24 | 现状解读：分布式数据库的最新发展情况

你好，恭喜你坚持到了课程的最后一讲。

上一讲，我们探讨了实现数据库中间件的几种技术，包括全局唯一主键、分片策略和跨分片查询，其中最重要的就是分布式事务，希望你可以掌握它。这一讲作为收尾，我将为你介绍 NewSQL 数据库。

首先我试着去定义 NewSQL，它是一类现代的关系型数据库，同时它又具备 NoSQL 的扩展能力。其擅长在 OLTP 场景下提供高性能的读写服务，同时可以保障事务隔离性和数据一致性。我们可以简单理解为，NewSQL 要将 2000 年左右发展而来的 NoSQL 所代表的扩展性与 20 世纪 70 年代发展的关系模型 SQL 和 ACID 事务进行结合，从而获得一个高并发关系型的分布式数据库。

如果我们使用 NewSQL 数据库，可以使用熟悉的 SQL 来与数据库进行交互。SQL 的优势我在模块一中已经有了深入的介绍。使用 SQL 使得原有基于 SQL 的应用不需要改造（或进行微量改造）就可以直接从传统关系型数据库切换到 NewSQL 数据库。而与之相对，NoSQL 数据库一般使用 SQL 变种语言或者定制的 API，那么用户切换到 NoSQL 数据库将会面临比较高的代价。

对于 NewSQL 的定义和适用范围一直存在争议。有人认为 Vertica、Greenplum 等面向 OLAP 且具有分布式特点的数据库也应该归到 NewSQL 里面。但是，业界更加广泛接受的 NewSQL 标准包括：

1. 执行短的读写事务，也就是不能出现阻塞的事务操作；
2. 使用索引去查询一部分数据集，不存在加载数据表中的全部数据进行分析；
3. 采用 Sharded-Nothing 架构；
4. 无锁的高并发事务。

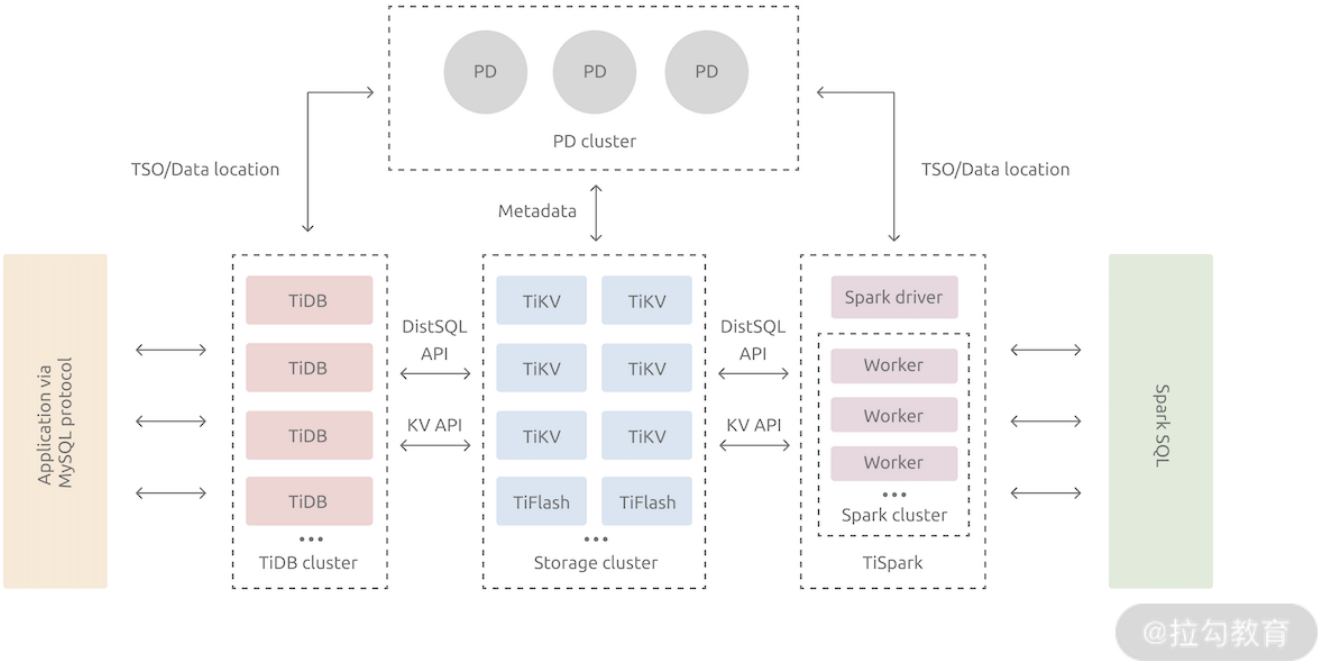
根据以上这些特点，我总结为：**一个 NewSQL 数据库是采用创新架构，透明支持 Sharding，具有高并事务的 SQL 关系型数据库**。

> 请注意 DistributedSQL 是一类特殊的 NewSQL，它们可以进行全球部署。

下面就按照我给出的定义中的关键点来向你详细介绍 NewSQL 数据库。

创新的架构

使用创新的数据库架构是 NewSQL 数据库非常引人注目的特性。这种新架构一般不会依靠任何遗留的代码，这与我在“22 | 发展与局限：传统数据库在分布式领域的探索”中介绍的依赖传统数据库作为计算存储节点非常不同。我们以 TiDB 这个典型的 NewSQL 数据库为例。



可以看到其中的创新点有以下几个。

1. 存储引擎没有使用传统数据库。而使用的是新型基于 LSM 的 KV 分布式存储引擎，有些数据库使用了完全内存形式的存储引擎，比如 NuoDB。
2. Sharded-Nothing 架构。底层存储到上层工作负载都是独立部署的。
3. 高性能并发事务。TiDB 实现了基于 Percolator 算法的高性能乐观事务。
4. 透明分片。TiDB 实现了自动的范围分片，可以弹性地增减节点。
5. 基于复制技术的自动容灾。使用 Raft 算法实现高可用的数据复制，自动进行故障转移。

可以说，NewSQL 与传统关系型数据库之间的交集在于 SQL 与 ACID 事务，从而保证用户的使用习惯得以延续。

以上描述的创新点我在前两个模块都有详细的说明。这里不知道你是否注意到一个问题：与使用传统数据库构建分布式数据库相比，NewSQL 最为明显的差异来自存储引擎。特别是以 Spanner 为首的一众 NewSQL 数据库，如 YugaByte DB、CockroachDB 和 TiDB 都是使用 LSM 树作为存储引擎，并且都是 KV 结构。如此选择的原理是什么呢？

我在介绍 LSM 的时候，提到其可以高性能地写入与读取，但是牺牲了空间。可能你就据此得出结论，NewSQL 数据库面对 OLTP 场景，希望得到吞吐量的提升，故选择 LSM 树存储引擎，而不选择 B 树类的存储引擎。但是，我也介绍过有很多方法可以改进 B 树的吞吐量。所以这一点并不是关键点。

我曾经也会困惑于这个问题，经过大量研究，并与项目组人员交流。从而得到了一个“结论”：**开源的 NewSQL 选择的并不是 LSM 树，而是 RocksDB**。选择 RocksDB 难道不是因为它是 LSM 结构的？答案是否定的。大部分以 RocksDB 为存储引擎的开源 NewSQL 数据库看中的是 RocksDB 的性能与功能。可以有一个合理的推论，如果有一款在性能和功能上碾压 RocksDB 的 B 树存储引擎，那么当代开源 NewSQL 数据库的存储引擎版图又会是另一番景象了。

你不用诧异，这种发展趋势其实代表了 IT 技术的一种实用特性。从 TCP/IP 协议的普及，到 Java 企业领域 Spring 替代 EJB，都体现了这种实用性。那就是真正胜利的技术是一定有实用价值的，这种实用价值要胜过任何完美的理论。这也启发我们在观察一个分布式数据库时，不要着急给它分类，因为今天我要介绍的评判 NewSQL 的标准，是基于现有数据库特性的总结，并不能代表未来的发展。我们要学会掌握每种数据库核心特性。

到这里，我还要提一个比较特别的数据库，那就是 OceanBase。OceanBase 的读写跟传统数据库有很大的一点不同就是：OceanBase 的写并不是直接在数据块上修改，而是新开辟一块增量内存用于存放数据的变化。同一笔记录多次变化后，增量块会以链表形式组织在一起，这些增量修改会一直在内存里不落盘。OceanBase 读则是要把最早读入内存的数据块加上后续相关的增量块内容合并读出。这种特点其实与 LSM 树的内存表和数据表有类似之处，只是 OceanBase 在更高维度上。

总结完架构上的创新，下面我将介绍 NewSQL 中关于分片的管理。

透明的 Sharding

我们上文介绍过数据库中间件是如何进行 Sharding 的，那就是使用逻辑的分片键来进行分片计算，将不同的行写入到不同的目标数据库。其中，需要用户深度地参与。比如，需要用户去指定哪个键为分片键，逻辑表与物理表的映射规则，必要的时候还需要进行 SQL 改造，等等。故中间件模式的 Sharding 我们一般称作显示 Sharding，与之相对的就是 NewSQL 数据库提供的透明 Sharding。

透明 Sharding 顾名思义，用户不需要去指定任何的规则，数据就能分散到整个集群中，并自动做了备份处理。那么 NewSQL 是怎么确定分片键的呢？我们以 TiDB 为例。

一个表中的数据是按照 Region 来进行 Sharding 的。每个 Region 有多个副本，从而保障了数据的高可用。不同的表将会有不同的 Region，而不是如传统分库那样每个库里的表都是相同的。 那么一个表下，每一行数据存储在哪个 Region 下是如何确定的呢?

首先要确定的是行是怎么映射到 KV 结构的： key 由 table_id 表 id+rowid 主键组成。如：

```
t[table_id]_r[row_id]
```

而 value 保存的就是一整行的数据。那么我们就使用这个 key 来计算这行数据应该落在哪个 Region 里面。

TiDB 是使用范围策略来划分数据。有索引情况下，负责索引的 Region 会根据索引字段范围来划分。基于 key 经过一个转换，将会得出一个数字，然后按范围划分多个区间，每个区间由一个 Region 管理。范围分片的好处我们之前介绍过，就是存储平衡和访问压力的平衡。其原因是，范围分片有机会做更好的动态调度，只有动态了，才能实时自动适应各种动态的场景。

虽然透明 Sharding 为用户带来了使用的便利。但聪明的你可以注意到了，这种隐式的分片方案相比上一讲介绍的方案，从功能上讲是存在缺失的。最明显的是它缺少跨分片 Join 的功能。可以想到，此种 NewSQL 与具有 ER 分片的数据库中间件在性能上是存在差异的。针对此种情况，TiDB 引入了 Placement Rules 来制定将相关联的数据表放在同一个 KV 存储中，从而达到 ER 分片的效果。

当然，随着 NewSQL 的逐步发展，哈希分片也逐步被引入 NewSQL 中。我们知道范围分片有热点问题，虽然可以通过动态拆分合并分片来缓解，但终究不是从根本上解决该问题。特别是对于具有强顺序的数据，比如时间序列，该问题就会变得很突出。而哈希分片就是应对该问题的有效手段。基于这个原因，Cockroach DB 引入哈希分片索引来实现针对序列化数据的扩展能力。

我们小结一下，透明的 Sharding 针对的往往是主键，一般会选择行 ID 作为主键。而如果要实现功能完善的 Sharding，一些用户参与的配置操作还是有必要的。

解决了 Sharding 问题，让我们看看 SQL 层面需要解决的问题吧。

分布式的 SQL

NewSQL 数据库相对于 NoSQL 最强大的优势还是对 SQL 的支持。我曾经在模块一中与你讨论过，SQL 是成功的分布式数据库一个必要的功能。SQL 的重要性我们已经讨论过了，但实现 SQL 一直是被认为很困难的，究其原因主要来源于以下两个方面。

1. SQL 的非标准性。虽然我们有 SQL99 这种事实标准，但是在工业界，没有一种流行的数据库完全使用标准去构建 SQL。每家都有自己的方言、自己独有的特性，故我们看到的 NewSQL 数据库大部分都会按照已经存在的数据库方言去实现 SQL 语义，比如 TiDB 实现了 MySQL 的语法，而 CockroachDB 实现了 PostgreSQL。
2. 声明式语言。SQL 语言是一种更高级的语言，比我们熟悉的 Java、Go 等都要高级。主要体现为它表述了希望得到什么结果，而没有指示数据库怎么做。这就引出了所谓的执行计划优化等概念，为实现高效的查询引擎带来了挑战。

以上是传统数据库实现 SQL 的挑战，而对于分布式数据库来说还需要将数据分散带来的问题考虑进去，同时在查询优化方面要将网络延迟作为一个重要因素来考量。

对于 NewSQL 而言，如果使用了上面所描述的创新架构，特别是底层使用 KV 存储，那么就需要考虑数据与索引如何映射到底层 KV 上。我在前面已经说明了数据是如何映射到 KV 上的，那就是 key 由 table_id 表 id+rowid 主键组成，value 存放行数据：

```
t[table_id]_r[row_id] => row_data
```

而对于索引，我们就要区分唯一索引和非唯一索引。

对于唯一索引，我们将 table_id、index_id 和 index_value 组成 key，value 为上面的 row_id。

```
t[table_id]_i[index_id][index_value] => row_id
```

对于非唯一索引，就需要将 row_id 放到 key 中，而 value 是空的。

```
t[table_id]_i[index_id][index_value]_r[row_id] => null
```

底层映射问题解决后，就需要进入执行层面了。这里面牵扯大量的技术细节，我不会深入其中，而是为你展示一些 NewSQL 数据库会面临的挑战。

首先是正确性问题。是的，要实现正确的 SQL 语法本身就是挑战。因为 SQL 可以接受用户自定义表和索引，查询使用名字，而执行使用的是 ID，这之间的映射关系会给正确性带来挑战。而其中最不好处理的就是 Null，不仅仅是 NewSQL，传统数据库在处理 Null 的时候也经过了长时间的“折磨”。这也是为什么一个 SQL 类数据库需要经过长时间的迭代才能达到稳定状态的原因。

然后就是性能了。性能问题不仅仅是数据库开发人员，DBA 和最终用户也对它非常关注。如果你经常与数据库打交道，一定对 SQL 优化有一定了解。这背后的原因其实就是 SQL 声明式语言导致的。

大部分 NewSQL 数据库都需要实现传统数据库的优化手段，这一般分为两类：基于规则和基于代价。前者是根据 SQL 语义与数据库特点进行的静态分析，也称为逻辑优化，常见有列裁剪、子查询优化为连接、谓词下推、TopN 下推，等等；而基于代价的优化，需要数据库实时产生统计信息，根据这些信息来决定 SQL 是否查询索引、执行的先后顺序，等等。

而具有分布式数据库特色的优化包括并行执行、流式聚合，基于代价的优化需要考虑网络因素等。故实现分布式数据库的 SQL 优化要更为复杂。

从上文可以看到，NewSQL 的数据库实现 SQL 层所面临的挑战要远远大于传统数据库。但由于 SQL 执行与优化技术经过多年的积累，现在已经形成了完整的体系。故新一代 NewSQL 数据库可以在此基础上去构建，它们站在巨人的肩膀上，实现完整的 SQL 功能将会事半功倍。

介绍完了 NewSQL 数据库在 SQL 领域的探索，最后我们来介绍其高性能事务的特点。

高性能事务

NewSQL 的事务是其能够得到广泛应用的关键。在上一个模块中我们讨论的 Spanner 与 Calvin，就是典型的创新事务之争。这一讲我将总结几种 NewSQL 数据库处理事务的常见模式，并结合一定的案例来为你说明。

首先的一种分类方式就是集中化事务管理与去中心化事务管理。前者的优势是很容易实现事务的隔离，特别是实现序列化隔离，代价就是其吞吐量往往偏低；而后者适合构建高并发事务，但是需要逻辑时钟来进行授时服务，并保证操作竞争资源的正确性。

以上是比较常规的认识，但是我们介绍过 Calvin 事务模型。它其实是一种集中化的事务处理模式，但却在高竞争环境下具有非常明显的吞吐量的优势。其关键就是通过重新调度事务的执行，消除了竞争，从而提高了吞吐量。采用该模式的除了 Calvin 外，还有 VoltDB。其原理和 Calvin 类似，感兴趣的话你可以自行学习。

而采用去中心化事务的方案一般需要结合：MVCC、2PC 和 PaxosGroup。将它们结合，可以实现序列化的快照隔离，并保证执行过程中各个组件的高可用。采用 PaxosGroup 除了提供高可用性外，一个重要的功能就是将数据进行分区，从而降低获取锁的竞争，达到提高并发事务效率的目的。这种模式首先被 Spanner 所引入，故我们一般称其为 Spanner 类事务模型。

而上一讲我们介绍了京东为 ShardingShpere 打造的 JDTX 又是另一番景象，数据库中间件由于无法操作数据库底层，所以事务方案就被锁死。而 JDTX 采用类似 OceanBase 的模式，首先在数据库节点之外构建了一个独立的 MVCC 引擎，查询最新的数据需要结合数据库节点与 MVCC 引擎中修改的记录，从而获得最新数据。而 MVCC 中的数据会异步落盘，从而保证数据被释放。JDTX 的模式打破了中间件无法实现高性能事务模型的诅咒，为我们打开了思路。

可以看到，目前 NewSQL 的并发事务处理技术往往使用多种经过广泛验证的方案。可以将它比喻为当代的航空母舰，虽然每个部件都没有创新点，但是将它们结合起来却实现了前人无法企及的成就。

总结

这一讲，我介绍了 NewSQL 数据库的定义，并为你详细分析了一个 NewSQL 数据库的关键点，即以下四个。

1. 创新的架构：分布式系统与存储引擎都需要创新。
2. 透明的 Sharding：自动的控制，解放用户，贴合云原生。
3. 分布式 SQL：打造商业可用分布式数据库的关键。
4. 高性能事务：NewSQL 创新基地，是区别不同种类 NewSQL 的关键。

至此，我们课程的主要内容就已经全部介绍完了。

NewSQL 和具有全球部署能力的 DistributedSQL 是当代分布式数据库的发展方向，可以说我们介绍过的所有知识都是围绕在 NewSQL 体系内的。接下来在加餐环节，我会为你介绍其他种类的分布式数据库，以帮助你拓展思路。

感谢学习，希望我的分享会对你带来帮助。