无法保障的.目前,我们只知道操作一致性会制约事务一致性,但两者的具体关系尚不清晰,有待进一步的研究工作加以澄清.Balis 在关于 HAT 的论文中阐述:当操作一致性受节点可用性的制约而无法被完全保障时,事务一致性只能达到读已提交的级别;这算是对二者关系的一个局部分析结论.

其次,即使操作一致性得到了全面保障,这并不能说明事务也能得到保障.为了说明这一点,我们关注操作的线性一致性和事务隔离级别之间的关系,即当系统可以保障所有事务的操作满足线性一致性时,该系统依旧无法为事务提供可串行化隔离级别.这是因为,事务的一致性不仅需要考虑操作之间的关系,还需要考虑不同事务之间的关系,而操作一致性仅需考虑不同操作之间的关系即可.因此,即使系统提供了线性一致性模型,也无法保证能够避免掉某些事务的异常现象的发生,如不可重复读和丢失更新,因而无法提供可串行化隔离级别.如图 6 所示的执行序列.

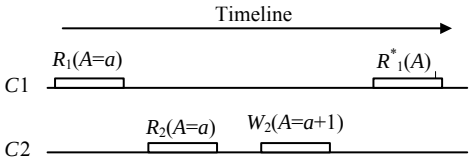

Fig.6 A schedule of the unrepeattable read anamoly

图 6 不可重复读的异常调度

事务 T_1 和事务 T_2 分别在客户端 $C1$ 和客户端 $C2$ 上执行,按照线性一致性模型,操作在服务端上执行的序列为 $R_1(A=a)R_2(A=a)W_2(A=a+1)R_1^*(A)$,因此,事务 T_1 的 $R_1^*(A)$应该获取到 $A=a+1$,这和该事务前一次的读取操作 $R_1(A)$返回的结果并不相等,属于典型的不可重复读异常.

事务系统为了避免这些异常,通常需要采取额外的手段来调整事务操作的执行序列.例如,系统可以采取两阶段封锁(two-phase locking)的机制,构建一个共享的锁表.当事务想要执行一个操作的时候,只有从锁表中获取对应数据项的锁权限后,才能真正执行该操作.

综上,操作一致性是事务一致性的必要条件.在实现操作一致性的基础上,事务一致性还需要进一步的并发控制机制才能得以保证.Lomet 等人研发的 Deuteronomy 系统[51−53]正是为了在保障操作一致性的 NoSQL 数据库上实现事务一致性.

3.2 节点级高可用与一致性的关系

在节点高可用系统中,无论副本节点在主分区还是备分区,当其收到客户端的请求后,均能进行处理.由于不同分区间无法进行通信,整个系统无法维护一个全局时钟.但每个副本节点可以维护一个本地的局部时钟,该时钟在跨分区操作中作用有限.

3.2.1 节点级高可用与操作一致性

本小节将讨论节点级高可用系统能够实现的操作一致性模型.由于操作的特性与一致性模型有直接的关系,我们接下来分析在节点高可用的系统中,有哪些操作的特性可以满足.

最新性:不可满足.由于分区的存在,并且分区的时间可以任意长.因此,当某个分区上的副本节点收到读取操作的请求后,如果请求的数据项在其他分区上有更新版本的数据,则该副本上的请求数据项无法满足最新性.如图 7 所示的执行序列.

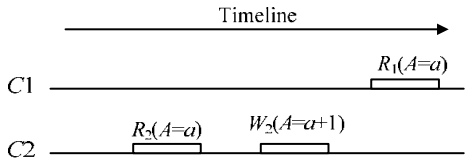

Fig.7 A schedule of the lost recency anamoly

图 7 缺少最新的调度

$C1$ 和 $C2$ 分别连接到主备分区,由于系统为节点级高可用,$C2$ 的写操作 W_2 可以顺利执行,但 $A=a+1$ 并没有同步到主分区,因此,$C1$ 上后续的读取操作 $R_1(A)$仍获取的是较旧的版本.

单调读、单调写和写后读:可满足.由于在同一个会话中的所有操作均由同一个副本节点负责处理,那么操作在服务端执行的顺序可以完全按照在会话中发起的顺序.因此,节点可用性系统可以保证同一个会话内的所有操作满足单调读、单调写和写后读.

读后写:可满足.副本节点在收到写入操作的请求后,并不立即将其结果作用于本地,而是将其缓存在一个单独的空间,客户端的读取操作暂时无法获取该写入的结果.当系统不存在分区后,所有的副本节点根据因果关系协商缓存的结果,并达成一个一致的结果.此时,客户端再访问任意一个副本节点,总能获取相同的结果.

由于节点级高可用系统无法满足最新性,因此无法对外提供线性一致性的服务.而单调读、单调写、写后读和读后写均可得到满足,因此节点可用系统可以对外提供因果一致性和 PRAM 一致性的服务.

3.2.2 节点级高可用与事务一致性

本小节将讨论节点级高可用系统能够实现的隔离级别.我们接下来分析,节点级高可用系统在分区的情况下,其能够避免哪些事务的异常现象.

脏读:可避免.在本文的事务模型中,由于每个事务仅包含一个写操作,并且系统保证该写操作的原子性,也就是说,当某个副本节点收到写入操作的请求后,需要保证该事务提交后,其对应的结果才可以对外可见.因此,在本文的系统模型下,系统可以避免脏读,可以提供读已提交的隔离级别.

不可重复读:可避免.由于一个事务的所有操作只能转发到同一个副本节点上,副本节点只需要为事务保存相应数据读取的版本,当事务再次请求相应的数据项时,副本节点返回保存的版本即可.当事务执行完成后,其对应的版本便可以删除.

幻读:可避免.由于同一个事务仅与一个副本节点保持连接,因此,我们可以让副本节点为事务读取的数据保存其对应的版本.当某个事务再次读取某个数据项时,可以将之前读取的版本直接返回.

丢失更新:不可避免.两个事务分别在主副两个分区上执行,这两个事务读取相同的数据项并获取到了同一个版本的数据,然后都修改了该数据项并将结果写回至各自的副本节点.由于分区间无法通信,事务在提交的时候无法对另一个分区上的并发事务进行检测.为了保证可用性,这两个事务都最终提交成功.如图 8 所示的执行序列.

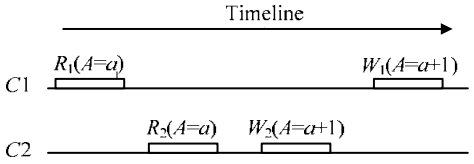

Fig.8 A schedule of the lost update anamoly

图 8 丢失更新的异常调度

事务 T_1 和事务 T_2 分别在 $C1$ 和 $C2$ 上,由于系统为节点级高可用,T_1 和 T_2 均提交成功,最终 $A=a+1$,但该结果并不符合 T_1 和 T_2 任何串行化执行的结果,即 $A=a+2$.

写倾斜:不可避免.丢失更新针对的是同一个数据项,如果同一个事务的读写发生在不同的数据项上,则节点可用系统将会产生写倾斜异常.

我们可以发现,节点级可用系统可以避免脏读异常,因此这类系统可以对外提供读已提交隔离级别.上文中提到的隔离级别,除了读已提交,均需避免丢失更新异常,而节点级高可用系统无法避免丢失更新,因此,该类系统无法提供可重复读、可串行化等更高的隔离级别.

3.3 服务级高可用与一致性的关系

在我们的例子中,最多存在两个分区,且两个分区的节点数目并不相等.如果系统为服务可用,则客户端的请求只能在主分区的某个副本节点上得到处理.也就是说,副分区上的副本节点将拒绝处理客户端的读写请求.

因为所有的读写请求均发送到主分区的副本节点,且主分区的副本节点之间可以进行通信,我们可以采取全局时钟的方法为每一个操作请求分配全局唯一且递增的编号.当其中某节点在处理一个客户端操作请求的

© 中国科学院软件研究所 http://www.jos.org.cn

时候,需要保证该请求前面的操作均已执行.因此,服务可用的系统可以保证一致性模型中所有的特性,可以对外提供线性一致性.但是,每当系统收到一个操作请求时,主分区内的节点均需要进行一次协商来为其分配编号.通常,协商/共识协议需要额外的节点间交互,这增加了每一个操作的延迟,从而会影响系统整体的性能.

事务系统需要维护额外的信息来保证事务的隔离性,我们可以在主分区的所有副本节点上维护一张全局的锁表,当其中的一个副本节点收到事务的一个操作请求后,就会对共享的锁表进行更新(获取锁相应数据项的锁权限).当更新成功后,其他的副本节点均可发现锁表的变更.因此,服务可用的系统可以采取两阶段封锁的机制来实现可串行化隔离级别.然而,一方面,主分区的副本节点之间需要通过协商来同步锁表的更新;另一方面,当事务操作的数据项已被其他事务获取锁权限时,该事务的操作会被阻塞.我们已经知道,一个事务执行的时间会影响与其他事务的冲突的概率[54].在服务可用的系统中,由于事务执行时间变长,系统的整体性能会受到较大的影响.

因此,系统在满足服务级高可用的情况下,可实现线性一致性和可串行化隔离级别.但是,协商/共识协议的实现增加了系统工程开发的难度;由于协商/共识协议引入了额外的开销,与节点高可用系统相比,其性能有所降低.

3.4 人工介入后可用与一致性的关系

在传统的基于复制的分布式数据库系统中,在所有的副本节点中存在一个主副本节点,其他副本节点为备份节点.客户端只能与主副本节点建立会话连接,也就是说,所有客户端的操作请求均会发送给主副本节点,由该节点负责处理执行.因此,该主备份节点可以对所有的操作请求进行调度,从而可以保障最新性以及会话特性,对外提供线性一致性服务.

我们可以在主副本节点上维护锁表信息,事务的操作请求需要先从锁表中获取对应数据的锁权限,再在主副本节点上执行处理.因此,人工介入后可用的系统可以采用两阶段封锁的策略来保证事务的串行化执行,可以对外提供可串行化的隔离级别.

为了保证数据的持久化,该类系统通常采用积极机制(eager)来复制数据.具体来说,当主备份节点收到事务的提交请求后,需要确保该事务在所有副本节点上都执行成功后才能提交该事务并响应客户端.但是,当系统产生分区的时候,主副本节点无法将数据同步至某些备份节点,这将导致不能及时对客户端的写入请求做出响应,因此,系统将不再对外可用.如果主备份节点在主分区上,管理员需要为主分区添加新的备份节点;如果主备份节点在备分区上,管理员需要为主分区添加新的备份节点并从中选择新的主节点.删除备分区中的节点后,新的主分区将继续对外提供服务.

我们可以看到,该类系统与服务级高可用系统均可对外提供最强的一致性.而且,与服务高可用系统相比,人工介入后可用系统不仅在工程实现上更加简单,并且可以避免协商产生的代价,从而提高了系统整体的性能.但是,当其出现分区的时候,需要管理员介入来恢复系统的服务,这大大降低了系统对外的可用性.

综上,对于可用性和一致性的关系,如果系统只提供服务级可用或更低的可用性级别,无论操作一致性或事务一致性都可以得到充分保障.但这样的一致性保障需要以性能为代价.在构建分布式数据库时,系统设计者还需要考虑到性能与可用性或一致性之间的平衡.例如,Spanner 同时保证了较高的可用性和一致性,但复杂的共识协议难免会给它造成性能上的损失.传统数据库系统则削弱了系统的可用性,从而获得了更好的性能.而大部分 NoSQL 数据库则通过放弃一定的一致性来获得可用性与性能的保障.

4 未来研究方向探讨

本文通过一致性和可用性对分布式数据库产品的格局进行了刻画.从这两个维度,我们清晰地看到不同产品之间的差异和差距,以及设计者的考虑和选择.由此,我们也看到一些重要却尚未被充分探索过的领域.对研究者而言,它们都是有价值的未来研究方向.本节选取了 3 个我们认为重要的方向加以阐述.

(1) 如何提高服务级高可用系统的读写以及事务性能.

服务级高可用系统既具备较高可用性,又可以提供强一致的数据服务.相比于其他系统,此类系统在功能上

© 中国科学院软件研究所 http://www.jos.org.cn

具备最高的普适性.但是,由于协商/共识协议本身需要额外的开销,该类系统通常性能有限,妨碍了它成为通用的系统.因此,如何提高服务级高可用系统的性能在最近几年受到了研究者的关注[55],也理应是未来研究的热点.

在跨数据分区的服务级高可用系统架构下,例如 Spanner,分布式事务的并发控制和数据更新的共识协议通常是两个相对独立的模块.事务在提交时,并发控制层需要采用多次交互(如 2PC 协议)来保证事务的原子性,协商层使用 Paxos 需要至少两次网络交互来保证数据的一致性.这使得系统在单次提交分布式事务时就至少需要 4 次网络交互.一方面,是否可以将多个事务进行批量处理,减少系统整体上的网络开销?另一方面,是否可以将 2PC 和 Paxos 的执行流程加以结合,减少单次分布式事务的网络交互次数?这些问题的答案尚不清晰.因此,在服务级高可用系统下,如何减少网络开销从而提升系统性能,将是未来研究者关注的热点.

在同一数据分区内部,我们可以选取具有 leader 角色的副本节点来降低协商/共识的代价.该节点可以由本数据分区内部的所有复本节点通过 Paxos 协议选举出来,并采用租约机制保持身份.但在该机制下,leader 节点将负责所有客户端的读写请求,使得其他副本节点不能被充分利用.一方面,是否可以通过降低一致性级别(例如 Spanner 采用的快照读取操作),使得副本节点无需进行协商即可提供服务[56]?另一方面,是否可以增加处理层次(例如,中间件),使得请求可以直接转发至满足一致性要求的副本节点?这些问题都值得探索.因此,在服务级高可用系统模型下,如何有效地利用副本节点的资源来提高系统整体的性能,也将是未来研究的热点之一.

(2) 可用性的多样化与量化.

数据库系统的高可用性对上层应用而言是非常重要的一个特性[57].例如,在一些电商或银行应用中,系统短时间的不可服务都会造成不可忽视的经济损失,甚至产生较大的社会影响.然而,提升系统的可用性可能会弱化系统的一致性.如上文所述,服务级高可用系统的性能在一致性和可用性方面达到了一定的平衡.一方面,它采用共识算法确保系统在发生局部故障时能够自主地在多个副本间迁移服务,避免了由人工介入来恢复系统服务.另一方面,它不需要牺牲系统的一致性.然而,服务级高可用系统中的协商/共识算法仅能容忍少数副本发生节点故障或者规模较小的链路故障.因此,提升系统的高可用性可以考虑如何让服务级高可用的系统能够容忍半数以上的副本无法访问.这或许会牺牲一定的操作一致性或事务一致性,但这样的牺牲未必是不能容忍的.如何在服务级高可用和节点级高可用之间加以折中?这是一个值得探索的问题.

高可用性的支持通常伴随着较高的性能代价.例如,在服务级高可用系统中,一次操作的修改需要持久化到半数以上副本中才认为提交完成了.从而较大程度地增加了完成一次操作的延迟.多数情况下,配置良好的服务器集群在大部分时间中都会处于正常状态,仅在较短一段时间内会发生局部故障.为了应对小概率的故障,花费巨大的性能代价是否值得?那么,是否能够在一定程度上弱化数据同步的要求来提升一般情况下系统的性能?例如,仅要求数据同步到某个副本的内存中.这可能导致系统在故障时不具备百分之百的自主恢复能力.但是,如果自主恢复在大部分情况下是可行的,则对大部分应用是可接受的.如果在极少数情况下系统服务无法恢复,仍然可以述诸于人工介入.如何在服务级高可用和人工介入后可用之间进行折中?这也是值得研究的.

此外,系统的高可用性目前是一种很难具体量化的指标.本文从故障对系统产生的影响程度这一角度对高可用性进行了一定的分类.然而,对应用而言,它可能更加关心系统在实际部署后能够达到的可用性.通常,应用开发人员可以根据某个系统在真实案例中的表现做出评估.然而,实际系统的可用性是受到故障的类型、发生的频率等因素的影响的.这些因素在不同的硬件环境中都是不同的.从研究的角度,我们是否可以设计出一套对高可用性的基准测试方案?它能够模拟不同类型故障发生的频率,并且统计系统可用的时间.最终,它能够输出一个系统可用性的报告,供开发人员参考.

(3) 如何为应用选择系统.

商业应用对系统一致性、可用性和性能的要求各不相同.例如,很多互联网应用对系统操作一致性和事务一致性要求相对较低,但希望系统的可用性较高;而面向金融的应用常常希望系统既支持较高的一致性又保证高可用.另一方面,即便应用对数据一致性有较高要求,它也可以选择对一致性支持较弱的系统,而通过应用逻辑保证一致性.这让应用既可以收获较高可用性,又可以减少性能损失.如何根据应用的特点从种类繁多的分布

© 中国科学院软件研究所 http://www.jos.org.cn

式数据库系统中做出合理选择?这是大多数软件开发人员面临的一大难题.到目前为止,大多数研究都从分布式数据库设计角度出发探讨了系统可能适合的互联网应用,但这样的论述缺少对应用本身的深入分析,也很难做到包罗万象.对于一个特定的应用,如果它的功能和性能需求已知,那么它在不同系统上的构建方式和构建代价到底如何?目前这方面的知识和经验总结还处于缺失状态.这也是一个重要且艰巨的研究课题.

5 总 结

本文从操作一致性、事务一致性和系统可用性的维度分析和总结了分布式数据库系统的设计空间,并对当今分布式数据库的产品格局进行了刻画.本文的目的并非提出新的概念、想法或技术,而是对现有技术进行梳理,从而对它们的能力边界有一个宏观认识.从这个意义上讲,本文是一篇综述文章,希望能够为当今的数据库系统设计者提供参考.由于分布式数据库系统的研究历史悠久,文献可谓汗牛充栋,我们未能将所有与一致性或可用性相关的技术全部纳入到本文中,甚至可能遗漏掉一些重要文献,也希望读者阅读时持开放、怀疑的态度.

References:

[1] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber R. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems, 2008,26(2):4:1−4:26. [doi: 10.1145/1365815.1365816]

[2] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. SIGOPS Operating Systems Review, 2010,44(2): 35–40. [doi: 10.1145/1773912.1773922]

[3] Plugge E, Hawkins T, Membrey P. The definitive guide to MongoDB: The NoSQL database for cloud and desktop computing. In: Springer Ebooks. Berkely, 2010.

[4] Anderson JC, Lehnardt J, Slater N. CouchDB: The Definitive Guide: Time to Relax. Sebastopol: O'Reilly Media, Inc., 2010.

[5] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. In: Proc. of the 21st SOSP. New York: ACM, 2007. 205−220. [doi: 10.1145/1294261.1294281]

[6] Stonebraker M, Weisberg A. The VoltDB main memory DBMS. IEEE Data Engineering Bulletin, 2013,36(2):21−27.

[7] Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems, 2013,31(3): 8:1−8:22.

[8] Gray J, Reuter A. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers, 1993.

[9] Brewer EA. Towards robust distributed systems. In: Proc. of the PODC. 2000. 7. [doi: 10.1145/343477.343502]

[10] Gilbert S, Lynch NA. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. ACM SIGACT News, 2002,33(2):51−59. [doi: 10.1145/564585.564601]

[11] Vogels W. Eventually consistent. Communications of the ACM, 2009,52(1):40−44. [doi: 10.1145/1435417.1435432]

[12] Adya A, Liskov BH. Weak consistency: A generalized theory and optimistic implementations for distributed transactions [Ph.D. Thesis]. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1999.

[13] Cerone A, Bernardi G, Gotsman A. A framework for transactional consistency models with atomic visibility. In: Proc. of the LIPIcs-Leibniz Int'l in Informatics. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015. 42. [doi: 10.4230/LIPIcs.CONCUR.2015.58]

[14] Bailis P, Fekete A, Hellerstein JM, Ghodsi A, Stoica I. Scalable atomic visibility with RAMP transactions. ACM Trans. on Database Systems, 2016,41(3):15:1−15:45. [doi: 10.1145/2588555.2588562]

[15] Ganesh A, Bamford RJ. Consistent read in a distributed database environment: U.S. Patent 7,334,004[P], 2008-2-19.

[16] Ardekani MS, Sutra P, Shapiro M. Non-Monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In: Proc. of the 32nd SRDS. 2013. 163−172. [doi: 10.1109/SRDS.2013.25]

[17] Thomson A, Diamond T, Weng SC, Ren K, Shao P, Abadi DJ. Calvin: Fast distributed transactions for partitioned database systems. In: Proc. of the SIGMOD. 2012. 1−12. [doi: 10.1145/2213836.2213838]

[18] Davison SB, Molina HG, Skeen D. Consistency in partitioned networks. ACM Computing Surveys, 1985,17(3):341−370. [doi: 10.1145/5505.5508]

[19] Herlihy M, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Trans. on Programming Languages and Systems, 1990,12(3):463−492. [doi: 10.1145/78969.78972]

© 中国科学院软件研究所 http://www.jos.org.cn

[20] Petersen K, Spreitzer M, Terry DB, Theimer M, Demers AJ. Flexible update propagation for weakly consistent replication. In: Proc. of the 16th SOSP. 1997. 288−301. [doi: 10.1145/269005.266711]

[21] Abadi D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. IEEE Computer, 2012, 45(2):37−42. [doi: 10.1109/MC.2012.33]

[22] Bernstein PA, Das S: Rethinking eventual consistency. In: Proc. of the SIGMOD. 2013. 923−928. [doi: 10.1145/2463676.2465339]

[23] Papadimitriou CH. The serializability of concurrent database updates. Journal of the ACM, 1979,26(4):631−653. [doi: 10.1145/322154.322158]

[24] Mohan C, Haderle DJ, Lindsay BG, Pirahesh H, Schwarz PM. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. on Database Systems, 1992,17(1):94−162. [doi: 10.1145/128765.128770]

[25] Malviya N, Weisberg A, Madden S, Stonebraker M. Rethinking main memory OLTP recovery. In: Proc. of the 30th ICDE. 2014. 604−615. [doi: 10.1109/ICDE.2014.6816685]

[26] Yao C, Agrawal D, Chen G, Ooi BC, Wu S. Adaptive logging: Optimizing logging and recovery costs in distributed in-memory databases. In: Proc. of the SIGMOD. 2016. 1119−1134. [doi: 10.1145/2882903.2915208]

[27] Lomet DB, Tzoumas K, Zwilling MJ. Implementing performance competitive logical recovery. The Proc. of the VLDB Endowment, 2011,4(7):430−439. [doi: 10.14778/1988776.1988779]

[28] Wang JH, Cai P, Qian WN, Zhou AY. Log replication and recovery in cluster-based database system. Ruan Jian Xue Bao/Journal of Software, 2017,28(3):476−489 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5162.htm [doi: 10.13328/j.cnki.jos.005162]

[29] Bernstein PA, Hadzilacos V, Goodman N. Concurrency Control and Recovery in Database Systems. Boston: Addison-Wesley. 1987.

[30] Kung HT, Robinson JT. On optimistic methods for concurrency control. In: Proc. of the VLDB. 1979. 351. [doi: 10.1109/VLDB.1979.718150]

[31] Fekete A, Liarokapis D, O'Neil PE, Shasha DE. Making snapshot isolation serializable. ACM Trans. on Database Systems, 2005, 30(2):492−528. [doi: 10.1145/1071610.1071615]

[32] Berenson H, Bernstein P, Gray J, Melton J, O'Neil E, O'Neil P. A critique of ANSI SQL isolation levels. In: Proc. of the SIGMOD. 1995. 1−10. [doi: 10.1145/568271.223785]

[33] Lamport L. Paxos made simple, fast and Byzantine. In: Proc. of the 6th OPODIS. 2002. 7−9. http://dblp.org/rec/bib/conf/opodis/Lamport02

[34] Lamport L. The part-time parliament. ACM Trans. on Computer Systems, 1998,16(2):133−169. [doi: 10.1145/279227.279229]

[35] Chandra TD, Griesemer R, Redstone J. Paxos made live: An engineering perspective. In: Proc. of the PODC. 2007. 398−407. [doi: 10.1145/1281100.1281103]

[36] Gray J, Lamport L. Consensus on transaction commit. ACM Trans. on Database Systems, 2006,31(1):133−160. [doi: 10.1145/1132863.1132867]

[37] Ongaro D, Ousterhout JK. In search of an understandable consensus algorithm. In: Proc. of the USENIX ATC. 2014. 305−319. https://www.usenix.org/conference/atc14

[38] Burrows M. The Chubby lock service for loosely-coupled distributed systems. In: Proc. of the 7th OSDI. 2006. 335−350. http://www.usenix.org/events/osdi06/tech

[39] O'Neil PE, Cheng E, Gawlick D, O'Neil EJ: The log-structured merge-tree (LSM-Tree). Acta Informatica, 1996,33(4):351−385. [doi: 10.1007/s002360050048]

[40] Ghemawat S, Gobioff H, Leung ST. The Google file system. In: Proc. of the 19th SOSP. 2003. 29−43. http://doi.acm.org/10.1145/945445

[41] Baker J, Bond C, Corbett JC, Furman JJ, Khorlin A, Larson J, Leon JM, Li YW, Lloyd A, Yushprakh V. Megastore: Providing scalable, highly available storage for interactive services. In: Proc. of the CIDR. 2011. 223−234. http://cidrdb.org/cidr2011

[42] Lloyd W, Freedman MJ, Kaminsky M, Andersen DG. Don't settle for eventual consistency. Communications of the ACM, 2014, 57(5):61−68. [doi: 10.1145/2596624]

[43] Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978,21(7):558−565. [doi: 10.1145/359545.359563]

[44] Bacon DF, Bales N, Bruno N, Cooper BF, Dickinson A, Fikes A, Fraser C, Gubarev A, Joshi M, Kogan E, Lloyd A, Melnik S, Rao R, Shue D, Taylor C, van der Holst M, Woodford D. Spanner: Becoming a SQL system. In: Proc. of the SIGMOD. 2017. 331−343. [doi: 10.1145/3035918.3056103]

© 中国科学院软件研究所 http://www.jos.org.cn

[45] Samaras G, Britton K, Citron A, Mohan C. Two-Phase commit optimizations and tradeoffs in the commercial environment. In: Proc. of the ICDE. 1993. 520−529. [doi: 10.1109/ICDE.1993.344028]

[46] Thomson A, Abadi DJ. The case for determinism in database systems. The Proc. of the VLDB Endowment, 2010,3(1):70−80. [doi: 10.14778/1920841.1920855]

[47] Stonebraker M, Madden S, Abadi DJ, Harizopoulos S, Hachem N, Helland P. The end of an architectural era (It's time for a complete rewrite). In: Proc. of the VLDB. 2007. 1150−1160. http://www.vldb.org/conf/2007/papers

[48] RenK, Thomson A, Abadi DJ. Lightweight locking for main memory database systems. In: Proc. of the VLDB. 2012. 145−156. [doi: 10.14778/2535568.2448947]

[49] Bailis P, Davidson A, Fekete A, Ghodsi A, Hellerstein JM, Stoica I. Highly available transactions: Virtues and limitations. In: Proc. of the VLDB. 2013. 181−192. [doi: 10.14778/2732232.2732237]

[50] Viotti P, Vukolić M. Consistency in non-transactional distributed storage systems. ACM Computing Surveys, 2016,49(1): 19:1−19:34. [doi: 10.1145/2926965]

[51] Levandoski JJ, Lomet DB, Mokbel MF, Zhao K. Deuteronomy: Transaction support for cloud data. In: Proc. of the CIDR. 2011. 123−133. http://cidrdb.org/cidr2011

[52] Levandoski JJ, Lomet DB, Sengupta S, Stutsman R, Wang R. High performance transactions in deuteronomy. In: Proc. of the CIDR. 2015. http://cidrdb.org/cidr2015

[53] Levandoski JJ, Lomet DB, Sengupta S, Stutsman R, Wang R. Multi-Version range concurrency control in Deuteronomy. The Proc. of the VLDB Endowment, 2015,8(13):2146−2157. [doi: 10.14778/2831360.2831368]

[54] Gray J, Helland P, O'Neil P, Shasha D. The dangers of replication and a solution. In: Proc. of the SIGMOD. 1996. 173−182. [doi: 10.1145/233269.233330]

[55] Mu S, Nelson L, Lloyd W, Li J. Consolidating concurrency control and consensus for commits under conflict. In: Proc. of the OSDI. 2016. 517−532. https://www.usenix.org/conference/osdi16

[56] Zhang I, Sharma NK, Szekeres A, Krishnamurthy A, Ports DRK. Building consistent transactions with inconsistent replication. In: Proc. of the SOSP. 2015. 267−278. [doi: 10.1145/2815400.2815404]

[57] Lin ZY, Lai YX, Lin C, Xie Y, Zou Q. Research on cloud databases. Ruan Jian Xue Bao/Journal of Software, 2012,23(5): 1148−1166 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4195.htm [doi: 10.3724/SP.J.1001.2012.04195]

附中文参考文献:

[28] 王嘉豪,蔡鹏,钱卫宁,周傲英.集群数据库系统的日志复制和故障恢复.软件学报,2017,28(3):476−489. http://www.jos.org.cn/1000-9825/5162.htm [doi: 10.13328/j.cnki.jos.005162]

[57] 林子雨,赖永炫,林琛,谢怡,邹权.云数据库研究.软件学报,2012,23(5):1148−1166. http://www.jos.org.cn/1000-9825/4195.htm [doi: 10.3724/SP.J.1001.2012.04195]

朱涛(1989－),男,江苏苏州人,学士,主要研究领域为事务处理,内存数据库系统.

周烜(1979－),男,博士,教授,博士生导师,主要研究领域为数据库系统,大数据处理技术.

郭进伟(1987－),男,硕士,主要研究领域为分布式数据库.

周傲英(1965－),男,博士,教授,博士生导师,CCF会士,主要研究领域为Web数据管理,数据密集型计算,内存集群计算,分布事务处理,大数据基准测试和性能优化.

周欢(1990－),女,学士,主要研究领域为内存数据库事务处理.

© 中国科学院软件研究所 http://www.jos.org.cn