除了传统的OLTP、OLAP类，近些年来针对数据的使用又有些新特点，我将其归入了“其他”类。

#### 多模

随着业务“互联网化”和“智能化”的发展以及架构 “微服务”和“云化”的发展，应用系统对数据的存储管理提出了新的标准和要求，数据的多样性成为突出的问题。早期数据库主要面对结构化数据的处理场景。后面随着业务的发展，逐渐产生了对非结构化数据的处理需求。包括结构化数据、半结构化(JSON、XML等)数据、文本数据、地理空间数据、图数据、音视频数据等。多模，正是指单一数据库支持多种类型数据的存储与处理。

#### 流式

流式处理(实时计算)，是来源于对数据加工时效性的需求。数据的业务价值随着时间的流失而迅速降低，因此在数据发生后必须尽快对其进行计算和处理。传统基于周期类的处理方式，显然无法满足需求。随着移动互联网、物联网和传感器的发展导致大量的流式数据产生。相应地出现了专有的流式数据处理平台，如Storm、Kafka等。近些年来，很多数据库开始支持流式数据处理，例如MemSQL、PipelineDB。有些专有流式数据处理平台开始提供SQL接口，例如KSQL基于Kafka提供了流式SQL处理引擎。

#### 高阶

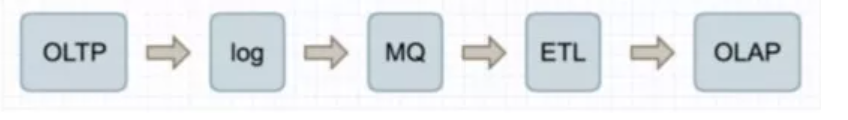
随着对数据使用的深入，数据的使用不再仅仅以简单的增删改查或分组聚合类操作，而对于其更为高阶的使用也逐步引起大家的重视。例如使用机器学习、统计分析和模式识别等算法，对数据进行分析等。

## 数据处理模式

面对上述复杂多变的应用场景，数据应用的多种类别，是由单一平台处理，还是由不同平台来处理呢？一般来说，专有系统的性能将比通用系统性能高一到两个数量级，因而不同的业务应采用不同的系统。但正如古人说“天下大势、分久必合、合久必分”，在数据处理领域也有一种趋势，由单一平台来处理。这里选择的核心在于如何来辩证看待需求和技术。它们是一对矛盾体，当这对矛盾缓和时，数据处理领域将更趋向于整合；而当这对矛盾尖锐时，数据处理领域将趋于分散。就软硬件技术发展现状和当前需求来看，未来整合的趋势更为明显。集成数据平台将能满足绝大多数用户的场景，只有极少数企业需要使用专有系统来实现其特殊的需求。

### 分散式（专有平台）

目前比较常规的方式，是采用多个专有平台，来针对不同场景进行数据处理。因此是跨平台的，因此是有个数据传输的过程。这之中会带来两个问题：数据同步、数据冗余。数据同步的核心是数据时效性问题，过期的数据往往会丧失价值。常见的做法如下：



OLTP系统中的数据变化，通过日志的形式暴露出来；通过消息队列解耦传输；后端的ETL消费拉取，将数据同步到OLAP中。整个链条较长，对于时效性要求较高的场景是个考验。此外，数据在链条中流动，是存在多份的数据冗余保存。在常规的高可用环境下，数据会进一步保存多份。因此这里面隐藏了比较大的技术、人力成本以及数据同步成本。而且横跨如此之多的技术栈、数据库产品，每个技术栈背后又需要单独的团队支持和维护，如DBA、大数据、基础架构等。这些都蕴含着巨大的人力、技术、时间、运维成本。正是出于在满足各种业务需求的同时，提高时效性，减低数据冗余、缩短链条等，收敛技术栈就变得很重要。这也是通用类平台解决方案，诞生的出发点。

### 集中式/HTAP（通用平台）

用户厌倦了为不同的数据处理采用不同的数据处理系统，更倾向于采用集成数据处理平台来处理企业的各种数据类型。对于融合了联机事务处理和联机实时分析的场景，也就是HTAP。

在互联网浪潮出现之前，企业的数据量普遍不大，特别是核心的业务数据，通常一个单机的数据库就可以保存。那时候的存储并不需要复杂的架构，所有的线上请求 (OLTP，Online Transactional Processing) 和后台分析 (OLAP，Online Analytical Processing) 都跑在同一个数据库实例上。

随着互联网的发展，企业的业务数据量不断增多，单机数据库的容量限制制约了其在海量数据场景下的使用。因此在实际应用中，为了面对各种需求，OLTP、OLAP在技术上分道扬镳，在很多企业架构中，这两类任务处理由不同团队完成。当物联网大数据应用不断深入，具有海量的传感器数据要求实时更新和查询，对数据库的性能要求也越来越高，此时，新的问题随之出现：

1、OLAP和OLTP系统间通常会有几分钟甚至几小时的时延，OLAP数据库和 OLTP 数据库之间的一致性无法保证，难以满足对分析的实时性要求很高的业务场景。

2、企业需要维护不同的数据库以便支持两类不同的任务，管理和维护成本高。

因此，能够统一支持事务处理和工作负载分析的数据库成为众多企业的需求。在此背景下，由Gartner提出的HTAP（混合事务/分析处理，Hybrid Transactional/Analytical Processing）成为希望。基于创新的计算存储框架，HTAP数据库能够在一份数据上同时支撑业务系统运行和OLAP场景，避免在传统架构中，在线与离线数据库之间大量的数据交互。此外，HTAP基于分布式架构，支持弹性扩容，可按需扩展吞吐或存储，轻松应对高并发、海量数据场景。

此类通用平台方案具备下面优点：

通过数据整合避免信息孤岛，便于共享和统一数据管理。

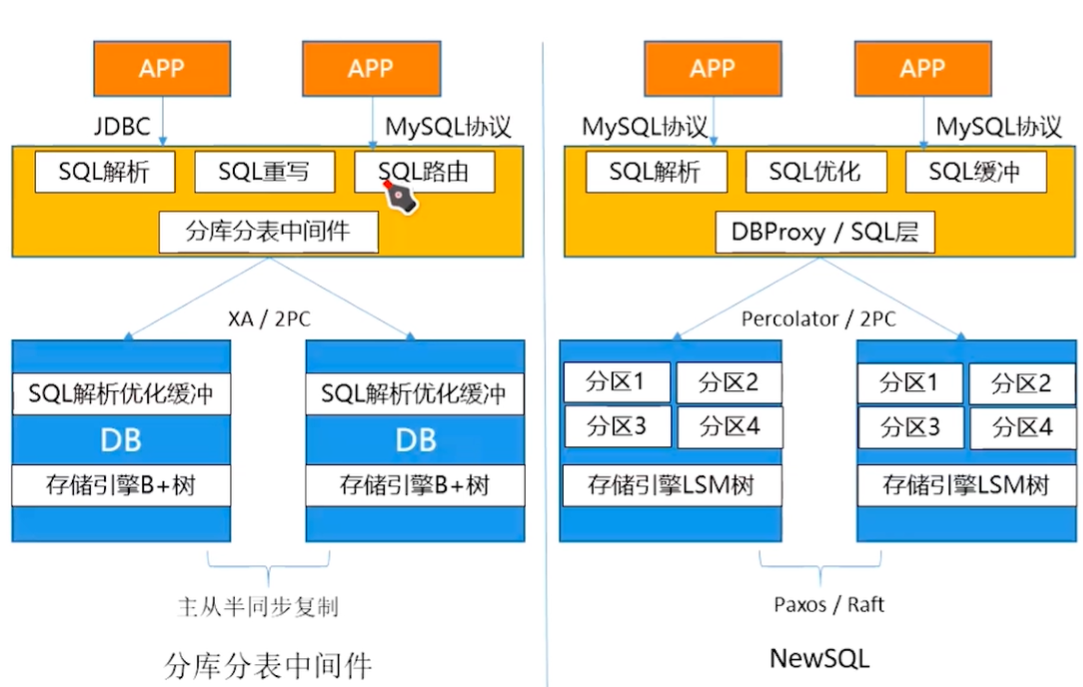
基于SQL的数据集成平台可提供良好的数据独立性，使应用能专注于业务逻辑，不用关心数据的底层操作细节。

集成数据平台能提供更好的实时性和更全的数据，为业务提供更快更准的分析和决策。

能够避免各种系统之间的胶合，企业总体技术架构简单，不需要复杂的数据导入/导出等，易于管理和维护。

便于人才培养和知识共享，无须为各种专有系统培养开发、运维和管理人才。

## 分片方式



### 分库分表中间件

中间件模式SQL解析、执行计划优化等在中间件与数据库中重复工作，效率相对比较低。

### NewSQL

#### 背景

NewSQL是一种新方式关系数据库，意在整合RDBMS所提供的ACID事务特性（即原子性、一致性、隔离性和可持久性），以及NoSQL提供的横向可扩展性。

NoSQL数据库给出了一种易于实现可扩展性和更好性能的解决方案，解决了CAP理论中的 A（可用性）和 P（分区容错性）上的设计考虑。但这意味着，在很多NoSQL设计中实现为最终一致性，摈弃了RDBMS提供的强一致性及事务的ACID属性。

NoSQL数据库使用了不同于关系模型的模型，例如键值模型、文档模型、宽列模型和图模型等。采用这些模型的NoSQL数据库并不提供规范化，本身在设计上是无模式的。大多数NoSQL数据库支持自动分区，无需开发人员干预即可轻松实现水平扩展。

NoSQL适用于可接受最终一致性的部分应用，例如社交媒体。用户并不关注看到的是否为不一致的数据库视图，并且考虑到数据的状态更新、发推文等，强一致性也并非必要的。但是，NoSQL数据库不宜用于对一致性要求高的系统，例如电子商务平台。

NewSQL系统的提出，正是为了满足整合NoSQL和RDBMS特性的需求。其中，NoSQL提供了可扩展性和高可用性，传统RDBMS提供了关系模型、ACID 事务支持和SQL。用户已不再考虑一招能解决所有问题（one-size-fits-all）的方案，逐渐转向针对OLTP等不同工作负载给出特定数据库。大多数NewSQL数据库做了全新的设计，或是主要聚焦于OLTP，或是采用了OLTP/OLAP的混合架构载的全新设计。

传统的RDBMS架构从一开始设计时并未考虑分布式系统，而是在分布式需求出现后，才考虑在最初的设计之添加支持分布式的设计。由于RDBMS实现了规范化模式，而非NoSQL那样的聚合表单，因此RDBMS中必须引入一些复杂的概念，才能在支持可扩展的同时保持一致性需求。由此，为支持RDBMS中的横向扩展，人们提出了手动分片和主从架构。

但是，RDBMS为实现横向扩展而在性能上做出了很大让步。这是因为连接运算中需要在各个节点间移动数据以实现聚合，运算实现代价增大。另外，数据维护开销变得更为耗时。为保持RDBMS的性能，一些企业推出了复杂的系统和产品。但是当前，人们依然并不认为传统RDBMS本身支持可扩展。

**NewSQL数据库为云时代而生，因此它从一开始就考虑了分布式架构。**

#### 概述

NewSQL数据库的分布式事务相比较于XA进行了优化，性能更高；新架构NewSQL数据库存储设计即为基于Paxos（或Raft）协议的多副本，相比较于传统数据库主从模式（半同步转异步后也存在丢数问题），实现了真正的高可用、高可靠（RPO<30s，RTO=0）。

NewSQL数据库天生支持数据分片，数据的迁移、扩容都是自动化的，大大减轻了DBA的工作，同时对应用透明，无需在SQL指定分库分表键。

这些是NewSQL数据库产品主要宣传的优点，两种架构成熟度都低于传统关系型数据库，SQL功能支持以及事务一致性、可靠性等都有待提高。

#### 特点

##### 一致性

相对于可用性而言，NewSQL更重视一致性，即侧重CAP中的C和P。很多NewSQL数据库为提供强一致性而牺牲了部分可用性。这些数据库为达成分布式一致性，在全局系统或本地分区层面使用了Paxos或Raft共识协议。MemSQL等一些解决方案还提供了一致性和可用性之间的权衡调优，支持不同用例的各种配置。

##### 内存数据库

传统RDBMS依赖二级存储（即磁盘）作为数据存储的介质。常用的二级存储包括SSD或HDD。鉴于OLTP工作负载可将历史数据归档到数据仓库中，因此并不需要大量的数据，只需要最新的数据。一些NewSQL解决方案使用内存（RAM）作为存储介质。内存访问要比磁盘访问快很多，具体而言，可比SSD快百倍，比HDD快万倍。

内存解决方案提供了更好的性能提升，因为内存的使用消除或简化了 缓存管理 和重度并发系统。鉴于内存中保持了全部数据（或是大部分数据），因此完全没有必要做缓存管理。对于并发而言，不同的实现有不同的解决方案，例如序列化等。

那么如何解决持久性问题？RAM本身是非持久介质。一旦掉电，需要持久化的数据就会丢失。内存数据库采用了多种方式解决该问题。常用方法包括组合使用基于磁盘的非频繁备份、保存状态的日志以实现可恢复性，以及对关键数据使用非易失RAM介质。

##### HTAP特性

很多NewSQL数据库是完全重新设计的。正因为重新设计，一些项目希望实现统一支持事务处理和工作负载分析的数据库。HTAP（混合事务/分析处理，Hybrid Transactional/Analytical Processing）一词由Gartner提出。支持HTAP功能的数据库提供对高级实时分析，进而支持实时业务决策和智能事务处理。VoltDB也提供HTAP能力，它更侧重于事务负载。其他主流HTAP数据库还包括TiDB和Google的Spanner。

##### 增强RDBMS

NewSQL也可以通过增强现有的RDBMS实现扩展的功能，无需完全重新设计数据库。这样的解决方案实现在经实战验证的SQL数据库之上，增强了现有数据库的功能。该理念对于那些现有系统运行良好而不愿意迁移到新数据库解决方案的大型企业是非常有用的。

#### 分布式事务

##### CAP限制

想想更早些出现的NoSQL数据库为何不支持分布式事务（最新版的mongoDB等也开始支持了），是缺乏理论与实践支撑吗？  
 并不是，原因是CAP定理依然是分布式数据库头上的颈箍咒，在保证强一致的同时必然会牺牲可用性A或分区容忍性P。为什么大部分NoSQL不提供分布式事务？  
 **那么NewSQL数据库突破CAP定理限制了吗？并没有。** NewSQL数据库的鼻主Google Spanner（目前绝大部分分布式数据库都是按照Spanner架构设计的）提供了一致性和大于5个9的可用性，宣称是一个“实际上是CA”的，其真正的含义是系统处于CA状态的概率高由于网络分区导致的服务停用的概率非常小，究其真正原因是其打造私有全球网保证了不会出现网络中断引发的网络分区。  
 另外就是其高效的运维队伍，这也是cloud spanner的卖点。详细可见CAP提出者Eric Brewer写的《Spanner, TrueTime和CAP理论》。

##### 完备性

两阶段提交协议是否严格支持ACID，各种异常场景是不是都可以覆盖？

2PC在commit阶段发送异常，其实跟最大努力一阶段提交类似也会有部分可见问题，严格讲一段时间内并不能保证A原子性和C一致性（待故障恢复后recovery机制可以保证最终的A和C）。  
 完备的分布式事务支持并不是一件简单的事情，需要可以应对网络以及各种硬件包括网卡、磁盘、CPU、内存、电源等各类异常，通过严格的测试。  
 之前跟某友商交流，他们甚至说目前已知的NewSQL在分布式事务支持上都是不完整的，他们都有案例跑不过，圈内人士这么笃定，也说明了**分布式事务的支持完整程度其实是层次不齐的**。  
 但分布式事务又是这些NewSQL数据库的一个非常重要的底层机制，跨资源的DML、DDL等都依赖其实现，如果这块的性能、完备性打折扣，上层跨分片SQL执行的正确性会受到很大影响。

##### 性能

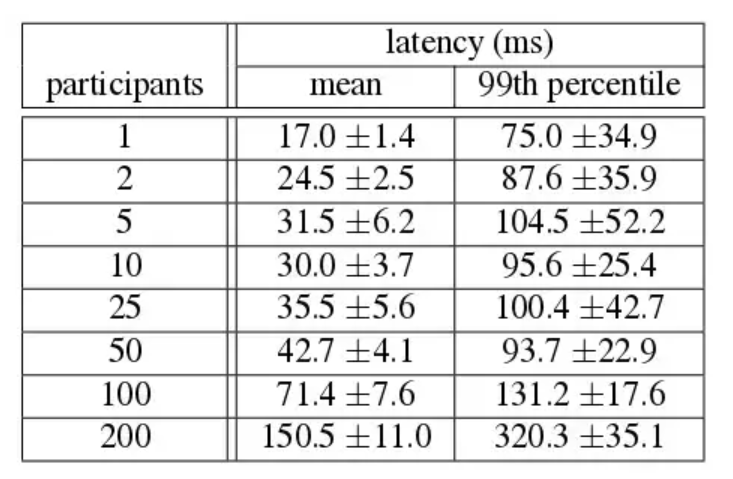
传统关系数据库也支持分布式事务XA，但为何很少有高并发场景下用呢？

因为XA的基础两阶段提交协议存在网络开销大，阻塞时间长、死锁等问题，这也导致了其实际上很少大规模用在基于传统关系数据库的OLTP系统中。

NewSQL数据库的分布式事务实现也仍然多基于两阶段提交协议，例如google percolator分布式事务模型，采用原子钟+MVCC+ Snapshot Isolation（SI）。  
这种方式通过TSO(Timestamp Oracle)保证了全局一致性，通过MVCC避免了锁，另外通过primary lock和secondary lock将提交的一部分转为异步，相比XA确实提高了分布式事务的性能。

SI是乐观锁，在热点数据场景，可能会大量的提交失败。另外SI的隔离级别与RR并无完全相同，它不会有幻想读，但会有写倾斜。

但不管如何优化，相比于1PC，2PC多出来的GID获取、网络开销、prepare日志持久化还是会带来很大的性能损失，尤其是跨节点的数量比较多时会更加显著  
 例如在银行场景做个批量扣款，一个文件可能上W个账户，这样的场景无论怎么做还是吞吐都不会很高。  
 Spanner给出的分布式事务测试数据：



虽然NewSQL分布式数据库产品都宣传完备支持分布式事务，但这并不是说应用可以完全不用关心数据拆分，这些数据库的最佳实践中仍然会写到，应用的大部分场景尽可能避免分布式事务。  
 既然强一致事务付出的性能代价太大，我们可以反思下是否真的需要这种强一致的分布式事务？  
 尤其是在做微服务拆分后，很多系统也不太可能放在一个统一的数据库中。尝试将一致性要求弱化，便是柔性事务，放弃ACID(Atomicity,Consistency, Isolation, Durability)，转投BASE(Basically Available,Soft state,Eventually consistent)。  
 例如Saga、TCC、可靠消息保证最终一致等模型，对于大规模高并发OLTP场景，我个人更建议使用柔性事务而非强一致的分布式事务。  
 关于柔性事务，笔者之前也写过一个技术组件，最近几年也涌现出了一些新的模型与框架（例如阿里刚开源的Fescar），限于篇幅不再赘述，有空再单独写篇文章。

解决分布式事务是否只能用两阶段提交协议？oceanbase1.0中通过updateserver避免分布式事务的思路很有启发性 ，不过2.0版后也变成了2PC。  
业界分布式事务也并非只有两阶段提交这一解，也有其它方案its-time-to-move-on-from-two-phase(如果打不开，国内有翻译版https://www.jdon.com/51588)

##### HA与异地多活

主从模式并不是最优的方式，就算是半同步复制，在极端情况下（半同步转异步）也存在丢数问题。

目前业界公认更好的方案是基于paxos分布式一致性协议或者其它类paxos如raft方式，Google Spanner、TiDB、cockcoachDB、OB都采用了这种方式，基于Paxos协议的多副本存储，遵循过半写原则，支持自动选主，解决了数据的高可靠，缩短了failover时间，提高了可用性，特别是减少了运维的工作量，这种方案技术上已经很成熟，也是NewSQL数据库底层的标配。

当然这种方式其实也可以用在传统关系数据库，阿里、微信团队等也有将MySQL存储改造支持paxos多副本的，MySQL也推出了官方版MySQL Group Cluster，预计不远的未来主从模式可能就成为历史了。

分布式一致性算法本身并不难，但具体在工程实践时，需要考虑很多异常并做很多优化，实现一个生产级可靠成熟的一致性协议并不容易。例如实际使用时必须转化实现为multi-paxos或multi-raft，需要通过batch、异步等方式减少网络、磁盘IO等开销。

需要注意的是很多NewSQL数据库厂商宣传基于paxos或raft协议可以实现【异地多活】，这个实际上是有前提的，那就是异地之间网络延迟不能太高。  
以银行“两地三中心”为例，异地之间多相隔数千里，延时达到数十毫秒，如果要多活，那便需异地副本也参与数据库日志过半确认，这样高的延时几乎没有OLTP系统可以接受的。

数据库层面做异地多活是个美好的愿景，但距离导致的延时目前并没有好的方案。

之前跟蚂蚁团队交流，蚂蚁异地多活的方案是在应用层通过MQ同步双写交易信息，异地DC将交易信息保存在分布式缓存中，一旦发生异地切换，数据库同步中间件会告之数据延迟时间，应用从缓存中读取交易信息，将这段时间内涉及到的业务对象例如用户、账户进行黑名单管理，等数据同步追上之后再将这些业务对象从黑名单中剔除。

由于双写的不是所有数据库操作日志而只是交易信息，数据延迟只影响一段时间内数据，这是目前我觉得比较靠谱的异地度多活方案。

另外有些系统进行了单元化改造，这在paxos选主时也要结合考虑进去，这也是目前很多NewSQL数据库欠缺的功能。

##### Scale横向扩展与分片机制

paxos算法解决了高可用、高可靠问题，并没有解决Scale横向扩展的问题，所以分片是必须支持的。NewSQL数据库都是天生内置分片机制的，而且会根据每个分片的数据负载(磁盘使用率、写入速度等)自动识别热点，然后进行分片的分裂、数据迁移、合并，这些过程应用是无感知的，这省去了DBA的很多运维工作量。以TiDB为例，它将数据切成region，如果region到64M时，数据自动进行迁移。

分库分表模式下需要应用设计之初就要明确各表的拆分键、拆分方式（range、取模、一致性哈希或者自定义路由表）、路由规则、拆分库表数量、扩容方式等。相比NewSQL数据库，这种模式给应用带来了很大侵入和复杂度，这对大多数系统来说也是一大挑战。

分库分表模式也能做到在线扩容，基本思路是通过异步复制先追加数据，然后设置只读完成路由切换，最后放开写操作，当然这些需要中间件与数据库端配合一起才能完成。

这里有个问题是NewSQL数据库统一的内置分片策略（例如tidb基于range）可能并不是最高效的，因为与领域模型中的划分要素并不一致，这导致的后果是很多交易会产生分布式事务。  
 举个例子，银行核心业务系统是以客户为维度，也就是说客户表、该客户的账户表、流水表在绝大部分场景下是一起写的，但如果按照各表主键range进行分片，这个交易并不能在一个分片上完成，这在高频OLTP系统中会带来性能问题。

##### 分布式SQL支持

常见的单分片SQL，这两者都能很好支持。NewSQL数据库由于定位与目标是一个通用的数据库，所以支持的SQL会更完整，包括跨分片的join、聚合等复杂SQL。中间件模式多面向应用需求设计，不过大部分也支持带拆分键SQL、库表遍历、单库join、聚合、排序、分页等。但对跨库的join以及聚合支持就不够了。  
 NewSQL数据库一般并不支持存储过程、视图、外键等功能，而中间件模式底层就是传统关系数据库，这些功能如果只是涉及单库是比较容易支持的。  
 NewSQL数据库往往选择兼容MySQL或者PostgreSQL协议，所以SQL支持仅局限于这两种，中间件例如驱动模式往往只需做简单的SQL解析、计算路由、SQL重写，所以可以支持更多种类的数据库SQL。  
 **SQL支持的差异主要在于分布式SQL执行计划生成器，由于NewSQL数据库具有底层数据的分布、统计信息，因此可以做CBO，生成的执行计划效率更高，而中间件模式下没有这些信息，往往只能基于规则RBO（Rule-Based-Opimization），这也是为什么中间件模式一般并不支持跨库join，因为实现了效率也往往并不高，还不如交给应用去做**。

注：GoldenDB分布式数据库中主要是基于RBO，但是由于有全局元数据管理节点的存在，所以也可以使用CBO进行优化，但是非常有限。

这里也可以看出中间件+分库分表模式的架构风格体现出的是一种妥协、平衡，它是一个面向应用型的设计；而NewSQL数据库则要求更高、“大包大揽”，它是一个通用底层技术软件，因此后者的复杂度、技术门槛也高很多。

##### 存储引擎

传统关系数据库的存储引擎设计都是面向磁盘的，大多都基于B+树。B+树通过降低树的高度减少随机读、进而减少磁盘寻道次数，提高读的性能，但大量的随机写会导致树的分裂，从而带来随机写，导致写性能下降。

NewSQL的底层存储引擎则多采用LSM，相比B+树LSM将对磁盘的随机写变成顺序写，大大提高了写的性能。

不过LSM的的读由于需要合并数据性能比B+树差，一般来说LSM更适合应在写大于读的场景。当然这只是单纯数据结构角度的对比，在数据库实际实现时还会通过SSD、缓冲、bloom filter等方式优化读写性能，所以读性能基本不会下降太多。

NewSQL数据由于多副本、分布式事务等开销，相比单机关系数据库SQL的响应时间并不占优，但由于集群的弹性扩展，整体QPS提升还是很明显的，这也是NewSQL数据库厂商说分布式数据库更看重的是吞吐，而不是单笔SQL响应时间的原因。

##### 成熟度与生态

分布式数据库是个新型通用底层软件，准确的衡量与评价需要一个多维度的测试模型，需包括发展现状、使用情况、社区生态、监控运维、周边配套工具、功能满足度、DBA人才、SQL兼容性、性能测试、高可用测试、在线扩容、分布式事务、隔离级别、在线DDL等等。

虽然NewSQL数据库发展经过了一定时间检验，但多集中在互联网以及传统企业非核心交易系统中，目前还处于快速迭代、规模使用不断优化完善的阶段。

相比而言，传统关系数据库则经过了多年的发展，通过完整的评测，在成熟度、功能、性能、周边生态、风险把控、相关人才积累等多方面都具有明显优势，同时对已建系统的兼容性也更好。  
 对于互联网公司，数据量的增长压力以及追求新技术的基因会更倾向于尝试NewSQL数据库，不用再考虑库表拆分、应用改造、扩容、事务一致性等问题怎么看都是非常吸引人的方案。  
 对于传统企业例如银行这种风险意识较高的行业来说，NewSQL数据库则可能在未来一段时间内仍处于探索、审慎试点的阶段。  
 基于中间件+分库分表模式架构简单，技术门槛更低，虽然没有NewSQL数据库功能全面，但大部分场景最核心的诉求也就是拆分后SQL的正确路由，而此功能中间件模式应对还是绰绰有余的，可以说在大多数OLTP场景是够用的。  
限于篇幅，其它特性例如在线DDL、数据迁移、运维工具等特性就不在本文展开对比。

##### 总结

如果看完以上内容，您还不知道选哪种模式，那么结合以下几个问题，先思考下NewSQL数据库解决的点对于自身是不是真正的痛点：

强一致事务是否必须在数据库层解决？

数据的增长速度是否不可预估的？

扩容的频率是否已超出了自身运维能力？

相比响应时间更看重吞吐？

是否必须做到对应用完全透明？

是否有熟悉NewSQL数据库的DBA团队？

如果以上有2到3个是肯定的，那么你可以考虑用NewSQL数据库了，虽然前期可能需要一定的学习成本，但它是数据库的发展方向，未来收益也会更高，尤其是互联网行业，随着数据量的突飞猛进，分库分表带来的痛苦会与日俱增。当然选择NewSQL数据库你也要做好承担一定风险的准备。  
如果你还未做出抉择，不妨再想想下面几个问题：

最终一致性是否可以满足实际场景？

数据未来几年的总量是否可以预估？

扩容、DDL等操作是否有系统维护窗口？

对响应时间是否比吞吐更敏感？

是否需要兼容已有的关系数据库系统？

是否已有传统数据库DBA人才的积累？

是否可容忍分库分表对应用的侵入？

如果这些问题有多数是肯定的，那还是分库分表吧。在软件领域很少有完美的解决方案，NewSQL数据库也不是数据分布式架构的银弹。  
相比而言分库分表是一个代价更低、风险更小的方案，它最大程度复用传统关系数据库生态，通过中间件也可以满足分库分表后的绝大多数功能，定制化能力更强。  
 在当前NewSQL数据库还未完全成熟的阶段，分库分表可以说是一个上限低但下限高的方案，尤其传统行业的核心系统，如果你仍然打算把数据库当做一个黑盒产品来用，踏踏实实用好分库分表会被认为是个稳妥的选择。

### Cloud Native DB

参考：

<https://zhuanlan.zhihu.com/p/87742609>

### 对比

首先关于“中间件+关系数据库分库分表”算不算NewSQL分布式数据库问题，国外有篇论文是pavlo-newsql-sigmodrec。

如果根据该文中的分类，Spanner、TiDB、OB算是第一种新架构型，Sharding-Sphere、Mycat、DRDS等中间件方案算是第二种（文中还有第三种云数据库，本文暂不详细介绍）。  
 基于中间件（包括SDK和Proxy两种形式）+传统关系数据库（分库分表）模式是不是分布式架构？

我觉得是的，因为存储确实也分布式了，也能实现横向扩展。但是不是"伪"分布式数据库？从架构先进性来看，这么说也有一定道理。

"伪"主要体现在中间件层与底层DB重复的SQL解析与执行计划生成、存储引擎基于B+Tree等，这在分布式数据库架构中实际上冗余低效的。

NewSQL数据库相比中间件+分库分表的先进在哪儿？画一个简单的架构对比图：



1. **传统数据库面向磁盘设计，基于内存的存储管理及并发控制，不如NewSQL数据库那般高效利用**；
2. **中间件模式SQL解析、执行计划优化等在中间件与数据库中重复工作，效率相比较低**；
3. **NewSQL数据库的分布式事务相比于XA进行了优化，性能更高**；
4. **新架构NewSQL数据库存储设计即为基于paxos（或Raft）协议的多副本，相比于传统数据库主从模式（半同步转异步后也存在丢数问题），在实现了真正的高可用、高可靠（RTO<30s，RPO=0）**；
5. NewSQL数据库天生支持数据分片，数据的迁移、扩容都是自动化的，大大减轻了DBA的工作，同时对应用透明，无需在SQL指定分库分表键。

注：分库分表和NewSQL架构主要区别在于索引结构、计算节点、分片方式、分布式事务、多副本管理、数据迁移。

## 共享介质

### 共享存储(主备)

### 共享存储/clusting(集群)

Share-memory：多个CPU共享同一片内存，CPU之间通过内部通讯机制进行通讯

Share-disk：每一个CPU使用自己的私有内存区域，通过内部通讯机制直接访问所有磁盘系统

Share memory体系结构的CPU之间通过主存进行通讯，具有很高的效率。但当更多的CPU被添加到主机上时，内存竞争contetion就成为瓶颈，CPU越多，瓶颈越厉害。Share-disk也存在同样的问题，因为磁盘系统由Interconnection Network连接在一起。

Share-memory和share-disk的基本问题是interference：当添加更多的CPU，系统反而减慢，因为增加了对内存访问（memory access）和网络带宽(network bandwidth)的竞争。这样shared noting体系得到了广泛的推广。

### Share-nothing

Share-noting：每一个CPU都有私有共享区域和私有磁盘空间，而且2个CPU不能访问相同磁盘空间，CPU之间的通讯通过网络连接。

Shared nothing体系是数据库稳定增长，当随着事务数量不断增加，增加额外的CPU和主存就可以保证每个事务处理时间不变。

总的来说，share nothing降低了竞争资源的等待时间，从而提高了性能。反过来，如果一个数据库应用系统要获得良好的可拓展的性能，它从设计和优化上就要考虑shared nothing体系结构。Share nothing neans few connection，它在oracle数据库设计和优化上有很多相同之处。

Share nothing对数据库应用主要体现在多用户并行访问系统时候，优化数据库操作的response time上。如果数据库操作能够顺利获得所需要的资源，不发生等待事件，自然可以减少response time，同时也体现在操作尽量少占用资源上，避免浪费时间在无用功上。

### Share-nothing certificate-based

# 架构

参考：《分布式数据库原理、架构与实践》 李海翔

## 分布式存储架构

### 数据分布

### 数据管理

### 多副本与数据存储

### 存算分离

### 多读与多写

## 分布式查询优化与并行执行架构

### 查询优化

### MPP

### 计算下推/外推

## 高可用架构

### 高可用事务

### 高可用架构设计

## 分布式事务架构

### 事务管理器在客户端、中间件、服务器端中的实现

### 去中心化的并发事务框架

## 可扩展性架构

### 可扩展性能力

### 事务处理的可扩展性

#### 并发算法层面

#### 事务任务执行层面

#### 数据库架构层面

## 强一致性

对于分布式事务性数据库而言，强一致性融合了分布式强一致性与事务一致性，即其做到了严格可串行化（Strict Serializability）或者顺序可串行化（Sequential Serializability）。

目前市场可见的分布式数据库中，能做到严格可串行化的是Spanner，能做到顺序可串行化的是CockroachDB（宣称可做到，但目前版本实现的是因果可串行化），腾讯的TDSQL数据库则支持可串行化和顺序可串行化。

### 一致性问题

### 一致性问题解法

## 解耦

# 原理

## 存储引擎

## SQL引擎

## SQL调优

## 分布式数据库同步时钟

## 分布式事务

## 路由

## 数据迁移

## 备份恢复

## 监控告警

## 故障排查

# 特性

数据库最关注的几个特性：一致性、扩展性、高性能和高可用。

## 一致性

TDSQL

GoldenDB

PolarDB

OceanBase

TiDB

## 扩展性

TDSQL

GoldenDB

PolarDB

OceanBase

TiDB

## 高性能

TDSQL

GoldenDB

PolarDB

OceanBase

TiDB

## 高可用

# 发展方向

## 元数据库

## HTAP

## 高速存储硬件