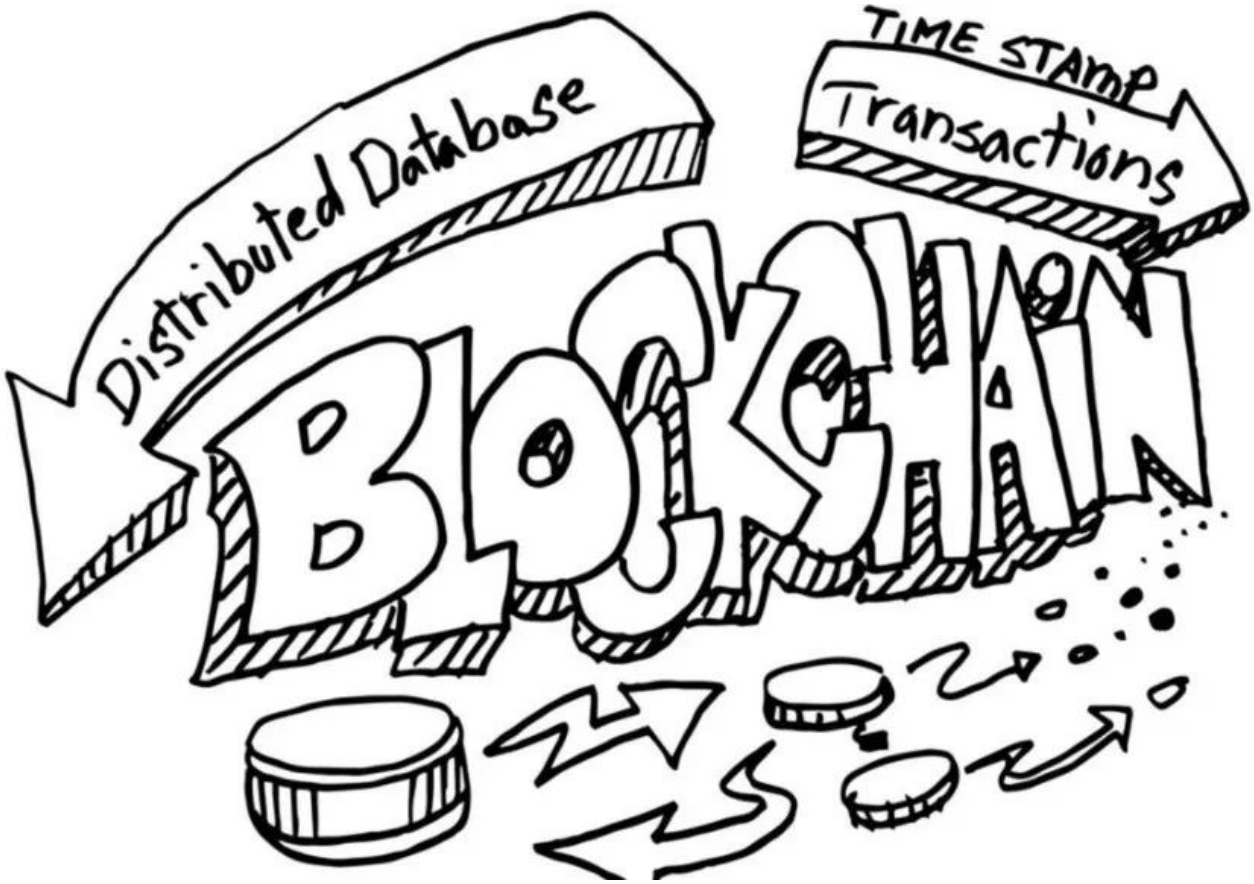
可验证云数据库架构与设计

本文分析可验证云数据库 Veritas 的架构设计，并藉此分享关于区块链行业的一些浅见。

Author	Taosheng Shi
WeChat Contact	data-lake
Mail Contact	tshshi@126.com
Document category	Distributed System & Blockchain
Document location	https://github.com/stone-note/articles

Version	Status	Date	Author/Translator	Description of changes
0.1	Draft	31/03/2019	Taosheng Shi	Initiate
0.2	Draft	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere
1.0	Approved	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere

Contents



区块链的发展到了一个关键阶段。向左走，是一眼望不到尽头的公链和交易所。向右走，是一脸茫然的探寻：区块链如何和古典互联网行业相结合。就像文章《货币、区块链和社交扩展性》所阐述的根本原理，区块链是为了扩大人类的协作范围。向左走的交易所和公链，本质上都是走的交易所和市场属性，构建人类的交易协作的市场。由此，可预见的发展趋势是：BitCoin->Ethereum->FileCoin。由于区块链的这个本质属性，和古典互联网行业的结合，也必须得寻找那些需要协作和共享的场景，烟囱型场景不适合区块链。

区块链的另一个本质属性是可验证中立。由于区块链只适合数字资产的特点（那些基于区块链做溯源的，我都懒得开口怼），我们必须关注区块链为数字资产所提供的基础设施。云计算和互联网的发展已经在计算、存储、网络等方面为数字资产积累了强大的基础设施，只是它们目前还不具备区块链的一个重要属性：可验证中立。而区块链又不具备已有基础设施所具备的强大性能、易用接口、产品化能力和丰富的周边工具。

回顾行业过去一年来的“区块链+”趋势，很多团队“为赋新词强说愁”，试图用区块链来解决所有行业的问题。一个个区块链创业公司倒下之后，都把责任推给寒冬大环境。区块链能做什么、不能做什么？ 去年底央行工作论文已经说得很清楚。


自己没能力就说没能力
怎么你到哪儿 哪都大环境不好
你是破坏大环境的人吗

如果转变认知，不是把已有基础设施能力和行业能力加到区块链，而是把区块链加到已有的基础设施和行业中，或许可以另辟蹊径。例如，bloxroute 实验室的《可验证分发网络：区块链扩容终极解决方案》，把可验证中立引入到 CDN 网络中，解决区块链 p2p 网络分发的根本问题。 再例如，本文要介绍的《Veritas:可验证云数据库和表设计》。

Veritas 是微软研究院和德国慕尼黑理工大学合作的一篇论文，其总体思想是，不是把数据库管理系统的能力增加到区块链，而是把区块链的可信和审计功能增加到已有数据库管理系统中。也就是说，Veritas 的目标是以现有的数据库技术为基础，将区块链技术集成到数据库系统中，从而维护区块链的信任属性和数据库系统的性能和生产力属性。

1 区块链的窘境

经济依赖于协作和交互，共享的数据内含经济价值。常见的数据共享架构如图 1 所示，公司 A 和 B 通过 Web Service 交换数据。但是这样的架构如何协调共享数据中发生的争端？当发生争端时，如何对共享数据和操作日志进行审计？

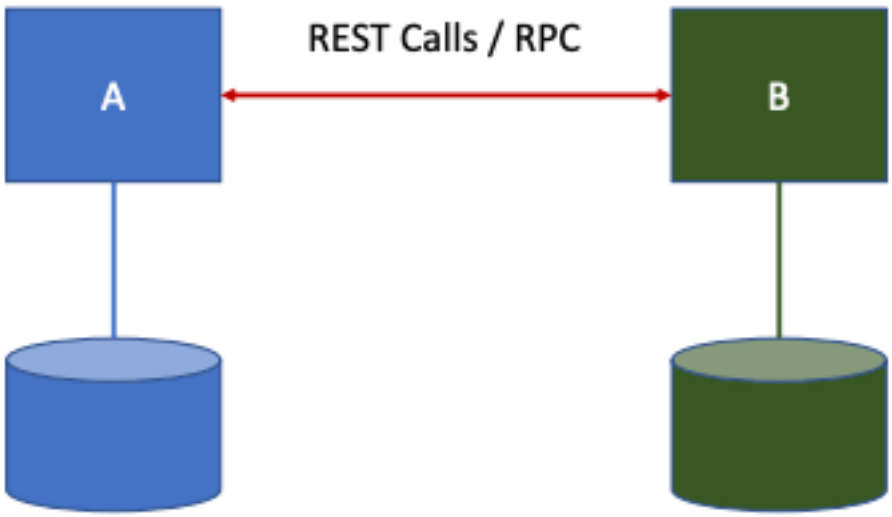


图 1 ： web service

区块链可以作为一个审计组件和共享组件服务于上面的数据共享架构，如图 2 所示，公司 A 和 B 都有自己本地的数据库，然后通过第三方实体（区块链）来进行交易，区块链提供交易的全序审计和记录的不可更改。这样的设计可能是一种进步，也可能是一种倒退，因为区块链既赋予了一些能力，也带来了一些问题。这些问题主要表现在应用程序需要重写，数据操作的原语不同，以及区块链性能低下。这也是整个行业目前面临的窘境：为了拥抱区块链，我们必须放弃在数据管理系统中几十年来的研究成果；一些初创公司为了向区块链中增加数据库系统的能力，至少还需要数年或者数十年的努力。区块链社区已经意识到区块链现有技术和现代数据库能力之间的巨大鸿沟。

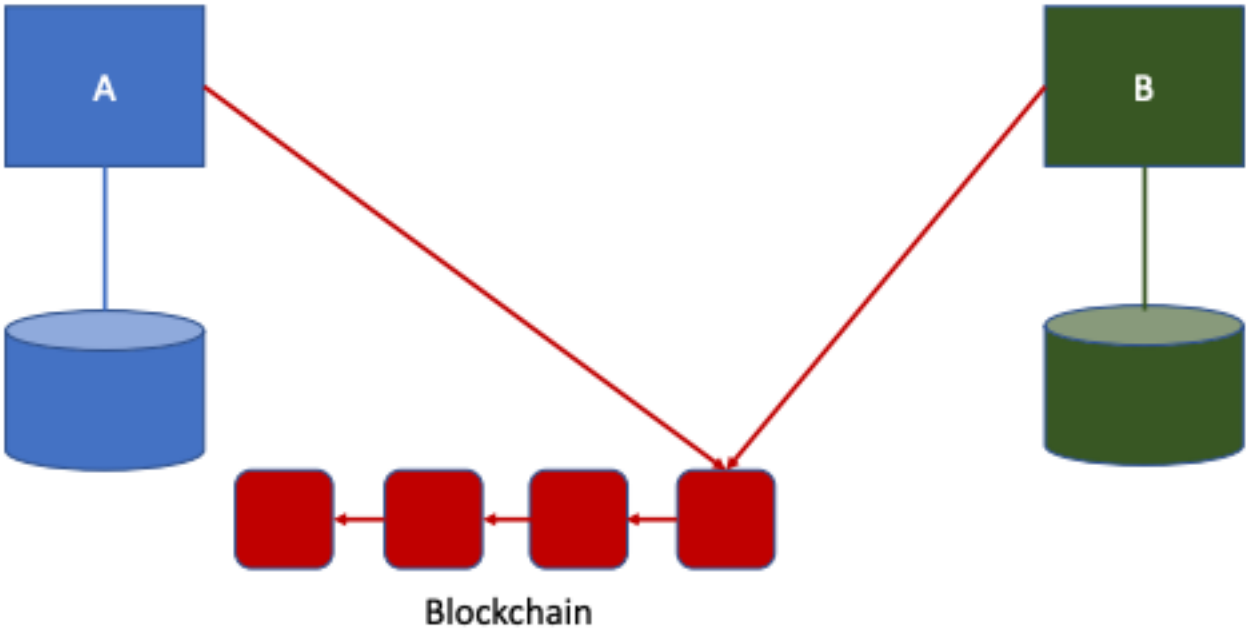


图 2 ：基于区块链的数据共享

2 Veritas 的抽象

Veritas 是如何跨越这个鸿沟呢？简单来说，Veritas 引入了两个新的抽象。

第一个抽象是可验证的数据库系统。可验证数据库创建一个不可变更日志，并允许审计人员检查数据库生成的查询答案和更新的有效性。区块链数据库系统是一个具有常规 SQL 接口的数据库系统，但是它提供了与区块链相同的状态转换、信任和开源可验证性的不变性保证。因此，公司 A 和 B 可以简单地将它们的共享状态写入到区块链数据库中，现在它们可以使用 SQL 进行交互。这样的区块链数据库将解决区块链的许多限制，比如缺少 SQL 接口。但是，由于区块链数据库是一个物理上不同的数据库系统，与它的交互仍然需要通过中间件进行。例如，A 公司不能简单地跨越自己的数据库系统提交区块链数据库的事务，它需要在自己的数据库系统和区块链数据库之间创建一个分布式事务，在实践中，这一功能总是通过异步队列和幂等写实现的。因此，尽管在功能方面迈出了有趣的一步，但区块链数据库始终与成熟的数据库基础设施有一定的距离。

为了缩小这种差距，Veritas 引入第二个抽象：一个共享的、可验证的表。可验证表是在表层次上创建了相同的抽象，这个表可以被共享，作为云数据库不同实例的一部分——这些实例对共享表的操作就像操作一个单独表一样。也就是说，可验证表是 A 公司和 B 公司数据库的一部分，假设 A 创建一个可以与 B 共享的可验证表，这个表同时出现在 A 和 B 的数据库中。这个表可以像其他表一样使用，并且可以被 A 和 B 同步访问，并基于该表编写应用逻辑，包括 ACID 事务。

Veritas 通过将区块链数据库的概念和可验证表的概念放在一起，得到具有不可变更、可访问的日志，具有干净的可审计功能。

3 Veritas 架构与设计

Veritas 抽象概念背后有哪些实现细节呢？图 2 的架构依赖分布式事务和中间件来处理多个系统，这给开发者和运维人员带来极大负担。因此，引入共享可验证表为用户提供无缝体验和免信任协作。图 3 展示了共享可验证表集成到区块链的架构示意。

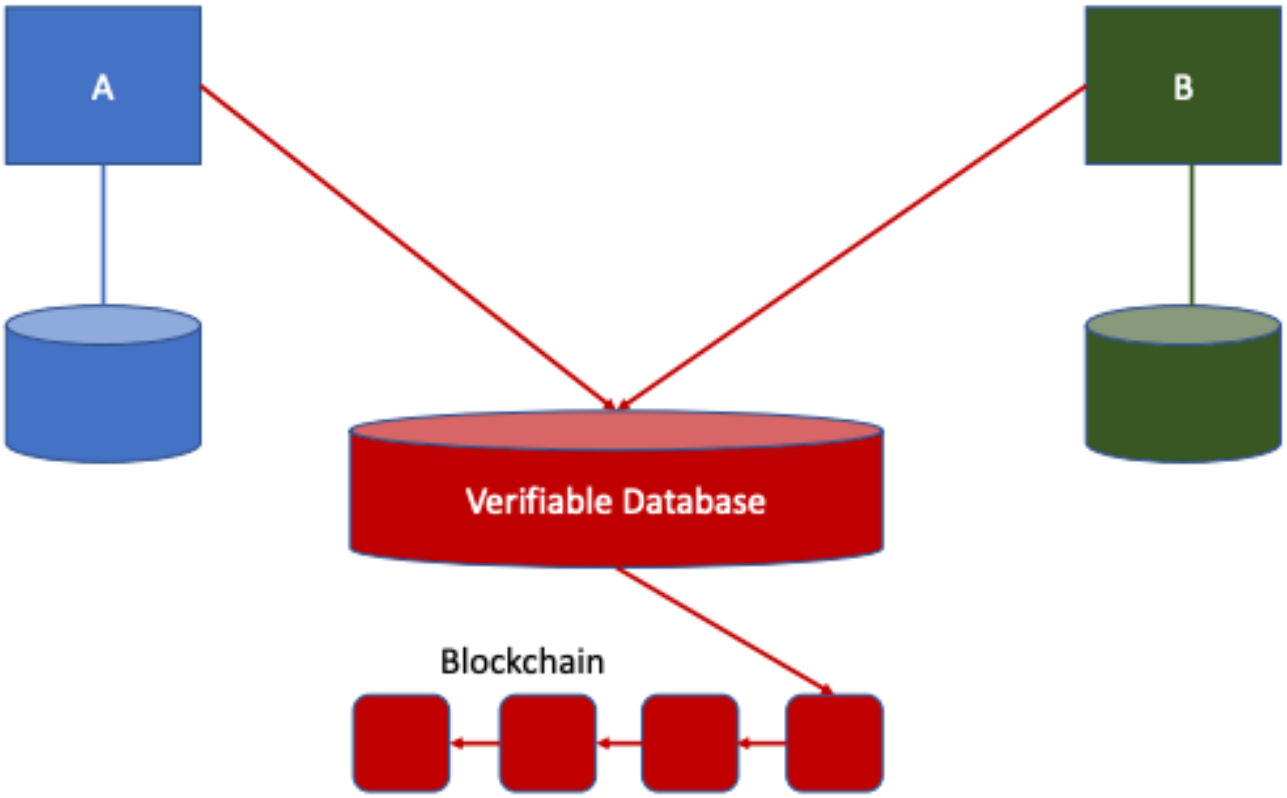


图 3 ： 可验证区块链数据库

图 4 显示了公司 A 和 B 如何使用共享可验证表进行协作。A 将所有小部件订单写入其本地数据库中的共享订单表。A 可以像其他表一样更新和查询这个表，没有排队延迟。并且 A 可以跨越共享表和本地表编写 SQL 事务。同时，这个表的同一个实例对 B 也是可见的，B 对共享可验证表具有与 A 相同的功能。此外，数据库中所有共享表的更新都被写入一个防篡改和可审计的日志，该日志可以实现为区块链。更准确地说，共享表和防篡改日志之间存在 N:1 关系。A、B 和任何可以访问日志的审计人员都可以查看和审查日志。审计人员可以证明 A 现在是否下了订单，以及下单是否发生在最近一次的价格变动之前。同样，B 可以将其所有的价格变化发布到一个共享可验证的报价表中，这个表对 A 也是可见的。

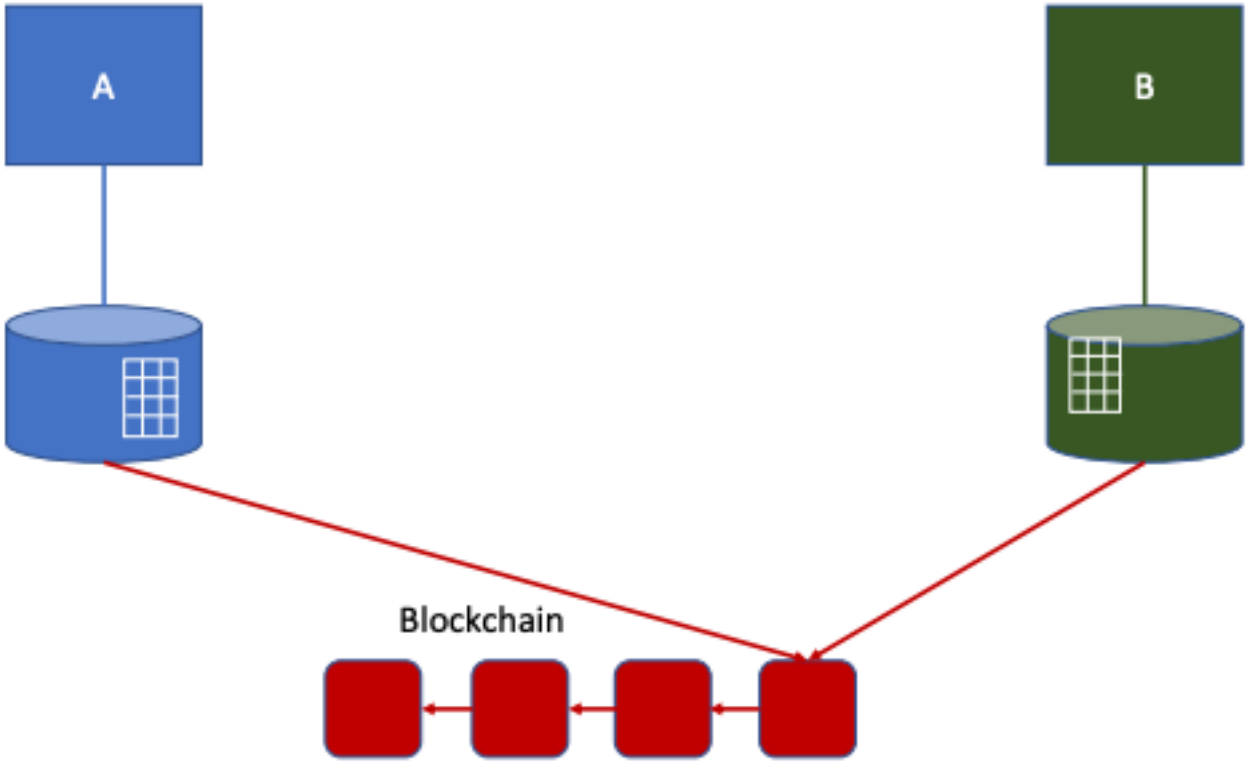


图 4 ： 可验证表

4 可验证数据库设计

可验证性是区块链数据库的最重要概念。验证者如何使用可验证数据库的日志，并对可验证数据库的状态产生共识?

图 5 显示了向可验证数据库中添加验证者的一种方法。在该架构中，区块链仅用于存储验证者的投票。根据特定的区块链技术，区块链操作的成本和性能各不相同，但是它们总体上是很昂贵的。因此，将区块链操作的数量最小化是有利的。在图 5 的架构中，验证者可以通过批量处理他们的投票来进一步减少他们向区块链写入的次数。

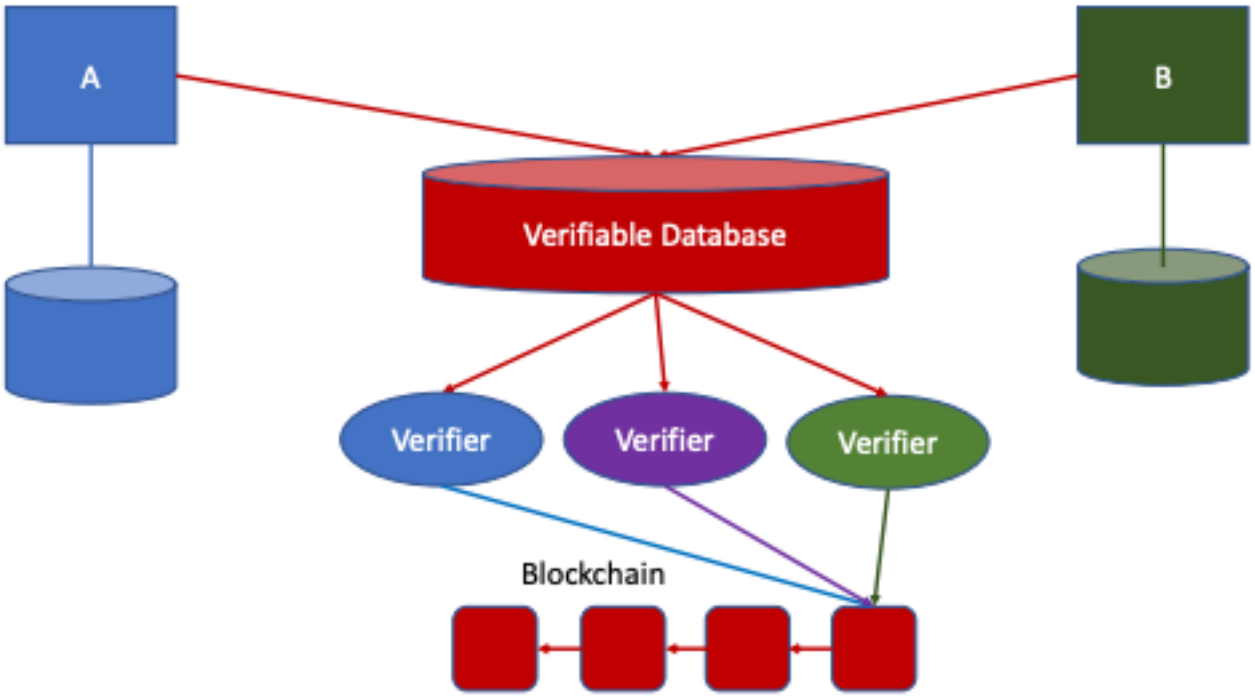


图 5 ： 验证架构

跨广域网络将可验证数据库的日志拆解到验证者的程序中是昂贵而缓慢的。图 5 架构的一个变体是利用新兴技术创建可信执行环境(trusted execution environments，TEE)，并将该 TEE 与可验证数据库集成。

而验证者的验证方法在任何可验证方案中都至关重要。Veritas 验证者内部可以分为粗粒度验证和细粒度验证。

粗粒度验证的每个验证者实现与可验证数据库系统完全相同的逻辑：它重新处理客户机请求并检查是否返回相同的结果。如果结果匹配，验证者投票确认一切正常。否则，则判定违规。

从概念上讲，粗粒度验证很简单。但是，它的成本也很高，因为它重复了昂贵的数据库操作，比如连接和排序（验证排序操作符的结果要比重新排序序列容易）。此外，粗粒度方法要求验证者维护整个数据库的完整副本。因此，粗粒度方法不能很好地与前面描述的 TEE 组合。最先进的 TEE(例如 Intel SGX)有严重的内存限制，不能可靠地执行 I/O 操作，因此很难将一个完整的 DBMS 嵌入到 TEE 中。

粗粒度验证方法的另一个严重问题是验证者继承了数据库系统的所有 bug：验证者的可信计算基础(TCB)过大，实际上不可能验证验证者本身软件的正确性。

细粒度验证方法只检查结果的逻辑属性和所有数据库操作的影响。这种方法对数据库特别有吸引力，因为数据库操作的语义(相当好)是标准化的。这种方法还可以很好地并行化验证。并行化验证是保证验证者能够与具有高并发性的可验证数据库系统保持同步的关键。但是，需要进行更多的研究来优化这种方法，使其适用于更复杂的查询。此外，需要对细粒度方法进行更多的研究，以检查可验证数据库系统是否使用了正确的查询计划(用于复杂查询)，以及可验证数据库系统是否保证了所需的隔离级别。

Veritas 采用细粒度验证方法，主要有三个原因：(a)验证的代价更小，(b)占用空间，(c)安全性。细粒度方法有一个小的 TCB，并且细粒度验证者实际上可以是无状态的。相反，粗粒度验证方法是一个功

能完备的 DBMS，包含数百万行代码，涉及复制整个数据库。小型 TCB 和内存占用空间对于使用具有内存限制的 TEE(如 Intel SGX 和 FPGAs)部署的验证程序尤为重要。

可验证问题中的同步验证（在线验证）和异步验证（延迟验证）的设计问题在在研究文献中引起了极大的关注。

在线验证意味着事务不能提交到可验证数据库中，直到它被所有(或一个仲裁)验证者验证。这种方法对于具有外部影响的事务很重要，例如，在自动取款机上提取现金。相反，延迟验证批量处理事务，并以异步方式定期验证这组事务的影响。

结果表明，在没有无法回滚的外部影响的情况下，在线和延迟验证提供了相同的信任保证。如果在线模型中的事务验证失败，那么仅仅回滚特定的事务可能还不够。事实上，事务可能是完全正常的，而验证失败的原因是可验证数据库的完整性被恶意管理员在此事务很久之前破坏了。因此，在线验证不能保证立即检测到恶意行为，这是对在线验证的常见误解。

5 可验证表设计

本质上，上述在可验证数据库中实现信任的所有设计考虑因素都同样适用于共享可验证表的实现。从概念上讲，可验证数据库和可验证表的最大区别在于并发控制。可验证数据库中的并发控制与任何传统数据库系统的实现方式无异(例如，使用快照隔离或两阶段锁定)，共享可验证表中的并发控制基本上都是分布式并发控制。

图 6 描述了一个包含三个节点的示例场景，分别表示为节点 0、节点 1 和节点 2。每个节点都代表其运行一个 Veritas 数据库的公司，其中包括仅对该公司用户可见的本地表和所有三个公司用户都可见的共享可验证表。每个节点的用户和应用程序以透明的方式向本地表和共享表发出事务，也就是说，用户和应用程序不需要知道其他节点和公司的存在，也不需要知道本地表和共享表之间的区别。每个节点上的数据库系统处理与分布式事务有关的所有问题，保持共享表的副本一致，并使对共享表的所有更新可验证。

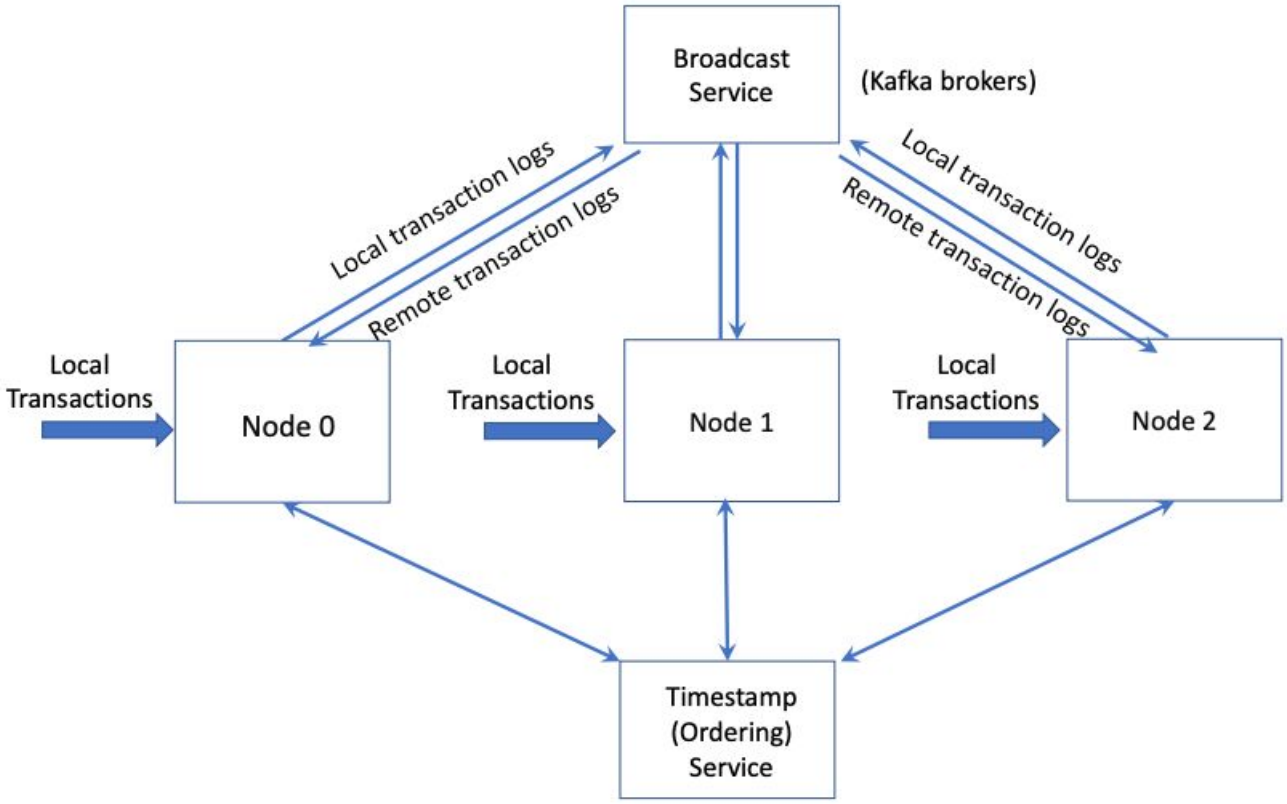


图 6 ：共享可验证表实现架构

涉及到读取和更新共享可验证表的事务，都是全局事务，需要协调。所以 Veritas 引入了以下两个组件：

- 并发控制：基于时间戳的并发控制。
- 日志传播和投票：基于 kafka 的广播服务将共享表的所有更新传播到所有节点。每个节点都将其投票嵌入到它发送到所有其他节点的日志中。

Veritas 基于时间戳排序和一个集中式时间戳服务的乐观协议作为并发控制的协议：一旦全局事务接收到(全局)时间戳，它就会在 Veritas 节点本地投机地(或乐观地)执行。在执行全局事务时，还不知道它是否会成功提交。在 Veritas 中，事务可以由于并发控制冲突或用户中止而中止，就像在任何其他数据库系统中一样。此外，如果其他 Veritas 节点没有验证共享表上的事务，则它们在 Veritas 中可能会失败。

为了执行事务，每个 Veritas 节点保留两个数据结构：

1. 提交水印，TC 表示最高的全局时间戳，节点知道时间戳小于提交成功或终止的所有全局事务。
2. 所有共享表的所有记录的版本历史。此版本历史记录包含 TC 之前全局事务提交的每个记录的最新版本，以及时间戳高于 TC 的乐观事务记录的所有版本。

每个 Veritas 节点定期将所有涉及到对共享表进行读写的日志记录，以及所有全局事务的事务开始、提交和中止日志记录使用广播服务发送给所有其他 Veritas 节点，如图 6 所示。当 Veritas 节点从另一

个节点接收日志记录时，它应用该日志并验证所有提交事务的影响。在应用该日志时，节点使用共享表记录的版本存储检查读/写和写/写冲突。如果节点检测到冲突，它将事务标记为中止，并忽略对共享表副本的所有更新。如果可以验证事务并且没有创建任何冲突，则将事务标记为已提交，并相应地更新共享表的节点副本。

图 7 描述了每个 Veritas 节点维护的状态和日志。除了本地(非共享)表之外，每个 Veritas 节点都保存所有共享表的干净副本。此干净副本包含共享表截止到提交水印时间 TC 的所有记录的状态。在当前的 Veritas 原型中，非共享表和干净的共享表保存在一个 Redis 数据库中。此外，Veritas 节点保存乐观交易共享表记录的所有(已知)版本，以便节点还不知道这些事务的命运。此外，每个 Veritas 节点都保存一个本地日志，其中包含对本地(非共享)表的所有更新，以及对共享表更新的全局日志。在此方案中，(集中式)时间戳服务是系统的一个潜在漏洞，因为它必须是可信的。当然，可以实现不依赖几种服务的分布式事务。

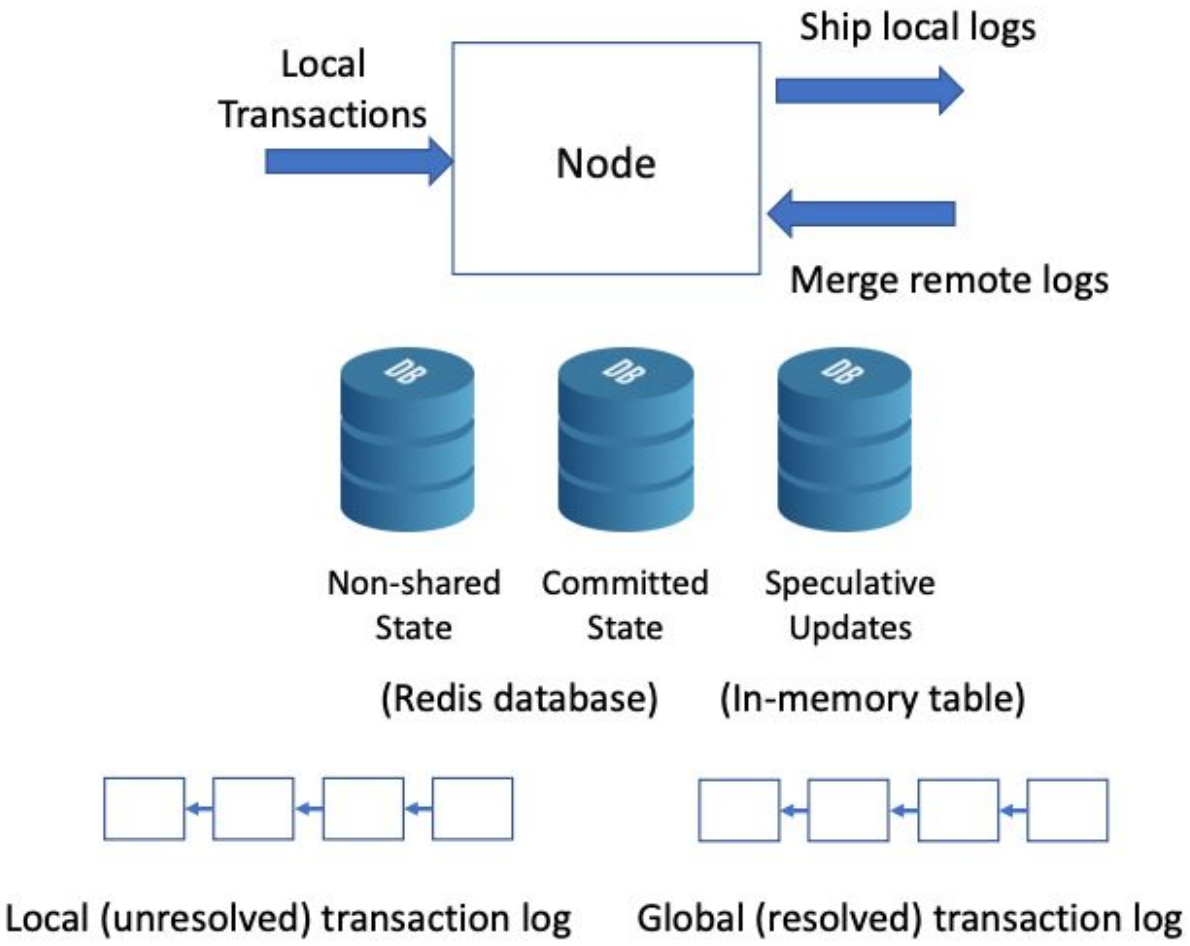


图 7 ： Veritas 节点状态

6 性能评估

Veritas 原型大约有 1500 行 C#代码。评估的重点是探索验证开销作为系统中节点数量的函数。实验使用 YCSB 基准测试和一个共享表，其中 100 万个事务均匀分布在所有节点上。使用了三种不同的工作负载。工作负载 A 是一个更新密集型，工作负载 B 是读密集型，而工作负载 C 是只读的。

图 8 显示了每个工作负载的吞吐量，因为改变了节点的数量：

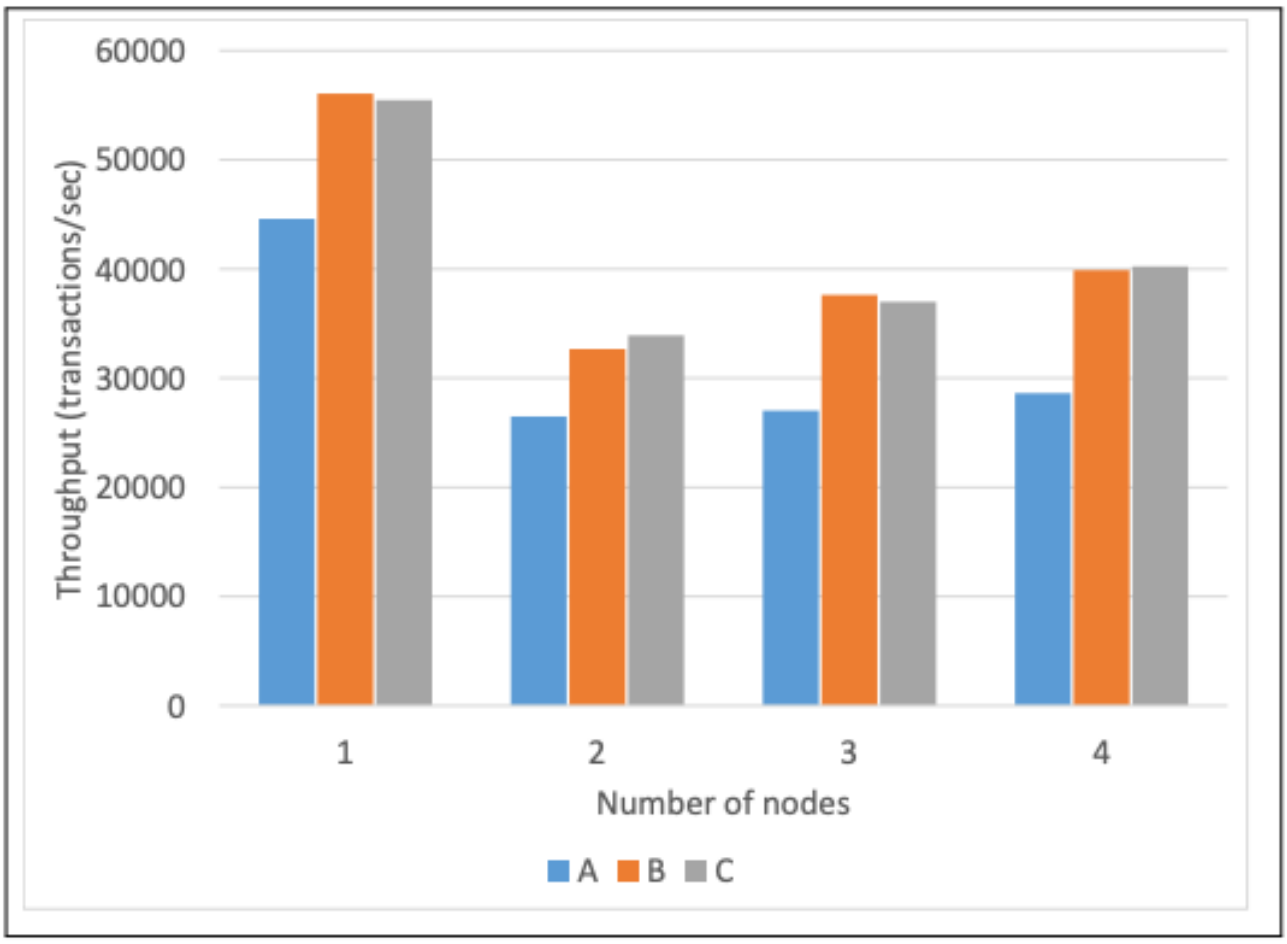


图 8 ，吞吐量：节点变化、数据库大小固定(100KB)

结果有好消息也有坏消息：正如预期的那样，Veritas 为分布式并发控制和验证付出大量额外开销（there is a tax for distributed concurrency control and verification in Veritas.）。对于更新密集型(A)，使用分布式并发控制和验证，吞吐量仅略高于一半。使用更复杂的分布式并发控制协议，可能会改进这些结果。然而，图中显示，即使对于只读负载(C)，即目前在 Veritas 中实现的乐观并发控制方案的最佳情况，开销也是巨大的。然而，图 8 还显示开销不是灾难性的，即使使用 Veritas 目前使用的简单方案，每秒也有可能实现数万个事务。

当我们将节点数量固定为 4 个，并改变数据库大小时，我们可以探索并发控制和验证的相对影响。(较小的数据库大小会导致更多的冲突)。

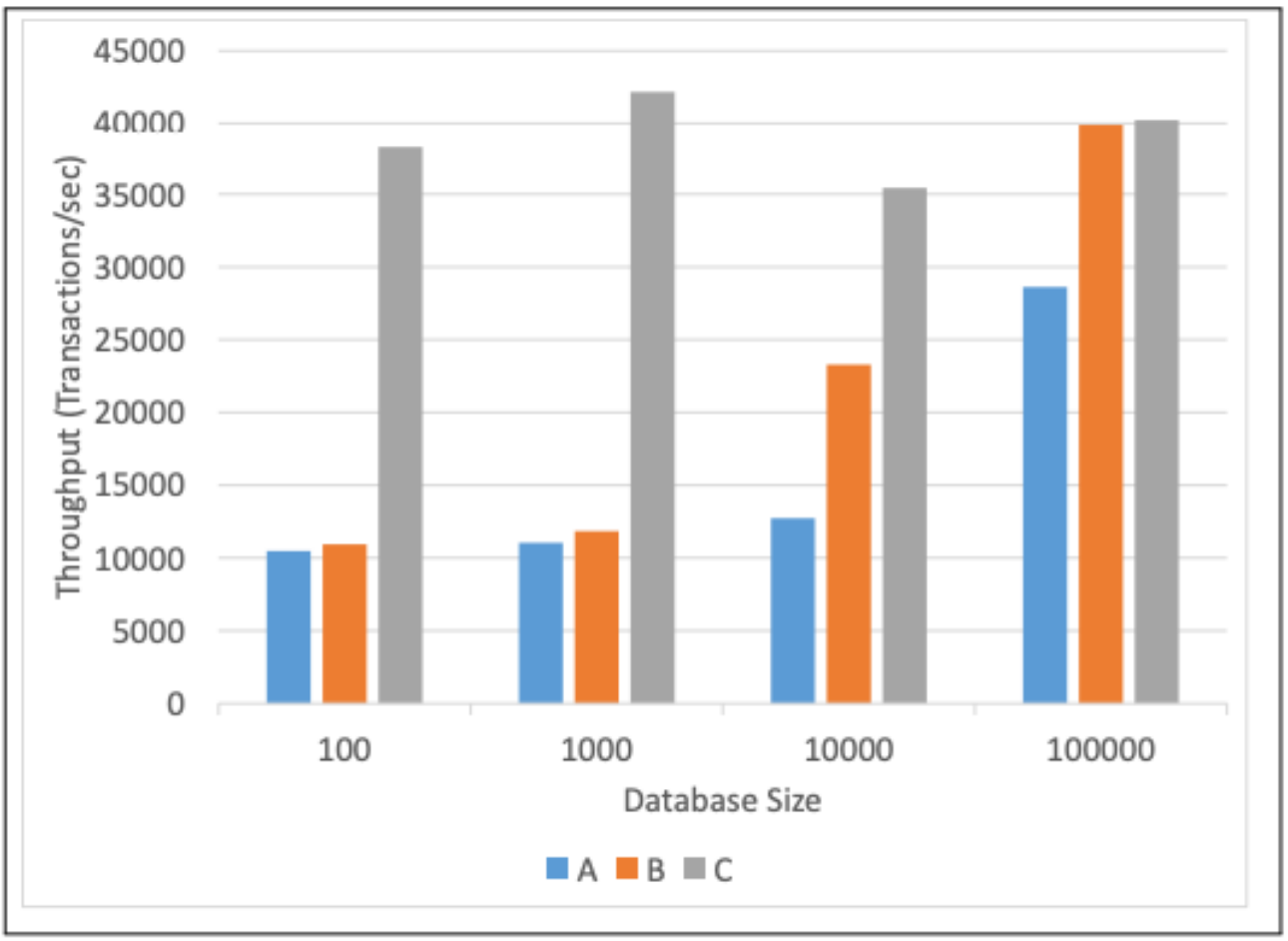


图 9，吞吐量：4 个节点，不同的数据库大小(记录数量)

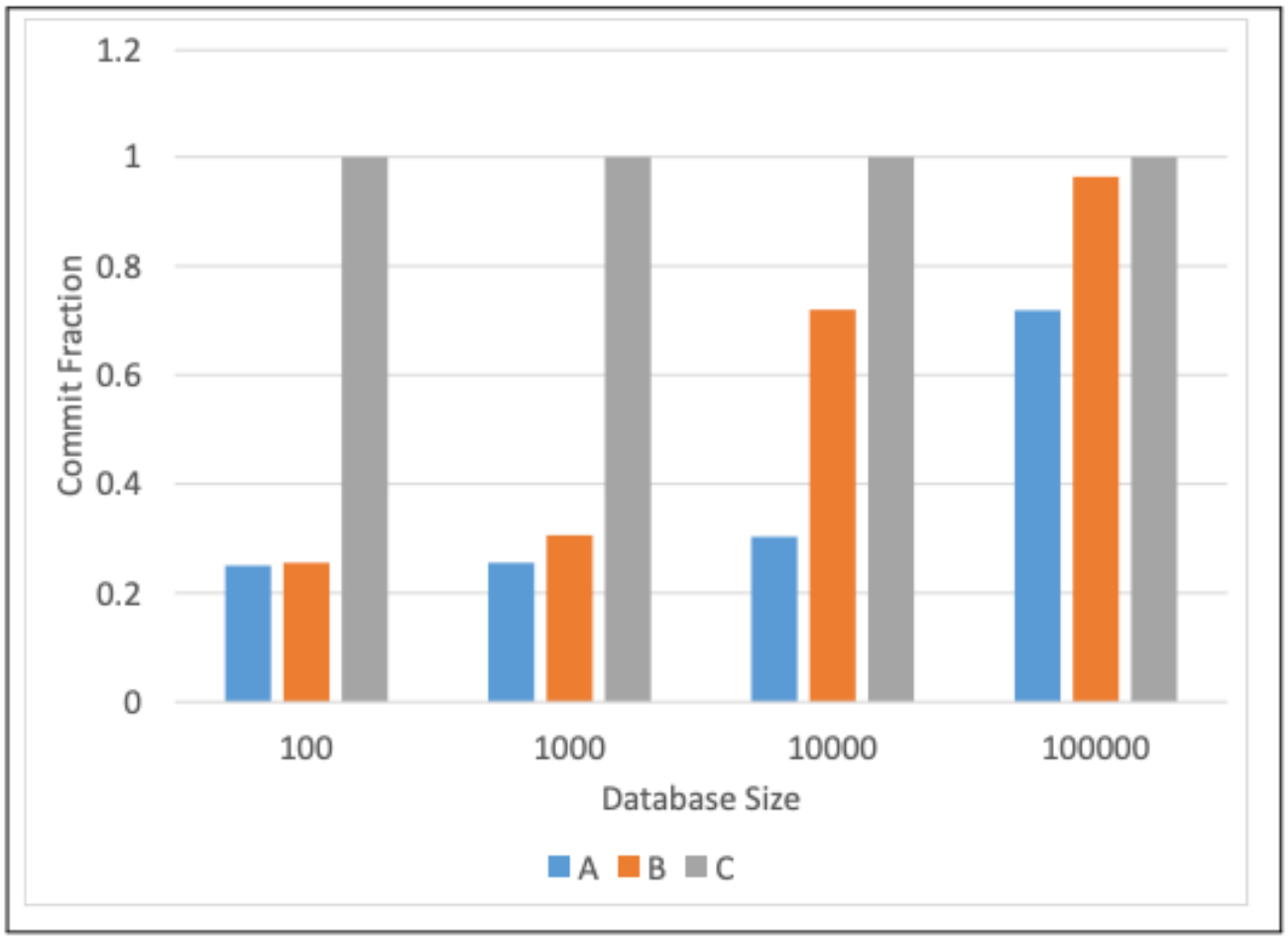


图 10，提交率：4 个节点，不同的数据库大小(记录数量)

这一结果令人鼓舞，因为它表明在 Veritas 中实现共享表不会产生任何额外的瓶颈。实际上，验证和分布式并发控制是有代价的。但是，使用验证(和分布式信任)的数据库系统的瓶颈与传统数据库系统的瓶颈相同。

7 相关研究进展

区块链项目众多，但能够作为相关工作引用的并不多。公链领域的有 Bitcoin 和 Ethereum，基于 POW 共识和全节点验证。联盟链领域的有 Hyperledger 和 Quorum，基于 PBFT 共识和全节点验证。最近提出来的一些项目：Coco，Ekiden 和 Intel Sawtooth Lake 依赖 TEE（例如 Intel SGX）来实现私密性和提高性能。在区块链数据库领域，BigchainDB 是一个与 Veritas 类似的项目，但在架构上差别很大。BigchainDB 更像是一个区块链，使用 PBFT 取代共识算法，使用 MongoDB 作为信任基，并把事务和状态复制到所有节点上。Veritas 则允许以中心化的方式保存数据库状态和进行查询处理，并且具备很小的验证者信任基。在产品方面，目前行业内提供类似产品的还有 Amazon 的 QLDB。

8 参考文献

《Veritas: Shared Verifiable Databases and Tables in the Cloud》

9 后记

前用于 Web、iOS、OSX、Windows 和 Android 的应用程序由前端和后端组成。后端与数据库交互，是存储数据的地方。前端是用户直观地看到应用程序的地方，即由各种标签和控件组成的界面。