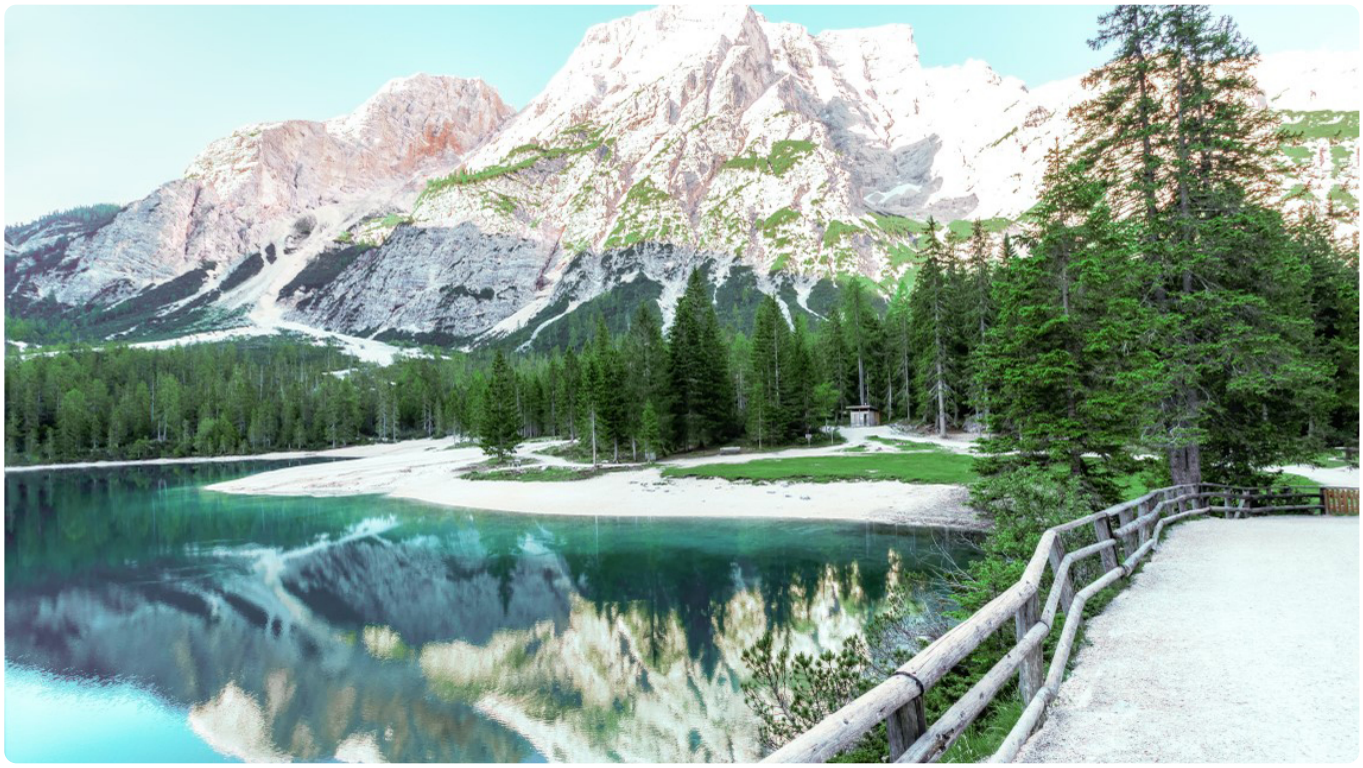
18 | 正确性案例（中）：常见分布式数据方案的设计原理是什么？

2021-02-03 任杰

分布式金融架构课 进入课程 >



讲述：任杰
时长 21:58 大小 20.12M

你好，我是任杰。这一讲我想和你聊一聊常见的分布式数据系统的设计原理。

所有的业务系统归根到底都需要处理数据，因此从本质上来讲都是数据系统。业务系统和一般数据系统只是在处理数据的逻辑上有所不同，它们对于数据的存储、读取、容灾等都有极大的相似之处。

因此，我希望在学完这节课之后，你既能了解常用数据系统的运作原理，更好地使用它们，同时也能举一反三，在今后设计金融系统架构的时候能借鉴一些思路。

大部分的原理我在前面的课已经讲过了，关键之处我还会再次提示，如果有不清楚的内容，你可以回到对应的文章去复习一下。

Redis

我们先从最简单的 K/V 存储开始。Redis 出现之后，一举取代了 Memcached，成为首选的基于内存的 K/V 解决方案。Redis 的核心竞争力是速度快，那我们就来分析一下，为什么 Redis 会拥有速度上的优势呢？

首先，我们看看 Redis 处理数据的方式。**Redis 默认用单线程处理所有数据。**

单线程是一种能优化延时的解决方案，不过单线程虽然适合处理数据，但是不一定适合 I/O。同一时刻可能会有多个客户端在访问 Redis，如果用多线程处理的话，就会出现多线程造成的加锁冲突。

这时候，**Redis 用了我们在第 11 节课讲网络优化时提到的 epoll 方法**，用较少的计算资源来支持大量的 I/O 并发。

我们再来看看 Redis 的容灾和高可用。**Redis 默认的容灾采用了主从备份的方法。**主节点将内容异步复制给从节点。从节点默认是只读，所有从节点的写操作会失败，之所以这么做，主要是为了简化一些复杂的 Redis 使用场景，比如处理数据过期的问题。

有了 Redis 的异步主从备份，主节点就能更快地返回消息给客户端。同时，如果你想让主节点运行得更快，还可以取消主节点的本地备份功能。这样主节点不需要写本地文件，处理的速度会更快。

Redis 这种异步主从备份的方式极大减少了处理的延时。但是按照我们在第 14 节课讲的会话一致性分析，**异步备份和只读从节点的方式无法满足单调读一致和自读自写的一致性要求。**

如果为了**更高的一致性要求，需要读操作也在主节点发生，这时候就能满足线性一致性。**但是要注意，**这个线性一致性假设主节点不能出问题。**如果主节点出了问题的话，异步备份可能会丢失数据，所以整个集群依然不是线性一致性的。

接下来，我们再看看本地数据备份的两个优化选择。主节点可以选择将一部分数据保存在本地，之后可以用这些数据来恢复单机状态。**一个选择是 RDB，Redis 会定期将内部状态保存到本地硬盘；另一个选择是 AOF，Redis 会将操作日志实时保存到本地硬盘。**

我们比较一下 RDB 和 AOF 在速度上的优缺点。AOF 需要一直写文件，而 RDB 只需要偶尔写文件，所以 **RDB 写入数据量小，频率低，因此速度会更快一些。但是 RDB 牺牲了数据完整性，两次 RDB 之间的数据无法恢复。**

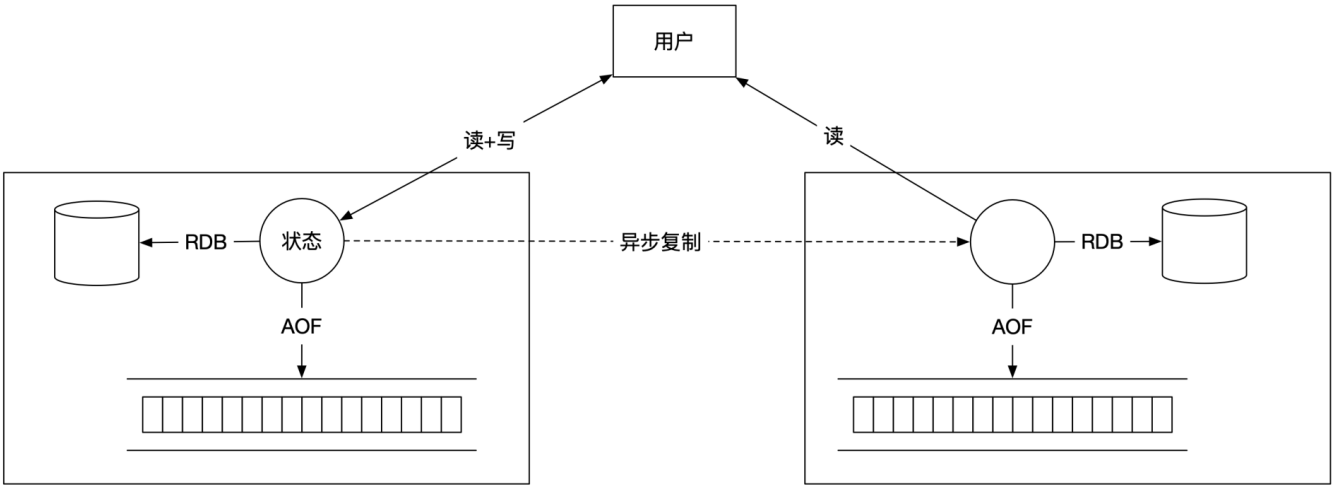
最后，我们来总结一下 Redis 关于速度上的一些优化思路：

1. 用单线程和 epoll 来处理数据。

2. 异步容灾，通过牺牲单调读一致和自读自写，换取消息返回速度。

3. 用 RDB 来实现定期的状态备份，通过牺牲数据完整性来换取处理速度。

Redis 的一个简单架构图如下：



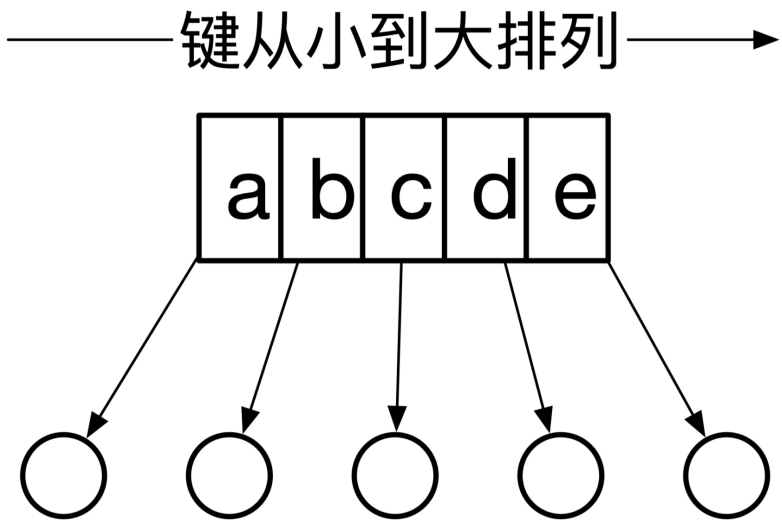
RocksDB

数据系统一般有三件事情要处理，分别是数据查询，数据保存和数据处理，也就是数据的读写和计算。Redis 优化的是数据的读和计算，接下来我们来看看，RocksDB 怎么来优化数据的写。

RocksDB 和我后面要讲的 Spanner、TiDB 一样，都属于同一类 K/V 的实现。这些实现都基于 SSTable 和 LSM 树。

SSTable 的全称是 Sorted String Table。其实 **SSTable 就是一个存储在文件的 Map**，这个 Map 的键的类型是字符串，在存储的时候这些键是按从小到大的顺序排列的。下面这幅

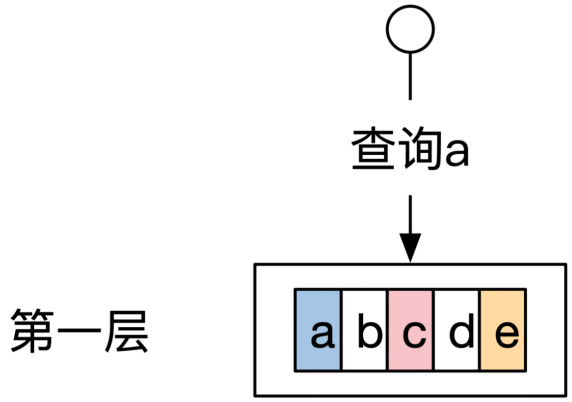
图是 SSTable 的一个简单例子：

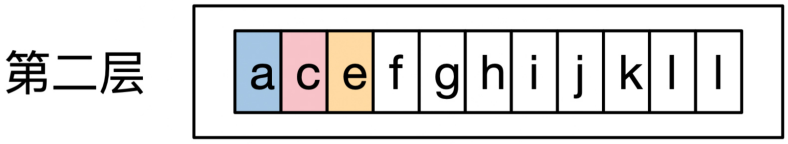


你也许觉得单个 SSTable 没有什么特别之处，但是多个 SSTable 放在一起之后会，就出现一个特殊的数据结构，它就是 LSM 树。

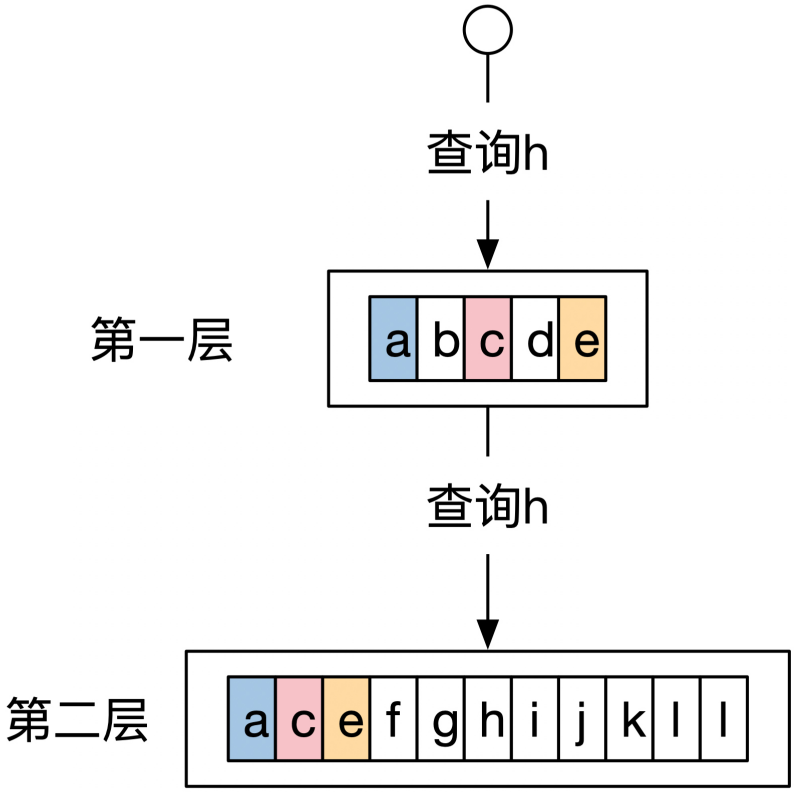
LSM 树用多个 SSTable 实现了一个 Map。LSM 树将所有 SSTable 从上到下排列，查询的时候先查最上面的 SSTable，如果查不到就查下一层，以此类推。

我还是结合一个例子给你讲解。LSM 树一共有两层。这两层有 3 个键是重复的，分别是 a、b 和 c。当我们想查询 a 的时候，在第一层就发现了数据，因此不需要访问第二层的 SSTable。





但是如果我们换一下查询的条件。这时候查询的是 h，而不是 a。第一层不包含键为 h 的数据，因此查询会继续访问第二层。下面这幅图展示了这个查询过程：



那 LSM 树的这个架构有什么优势呢？前面提到过，这种架构的写入速度非常快。其实写入速度无论多快，肯定也比不上 Redis，因为 Redis 可以选择不写文件。所以 **LSM 树的真正优势是当 K/V 数据量超过内存大小时，基于文件系统的 LSM 树结构提供了基本的 K/V 查询能力，同时它还具有很高的数据写入速度。**

传统的基于文件的 K/V 结构是 B+ 树。B+ 树在更新数据的时候需要对文件做修改。LSM 树在更新数据的时候，并不会对已有的 SSTable 进行修改，而是在所有的 SSTable 之上再建一个新的 SSTable。

这种方式和 MVCC 的方法类似（MVCC 的内容详见第 12 节课），都是**将一个修改操作变成了新增操作**，这样对文件的所有操作都是在末尾添加新的内容。

第 7 节课讲事件溯源架构的时候，我们说过这种在文件末尾添加内容的操作方式，它能最大程度地使用硬盘提供的写入能力，所以写入速度会很快。

但是 **LSM 树也有一个缺点，就是它的查询速度慢**，这是因为随着时间的推移，系统可能有很多层 SSTable。当你查询一些更新频率不高的数据时，很有可能需要读很多层 SSTable 之后才能知道自己需要的值。

尽管我们有一些优化的方式，比如使用 bloomfilter 来做缓存，但是依然无法完全解决。因此，从监控上可以看到，LSM 树的查询延时像一个锯齿，很多时候延时都很低，但是偶尔会有几秒甚至更高的延时。B+ 树就没有这个问题，延时都很平稳。

所以，如果你对读写的速度要求不高，但是希望延时可控，那么你需要选择 B+。如果你对查询的延时要求不高，但是对写入速度要求很高，那么你需要选择 LSM 树。

那 RocksDB 这些基于 LSM 树的 K/V 存储对于分布式系统有什么帮助呢？

我们在这节课的最开始，提到了 Redis 的 K/V 存储和它默认的异步同步机制。

如果将 Redis 的 RDS 快照实现方式换成 LSM 树，那么这个快照就可以实时生成，并且时间开销小。这样我们就有了一个具有一定异步容灾能力的 K/V 集群，这个集群能绕开内存大小的限制，用更大的硬盘来提供存储能力。

这就是基于内存和基于文件的两种不同的 K/V 集群实现方式。这些方式依然有一些问题，那就是没法实现会话一致性。按照我们前面介绍的思路，如果想要有所得就需要有所失。

如果我们想要有更高的正确性，那么就需要牺牲一些速度。Google 的 Spanner 数据库基本上是这方面的开山鼻祖。

Spanner

Spanner 是 Google 在 2012 年公布出来的一个全球性分布式关系型数据库，它的横空出世惊艳了所有人，里面有很多神奇的数据解决方案。我们在平时可能就接触过这些架构和思路，但是 Spanner 是第一个将这些内容整合在一起的系统，并且它证明了这个方案是可行的。

有了 Spanner 开路，接下来开源领域就纷纷放手一搏，将 Spanner 的论文吸收引进，消化以后再创新，出现了很多一致性分布式数据方案，这些方案普通人也能使用。

Spanner 将传统意义上的单机版关系型数据库按照组件拆分了出来。传统数据库用顺序增长的日志文件来存储数据库的操作，这个日志文件可以用来恢复数据库的状态。既然日志文件可以用来恢复本地数据库，那么它也可以用来恢复其他机器上的数据库，而且如果日志文件同步得足够快，那么其他数据库有可能实现实时同步。

但是，跨机器日志文件同步会碰到分布式系统的一致性问题。我在第 16 节课说过，如果在分布式环境下将文件正确地同步到多台机器上，这时候我们需要用共识算法来达到全序广播的能力。

在 2012 年的时候，Raft 共识算法还没有设计出来，所以 Google 在当时用了另一种叫 **Paxos** 的共识算法。

算法虽然不一样，但它们都有一样的共识能力。Spanner 会先通过 Paxos 算法将日志文件同步到多台机器，然后每台机器通过日志文件实时同步彼此的状态（这个方式第 17 节课最后提到过）。

对于数据库来说，它的状态其实就是数据库表。我们学习 KDB 列数据库（内容详见第 10 节课）的时候说过，关系型数据库其实就是个 Map，也就是 K/V 结构。它的键是列的名字，它的值是每一列的数据。这是一种将数据库表垂直划分的方式。

还有一种是水平划分的方式，划分的结果也是 K/V 结构。数据库表一般都有主键，这些主键就是 Map 的键，每一行内容就是 Map 的值。**因此无论哪种划分方式，数据库表最后的存储都是 K/V 结构。**

既然数据库状态是一个 K/V，那么只要我们在日志文件中记录了 K/V 的插入和修改操作，就可以用共识算法来实现分布式状态机，这样就能实现多个数据库的实时状态同步了。

数据库除了要生成关系型表之外，还需要支持很多重要的操作，比如数据库事务。

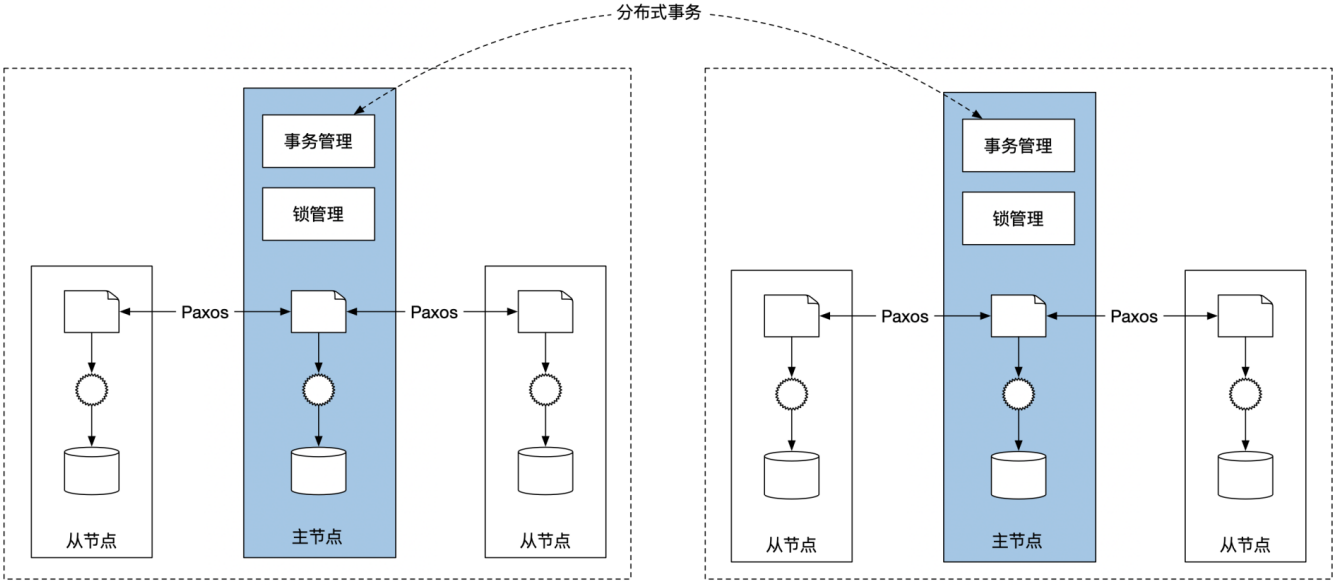
数据库事务的实现需要给数据加锁，这就意味着**分布式环境下数据库事务的实现需要分布式锁。**到目前为止，我们只说过一种实现分布式锁的方式，那就是用共识算法实现线性化存储，然后用线性化存储实现分布式锁。这里的内容如果你记不清了，可以回顾第 16 节课的内容。

Paxos 算法的节点也有主从之分**。共识算法的主节点会承担起实现分布式锁的责任，同时它也负责维护集群的事务管理。**如果事务需要横跨多个不同的 Spanner 组，那么每个组的主节点会互相沟通，选一个主节点作为分布式事务的协调者，用两阶段提交的方式来实现分布式事务。

通过前面的学习，我们知道了两阶段提交需要有一个协调者，这个协调者会成为分布式事务的单点。Spanner 在设计的时候，分布式事务的协调者其实是一个 Paxos 算法的主节点，因此它本身就有一定的容灾能力，不再是单点。

其实我们在上节课学习分布式的事件溯源架构的时候，采用的也是同一个思路。共识算法的主节点承担所有的写入操作和对外沟通工作，其他节点负责同步主节点的修改行为。

下图展示了简化版的 Spanner 架构图：



Spanner 还有另一个著名的优化技巧，那就是用原子钟实现的 TrueTime API，它用物理的方式优化了事务的实现。这个方法的普适性不高，所以在这里不再详细介绍，如果你有兴趣可以查看 Spanner 的论文。

在这里我要特别提到的一点是，虽然现在看起来 Spanner 的设计比较平常，用的都是我们常见的数据系统解决方案，但是不要忽略先后顺序。现在的数据系统用到的成熟解决方案，它们普遍来自于 Spanner，其实我们只是站在了巨人的肩上而已，饮水不忘打井人。

TiDB

江山代有才人出，了解了 Spanner，我们再看看国内的代表方案。TiDB 是国产分布式数据库的领军人物之一，它在 Spanner 的基础上有一些功能的加强。那接下来，我们来看看都是哪些架构优化的思路。

在设计 Spanner 的时候，只有 Paxos 这一个共识算法。但当 TiDB 出现的时候，已经有了 Raft 算法，所以可以用新的算法实现日志文件的全序广播。Raft 算法相对于 Paxos 来说更清晰易懂，从工程的角度讲能节省不少的研发时间。

讲 Spanner 的时候我们已经说了，数据库表其实就是一个 K/V。TiDB 将这个概念直接具现化，抽象出了一个 TiKV 的组件，这个组件通过 Raft 共识算法来实现分布式 K/V。你也可以直接使用 TiKV。对于 TiDB 来说，TiKV 存储了数据库的每一行的信息。

关系型数据库在设计之初是为了解决业务的事务问题，也就是解决每一行应该怎么操作。这样的优化结果却不方便我们分析数据，所以才有了用列存储方案的列数据库。

TiDB 在设计的时候，将自己定位为既能解决事务问题，又能解决分析问题的数据库。那么按照我们之前的介绍，TiDB 需要有一个列数据库，这样才能优化数据的分析。那这个列数据库应该怎么实现呢？

这里我来给你讲讲这个架构的分析思路。我们在第 7 节课介绍事件溯源架构的时候，提到过一个概念叫做 CQRS，也就是读写分离。读写分离的一个好处是可以将读和写拆分出来分别优化。

对于 TiDB 的设计目标来说，事务处理是写的部分，可以用基于行存储的 TiDB 来实现。TiDB 的另一个设计目标是数据分析，而分析显然是读操作，因此可以用基于列存储的解决方案来实现，这就是 TiDB 的另一个组件 TiFlash。

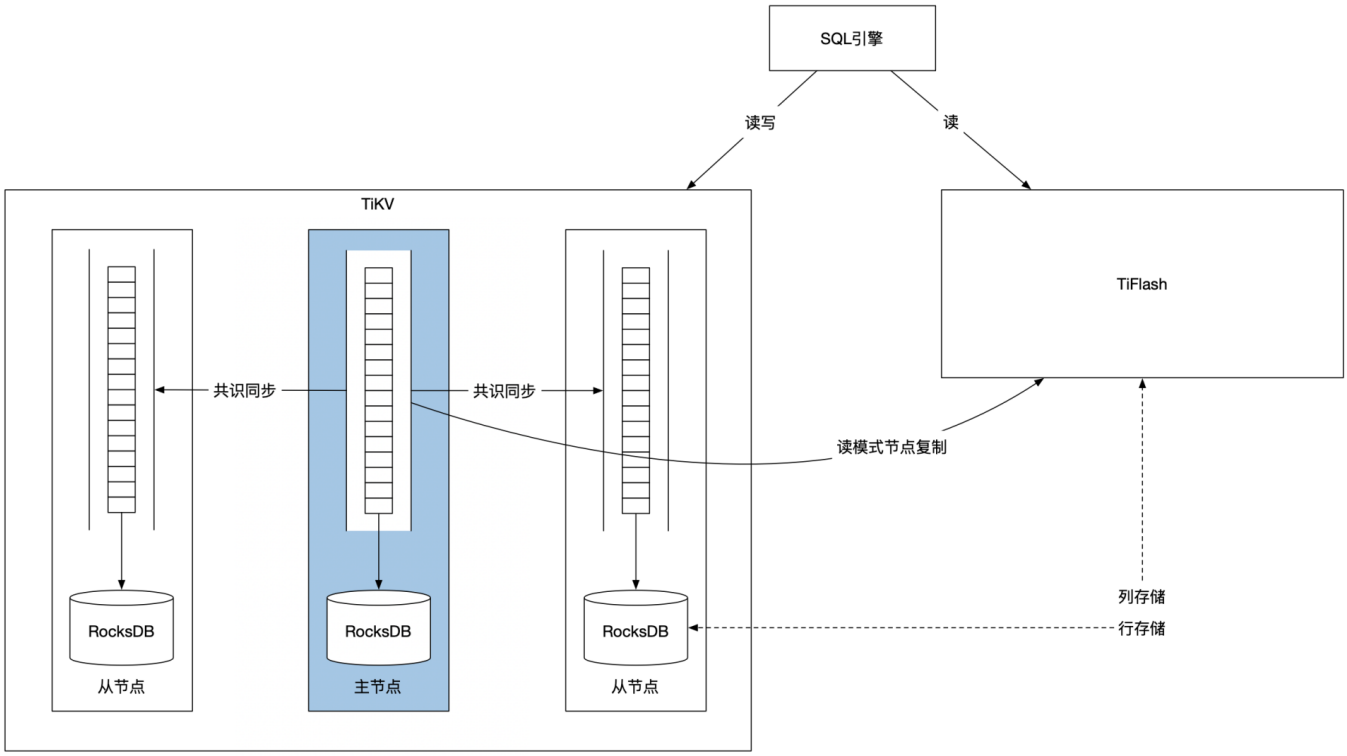
所以在 TiDB 中，TiKV 负责用共识算法实现数据库表的事务功能，TiFlash 负责用列存储的方式实现数据查询功能。那么 TiFlash 需要用到共识算法吗？

为了回答这个问题，你可以回忆一下第 17 节课讲分布式事件溯源架构时，我们提到的两种数据查询方式，分布式环境下的查询包括常规查询和一致性读两种。

数据分析不需要有一致性读的一致性能力，因此有常规查询的支持就足够了。**常规查询需要用到读模式的状态机**，因此 TiFlash 可以从 TiKV 的节点上复制日志文件。TiFlash 选择只从主节点复制，主要是为了节省延时。

TiKV 的存储方式和 RocksDB 类似，也用的是 LSM 树，因此写入速度快，但是查询速度慢。读写分离的优势是可以分开优化，既然写模式的 LSM 树查询慢，那么读模式就需要改变文件的组织结构，于是 TiFlash 用了一些 B+ 树来优化列文件的查询。

下图是简化版的 TiDB 架构图。



小结

这节课我们一起学习了分布式数据系统的设计原理。所有业务系统从本质上来讲都是数据系统，因此我们可以从分析分布式数据系统的架构，来学习怎么设计业务系统架构。

首先来看 Redis。Redis 是一个基于内存的 K/V 存储，用单线程和 epoll 来提高数据处理速度，用异步主从备份的方式来牺牲一致性，从而降低容灾延迟。Redis 还可以通过 RDB 实现定期的状态备份，牺牲数据完整性来换取处理速度。

RocksDB 是基于文件的 K/V 存储。它用 LSM 树来提高数据的存储速度，但代价是增加了查询的延时。

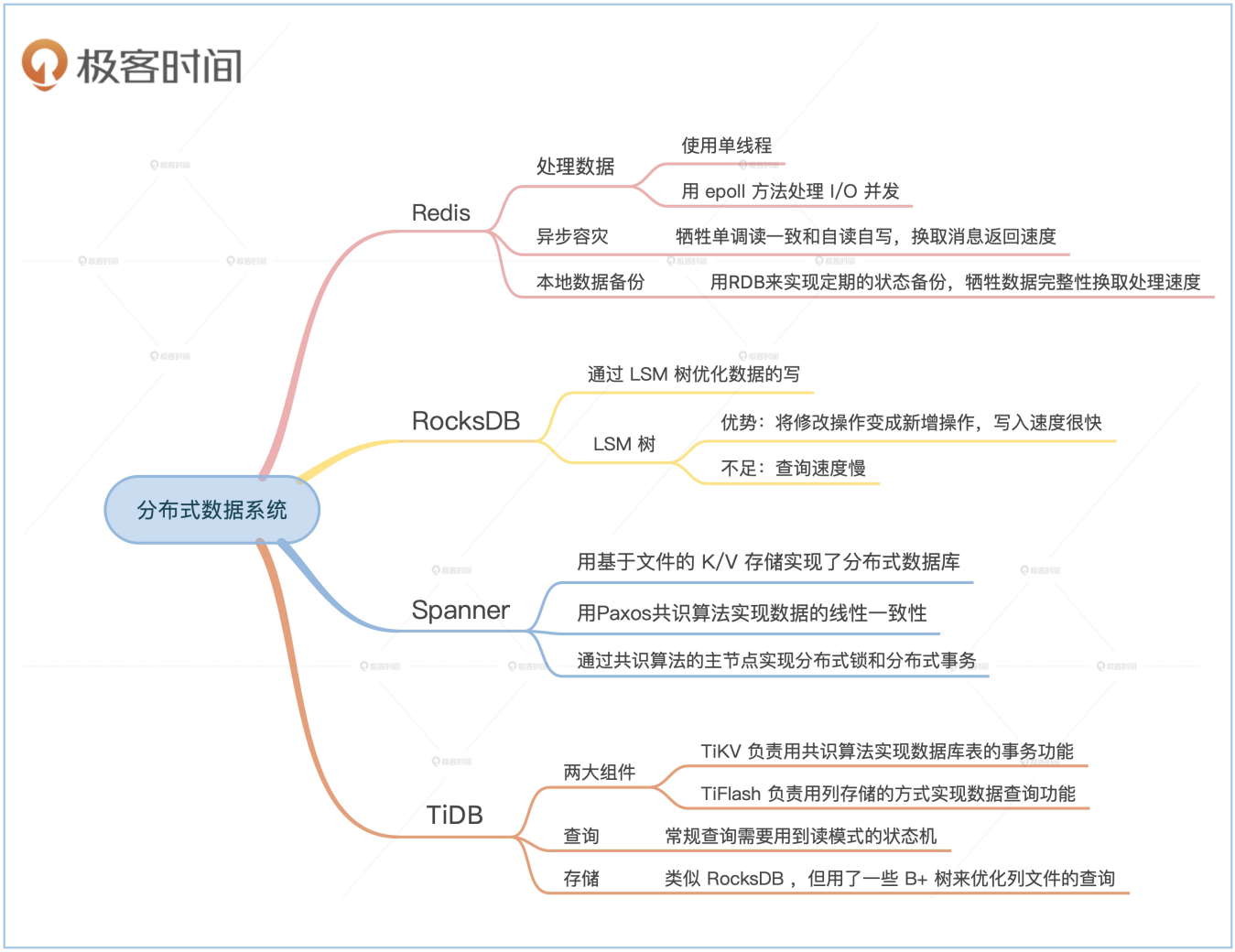
Spanner 用基于文件的 K/V 存储实现了分布式数据库。它用 Paxos 共识算法实现了数据的线性一致性，并用共识算法的主节点实现了分布式锁和分布式事务。

TiDB 进一步优化了 Spanner 的设计。它将关系型数据库的存储抽象成了独立的分布式 K/V 存储，负责解决数据库的写入操作，同时用基于列存储方式的 TiFlash 来实现数据库的查询操作。TiFlash 通过读模式状态机的方式，从 TiKV 的共识算法主节点同步数据。

从这 4 个例子我们可以看出一些常用的架构取舍思路。在分布式环境下，系统稳定性的提高会伴随着处理速度下降，而处理速度的提升会伴随着正确性的下降，因此**稳定性、正确**

性和速度不能三者同时兼顾。

另外，写入的速度快，那么读取的速度就会慢，如果想读写都很快，那么需要用到多份存储，所以**读、写和存储空间也不能三者同时优化。**因此，在分布式环境下一般不会有正确的架构，只有满足业务需求的合适的架构，这就是分布式环境下架构的艺术性所在。



思考题

在大数据之前的时代里，关系型数据库倾向于把所有功能都实现在一起，这样就能充分优化各个组件之间的交互，从而达到很高的单机吞吐量。

但是到了现在这个大数据的时代，单机再也无法承载这么大的数据量，因此需要把数据分散在多台机器上处理。分布式环境下一定存在网络延时，所以单机版的优化并没有之前那么好的效果。

这时候的数据库实现就有了另一个思路。有没有可能把原来的数据库组件都拆分出来，再做一个分布式版本，把每个组件进行独立的分布式横向扩容呢？

这样一来，从整体上看还是一个完整的数据库，但是从实现上来讲是分布式数据库。

关系型数据库有这几个关键的功能：

1. 表存储

2. 主索引和二级索引

3. 缓存

4. 表的复制和容灾

5. 表查询

6. 单机事务

7. 分布式锁

8. 分布式事务

那么问题来了，如果要求你用现在市面上常用的数据解决方案，来拼凑一个分布式关系型数据库，你应该如何选择，如何搭配呢？

欢迎你在留言区记录你的疑问或思考。如果这节课对你有启发的话，也欢迎你转发给同事、朋友，和他一起交流讨论。

提建议

上一篇 17 | 正确性案例（上）：如何实现分布式的事件溯源架构?

下一篇 19 | 正确性案例（下）：如何在运行时进行数据系统的动态分库?

精选留言 (3)

写留言

tt

2021-02-03

作答如下，有的就是一些堆砌，逻辑还不是很顺。

首先，数据库是有状态的，最终目的是为了存储数据，基于老师在这节课中的内容，应该有以下的几个考虑:

...

展开

2

Troy@InfoQ_0a1dfd51...

2021-02-04

老师课程前半部分比较多关于金融系统的业务逻辑，后面倒是比较偏技术实现。有没有考

虑再出一系列课程专门介绍更多金融系统的业务逻辑、牵涉到的机构。本身不是从事金融相关技术，对真实世界金融系统的运作还不是很了解

展开

作者回复: 这位同学你好。金融业务系列也正在筹备中，敬请期待。

1

小动物

2021-02-03

对课后问题是否可以换个角度来思考。数据库无非就是支持数据查询、修改等等操作。而我们平时的业务系统对外暴露的也是查询、修改等等操作。从外部来看，两者行为上是类似的。
那么课后的问题就可以更换为，在分布式的情况下如何保证业务的正常运行。似乎就会熟悉些。...

展开

作者回复: 这位同学你好。这两者从行为上确实是类似的。但是如果类似的处理，恰好就失去了我们这节课的意义。

认知过程是一个从熟悉到不熟悉，然后再到熟悉的过程。因此不熟悉可能才是进步的开始。与君共勉。