《Centiman: Elastic, High Performance Optimistic Concurrency Control by Watermarking》读书报告

16337102 黄梓林

文章介绍了一个高性能的可扩展的 OLTP(On-Line Transaction Processing)系统——Centiman，它提供了可串行化的保证。Centiman 使用带有分片验证的 OCC(optimistic concurrency control)，设计用于松散耦合的云架构;它可以部署在现有的键值存储上，组件可以根据应用程序的需求灵活地伸缩。Centiman 尽可能地避免了同步，并使用了水印来减少错误的中止和以及优化只读事务。除了介绍 Centiman 的原理、运行机制外，文章还描述了与 Centiman 相关的实验探究，证明了系统的性能和可扩展性。

事务(transaction)处理一直是数据库功能的基石，而在云时代，它仍然如此。当前，研究界正付出巨大的努力来支持云上和分布式系统中的事务处理。为此，各种各样的权衡已经出现：例如，为了获得可伸缩性的好处，减弱了一致性保证；而为了支持强一致性保证，限制了系统允许处理的事务类型；或仅在分区内，划分数据和支持 ACID 语义，而在分区间提供较弱的保证。

文中的 Centiman，则是一种专为云计算而设计的事务处理解决方案。它提供完整的 ACID 保证，且每秒能进行数百万次操作，而不会限制允许的事务类型或语义；除此之外，它还能较简单地在现有的键-值对存储上部署基础架构；而最后，它还允许对系统的每个组件（数据存储区，处理器，验证器）进行弹性扩展。

Centiman 使用 OCC 协议的变体来维护事务性保证，而它在低争用（low-contention）设置中的低开销，使得它在分布式事务处理中非常流行。

Centiman 的显著特点是验证是分片的，并且在一组验证器上并行进行（验证器数量在需要时可以增长或减少）。因此，在多租户环境中，小租户可以使用一个验证器，并享受当地的优势，而大租户可以弹性扩展。在正常操作下，系统中唯一的同步点是验证的开始（来确保事务以时间戳顺序进入验证）以及负责事务的处理器收集所有验证器的结果并将它们组合成单一结果（提交或终止）。而其他所有阶段都异步进行。特别是，事务的写入可以异步传播到存储，并与其他事务的读取和写入交织。

这种松散耦合的设计确实带来了挑战。第一个是虚假中止。验证器不知道验证的全局结果; 即使事务中止，个别验证器也可能认为它已经提交。验证器将因此验证将来的事务，潜在地检测虚假冲突，从而可能检测到虚假冲突

第二个挑战涉及只读事务。在正常的 OCC 中，如果能够确保事务读取数据库的一致快照，那么可以允许这样的事务绕过验证。由于 Centiman 不需要将事务的写入以原子方式安装到存储器，因此很难确保事务读取数据库的最新且一致的快照。因此，上述优化难以实现。

而上述两个问题都可以通过同步来解决; 可以将中止决定发送给验证器，并且可能需要对写存储操作的安装进行事务性保证。但是，这会引入额外的阻塞。因此，可以使用水印抽象技术通过系统异步传播关于哪些事务已经提交并

完成了写入存储操作的信息，存储中的哪些记录是“稳定的”。这解决了这两个问题，而没有引入系统中的阻塞点。水印允许验证器了解中止的事务并更精确地维护其状态，并且在某些情况下，可以使用它们绕过只读事务的验证。除了 Centiman 之外，水印技术还可用于提高其他 OCC 系统的性能

Centiman 包含以下主要组件：数据存储区，由处理器和验证程序组成的事务处理子系统，全局主服务器和发布业务处理请求的客户端。

数据存储区是一个键-值储存区，可以被分块或者复制，而这些操作都是对 Centiman 透明的。将数据储存区中的键-值对称为记录，每个记录都有相应的版本，即与记录对应的时间戳。

客户端向系统发送事务请求。无论何时，当客户端发布一个请求时，它与一个特定的处理器交流，而这个事务与此处理器关联。在此事务处理的生存周期中，此处理器将它的读请求发送到数据储存区，将它的写请求缓存在一个私有工作区，为每个事务安排一个时间戳，发送验证请求到一个或多个验证器，发送写请求到内存（若必要），以及向客户端反馈。

而关于事务的生存期：事务的生存期遵循 OCC 模型。每个事务有一个读请求序列，当它从数据储存区读和向私有工作区写时；有一个验证序列，当系统判定这个事务能否被呈递时；而当验证成功时，还会有一个写序列，事务的写操作会安装在数据储存区。

当事务想要读一个记录时，处理器会发送一个合适的 get(key)请求到存储。get 会返回一个(value, version)对，处理器会将这个对加入到事务的读操作集合中。

当事务想要写一个记录的 key 时，处理器会把这个写操作缓存在本地的私有工作区，并将要写的 key 值加入到事务的写操作集合中。

当一个事务准备好呈递时，处理器会给它安排一个时间戳；只要事务被赋予了时间戳，它就准备经历验证了。验证会在一个或多个验证器上按照时间戳顺序进行，而验证过程会使用读操作集合 RS(i)和写操作集合 WS(i)： 对每个验证器，RS(i)仅是用于验证 i，在验证后会被丢弃；而 WS(i)可能会被用于验证后面的事务，故会被缓存在验证器中。

只有当处理器收到所有参与验证的验证器的提交命令时，它才会决定将事务呈递；否则被丢弃，丢弃之后将不会后续操作，事务将会完成；若事务被呈递，处理器会为写操作集合中的每个 WS(i)发送一个 put 请求到数据储存区。

而关于验证，Centiman 的验证方法为——用水印进行分片验证。

水印是在关于最近完成的交易的时间戳的系统信息中传播的元数据。在经典的 OCC 中，验证和写入阶段处于关键部分，这相对容易获得此信息。

而在 Centiman 中，当事务从存储中读取记录 r 并接收对(key, version)时，处理器计算具有以下属性的水印 w：如果记录 r 在时间 t 具有水印 w，那么在时间 t，所有带有时间戳 i≤w 的事务都已经在 r 上安装了它们的写入，或者不会对 r 进行任何写入。

直观地说，这表示任何带有小于 w 时间戳的事务对 r 进行的更新都会反映在读取中。处理器在读取时计算水印，并将其与密钥和版本一起存储在读取集中。因此，读取集现在包含三元组(key，v，w)，其中 v 是版本，w 是水印。

水印一方面可以减少虚假的中止，且不需要额外的通信，因为中止事务的写入集最终会“老化”并且落后于水印的时间戳，因此它们再也不会被认为是在验证中；另一方面还可以通知验证器垃圾收集策略，一旦所有在执行中和未来

的事务的读水印大于 i，垃圾收集 WS（i）是安全的。

水印还可用于对只读事务的优化。如果能确保事务读取的是与数据库一致的快照，那么它们将不需要验证。在 Centiman 中，水印允许处理器对只读事务进行简单而保守的检查; 如果检查通过，事务可以立即提交，需要发送给验证器；而如果检查失败，事务将进行正常验证。

Centiman 拥有着另一特性：弹性扩展部件。Centiman 的设计是为了让每一层的功能——数据存储、处理器和验证器——可以根据需要添加更多的节点和/或迁移工作负载。关于前两者，可以假设在存储级别，缩放是由数据存储本身处理的，而处理器也可以直接在系统中添加或删除。

而对于验证器，从根本上说，验证器扩展(向上或向下)和验证器迁移在系统中需要相同的操作:改变在一组验证器之间划分密钥空间的方式。全局主服务器监视系统负载，决定何时执行缩放和/或迁移，并决定新的关键空间分区。

任何新的验证器节点都注册到全局主服务器并连接到所有处理器。接下来，全局主服务器决定新的密钥空间分区。它基于新旧划分，通知所有处理器，发送验证请求。

当系统处于过渡阶段时，验证器在旧分区和新分区下运行，并在旧分区和新分区下发布验证决策。然而，权威的决策是在旧的分区下做出的。当验证器在新的分区下操作时，它可能并不总是有足够的状态来验证事务；在这种情况下，它要么使用“abort”，要么使用特殊消息“unknown”进行响应。

最初，在旧的和新的分区下做出的验证决定可能是不同的。有可能全局决策在旧分区下是“提交”，而在新分区下是“中止”，因为至少有一个验证器还没有建立足够的状态。然而，相反的情况永远不会发生:新分区下的决策至少与旧分区下的决策一样保守。随着时间的推移，验证器在新的分区下进行计算，在两个分区下所做的决策会趋于一致。

在合适的时间点上，全局主服务器决定完全切换到新的分区。它可以基于在两个部分下的验证决策的收敛性来做出此决策——例如，当验证决策在特定的时间段内没有出现任何分歧时，它就会进行切换。另外，单个验证器可以通知全局主服务器它们已经积累了足够的状态切换。

切换到新的分区需要主服务器器和所有处理器之间的同步。一旦切换完成，旧的分区就被“遗忘”;处理器只在新的分区下向验证器发送请求。而验证器不需要通知交换机;它们只会停止接收旧分区下的验证请求。

验证器缩放协议只引入了两个开销来源。首先，在过渡期间，发送给一部分数据的验证请求的正常数量是正常数量的两倍。其次，系统需要全局同步来完成切换到新的分区。但是，在验证器之间不需要移动任何状态。

Centiman 的分片验证也是其一大特点。实现 Centiman 分片验证的三个重要方面是：时间戳顺序验证、水印和故障恢复。

关于时间戳顺序验证：

在验证器算法中，事务必须在处理器上加上时间戳，以及按照时间戳顺序验证。

处理器能够基于物理系统时间安排时间戳，也可基于一个唯一的整数集安排单调递增的时间戳。

每个验证器维护一个在之前开始，在当前结束的滑动时间窗口。窗口或是在物理时间上被定义，或是基于时间戳，在逻辑的时间上被定义。

验证器缓存到达的验证请求。它通过不断推进滑动窗口来处理它们；任何

移出窗口的请求被按时间戳顺序处理。如果验证器收到一个时间戳前于当前滑动窗口的请求，它以‘abort’回应。因此，一些事务会被终止因为他们的验证请求没有按时到达验证器。但是，只要在专用的机器上配置 Centiman，以及使用一个快速、预测延迟较小的网络，是能够调整验证器的时间窗口以得到最少的中止的。

关于水印的实现：

每次一个记录被读，处理器就要为这个记录计算一个水印。Centiman 使用一个合适的、保守的计算水印的方法，利用了水印是向下闭合的事实。

关于故障恢复：

处理器以及验证器的故障恢复。假设数据存储使用现有的技术来实现容错；全局主处理器是使用健壮的基础设施实现的；处理器和验证器节点是故障停止的。

每个处理器节点都维护一个写前日志，以便对数据存储重新执行写操作。因为在验证完成之前不会执行写操作，所以不需要撤销日志记录。收到一个来自客户端的事务 T，处理器为 T 创建一个初始目日志，并且异步地写它。在发送验证请求给验证器之前，它异步地记录下 T 的写操作集。等到收到所有验证器的回应，它决定是递交 T 还是中止 T ，并写下对 T 的决策；日志包含 T 的时间戳。如果决策是呈递，处理器强制提交日志，并最终通知客户端以及将写操作集发送到数据存储区。当写操作被安装，处理器会异步地为 T 做一个完成日志。若是日志包含一个递交决策但没有完成，为了修复一个故障的处理器，需要读日志并重做事务 T 的写操作。

从验证器和其他处理器的角度来看，处理器故障是透明的；唯一的负面影响是会延迟将某些写入安装到存储中，并中止处理程序尚未作出提交决定的事务。特别是，处理器可能在做出提交决策后失败，但在强制提交日志条目之前失败；在这种情况下，在恢复时，事务将被中止。这不会违反正确性，因为客户端还没有收到提交响应。验证者会认为 T 已经提交，这可能导致虚假的中止;然而，这个问题会随着时间的推移而自行解决。

不需要对处理器水印进行日志记录；在恢复时，处理器可以简单地将其本地处理器水印设置为上一个事务的最高时间戳，一旦确定该事务的写操作已经安装，它就会为上一个事务设置提交日志条目。

即使验证器保持状态(例如事务的写操作集)，也不需要日志记录来恢复。当一个验证器故障时，需要在那个验证器上终止所有等待验证请求的事务——如果处理器在等待响应时超时，这将自动发生。而为了恢复验证器故障，需要添加一个新的验证器来代替失败的验证器。

以上即为对 Centiman 这一系统的简要介绍。

而在未来的工作中，我们可以在系统设计空间中探索替代方案，例如通过让验证器彼此共享状态或让处理器将 commit 决策广播回验证器来消除假中止；还可扩展 Centiman，将 OCC 用于低争用设置和高争用设置中的其他协议，并将水印应用于其他基于 OCC 的系统。