```
8.          if log_i 没有在磁盘上 then
9.              将 log_i 写入磁盘
10.         else if log_i 在磁盘上但内容不一致 then
11.             清除磁盘上编号≥i 日志
12              将 log_i 写入本地磁盘
13.             break
14.         else if log_i 在磁盘上且内容一致 then
15.             continue
16.         end if
17.         更新已确认的日志号 cfN=i
18.     end for
```

4.3 主节点恢复

如果某个节点被选为了新主节点,则该节点恢复过程与其他节点恢复过程稍有不同.在新主节点上的已提交日志与其他节点上的意义相同,可以直接回放提交.而未提交日志中,一定是旧主节点发来的日志,这些日志可能已经被旧主节点提交了,也可能没有提交,所以为了保险起见,新主机必须等待多数节点确认之后再提交这些日志.因此,新主节点在恢复正常服务之前,先将旧主机的未决事务全部与其他节点进行商榷并提交,保证了旧主已经提交的事务日志不会丢失,具体算法描述见算法 3.

算法 3. 主节点恢复算法.

```
1.      if 已提交日志未全部回放 then
2.          replay_commit_log()   //回放已提交日志
3.      end if
4.      从日志文件中获取 cmN=已提交的最大日志号
5.      从日志文件中获取 lsN=最大日志号
6.      for (i=cmN+1 to lsN) do
7.          do
8.              get 所有节点与主机的确认号 cfN
9.          while (多数个节点的 cfN<i)
10.         提交日志号为 i 的日志
11.     end for
```

5 一致性分析

表 1 所示的主节点负责事务处理,是系统中唯一的写节点,在主节点上的读写都是强一致性的,但是负载相对较高,吞吐量较低;而非主节点上的日志回放在主节点之后,因此读取非主节点机的数据可能会导致读出旧数据,而最终主节点会将提交信息发给非主节点,最终非主节点会与主节点保持一致,因此读取非主节点上的数据是弱一致性,而非主节点负载相对较低,因此吞吐量可以达到很高.当系统出现异常导致节点数不足多数时,此时没有主节点,所有节点只能提供弱一致性的读服务.由于基于 Paxos 系统中通常会配备 3 个节点或者 5 个节点,所以多数节点故障的几率是比较少见的,即使出现也不会对一致性造成影响.

本文的日志复制实现基于 OceanBase 数据库开源的 0.4.2 版本,该数据库主要包含 4 个部分:RootServer, UpdateServer,MergeServer 和 ChunkServer.其中:RootServer 是集群的管理者;UpdateServer 是内存数据库,维护着系统的增量数据,也是事务的处理中心;ChunkServer 存放基线数据;MergeServer 接受客户端的连接,并将请求分发到 UpdateServer 和 ChunkServer,合并增量数据和基线数据,并提供负载均衡.由于 MergeServer 是专门负责处理 SQL 请求的节点,该节点可以根据不同的应用提供不同的一致性:当需要强一致性读取时,将读请求发给主节

点;当使用弱一致性读时,将读请求发给非主节点.大多数业务对读的一致性要求不高,因此这种机制有很好的负载均衡功能,有助于提高数据库的扩展性,使得主节点不会成为性能瓶颈.

Table 1 Tradeoff in consistency and throughput

表 1 一致性与吞吐量的取舍

	服务	读主节点	读备节点
多数节点存活	写/读服务	强一致性 低吞吐量	弱一致性 高吞吐量
不足多数节点存活	读服务	– –	弱一致性 高吞吐量

6 实验评估

本文将日志同步策略和恢复机制实现于 OceanBase 系统中 UpdateServer 内存数据库,实验分为两部分:第 1 部分是在不同的读写比例、网络等情况下,对吞吐量、恢复时间等性能指标的测试分析;第 2 部分是与未使用本文技术的 OceanBase 系统中 UpdateServer 的对比测试.其中,第 1 部分还分为正常情况下的测试和异常情况下的测试.

6.1 实验环境

第 1 部分实验基准测试工具 YCSB[21]的 0.7.0 版本进行测试,使用了 OceanBase 的一个集群,集群中有 3 台 UpdateServer,1 台 RootServer,6 台 ChunkServer 和 6 台 MergeServer.其中,3 台 UpdateServer 内存数据库使用了本文提出的日志复制策略.每台机器配置了两颗 Intel(R)Xeon(R)E5606@2.13GHz 型号的 4 核心 CPU 以及 100GB 内存和 100GB 的 SSD 盘.

第 2 部分实验是使用本文提出技术前后的对比测试,在某银行的生产测试环境中进行,使用了基准测试工具 Sysbench 0.5 版本,搭建了原 OceanBase0.4 版本的 1 个集群和本文改造后的 1 个集群.集群中有 3 台 UpdateServer,1 台 RootServer,4 台 ChunkServer 和 4 台 MergeServer.每台机器配置了两颗 Intel(R)Xeon(R)E5-2650 v3@2.30GHz 型号的 10 核心 20 线程的 CPU 以及 100GB 内存和 4T 的机械硬盘.两个集群配置相同,只是其中 UpdateServer 使用的日志复制策略不同,原 OceanBase0.4 版本集群中使用的是主备日志同步策略,本文改造后的集群中使用了本文提出的日志复制策略.

6.2 实验方法与结果分析

- 正常情况测试

实验 1:评估不同写负载情况下,强一致性和弱一致性的吞吐量和延迟.其结果如图 6 所示,将写负载所占比例分成 10%~50%这 5 个梯度(剩下的比例为读负载),使用 YCSB 的 200 个线程并发请求.

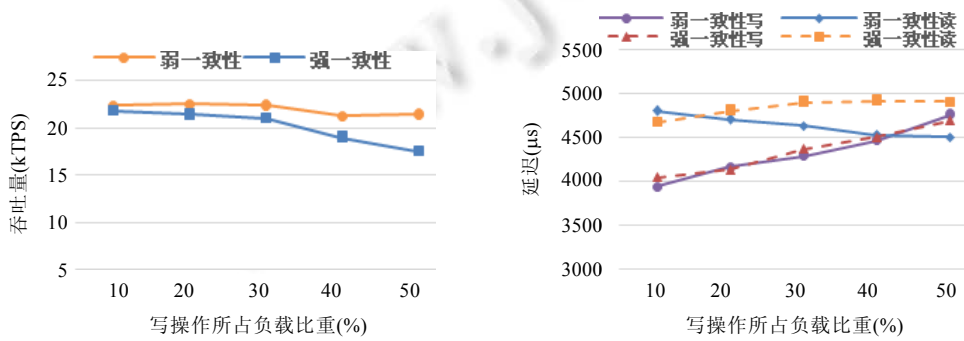

Fig.6 Throughput and latency of two model under different workloads

图 6 两种一致性模在不同写负载比例下的吞吐量和读写延迟

在吞吐量方面,在写比例较少时,强一致性与弱一致性的吞吐量相当;随着写负载比例的增加,强一致性模型的吞吐量会稍有下降,而弱一致性表现比较平稳.在读写延迟方面,写负载的增加将会影响强一致性的读延迟升高,而弱一致性的读延迟随着写比例的上升会下降.这是由于弱一致性读事务分担到了每个节点上,因此性能较好.而在写事务上,由于无论哪个一致性模型都会将写操作作用于主节点,因此写操作的延迟和性能几乎相同.由此可见:选择弱一致性读平衡了读操作的负载,能够提高系统的吞吐量,降低延迟.

实验 2:评估系统的扩展性.对比不同客户端的连接数的情况下,两种一致性的吞吐量,负载使用的是 YCSB 的 workloadb(读 95%写 5%).其结果如图 7 所示:随着客户端连接数量的增加,强一致性的吞吐量逐渐达到瓶颈,而弱一致性吞吐量继续呈增长趋势.这是由于弱一致性读操作将会分担到每个节点上,扩展性相对强一致性读较好.Raft 算法中采用了强一致性的读,而本文采用的是扩展的 Raft,可以使用强弱两种一致性,通过实验看出弱一致性具有较好的扩展性.因此,本文实现的扩展 Raft 与原算法相比有较好的扩展性.

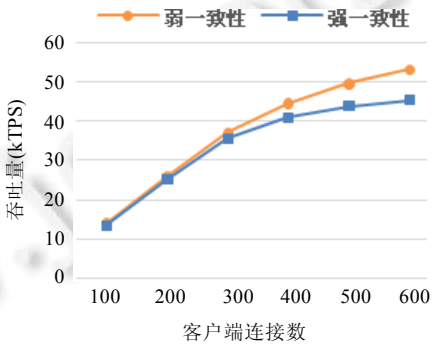

Fig.7 Scalability of two model

图 7 两种一致性模型的扩展性

- 异常情况测试

实验 3:评估系统在不同的负载比例下,将某个非主节点网络中断一段时间之后再恢复,记录该节点的恢复时间.负载使用 6 个梯度,写操作分别所占比例 0~50%,其结果如图 8 所示:写负载越重,网络中断时间越长,恢复时间也越长.这是由于在故障期间,该节点上网络中断,无法收到主节点的日志,而写比例越重的负载,随着时间增加,该节点与主节点日志差距越大,该节点恢复时需要从主节点拉取缺失的日志,因此恢复时间就越长.在写负载为 50%时,网络故障 60s,恢复时间大约为 80s,系统能够较快恢复.

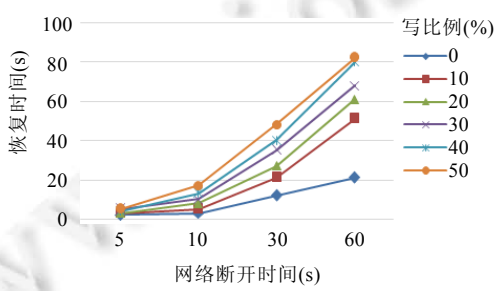

Fig.8 Recovery time in the scenario of network disconnected

图 8 网络故障恢复时间

实验 4:评估主节点故障后,重新选主的效率.在系统服务过程中,将主节点进程杀掉,模拟主节点故障的情况,测试主节点的切换时间和选主时间.其结果如图 9 所示,恢复时间大约在 25ms 左右.其中,主要时间用于切换的开销,约 20ms;选主算法所用时间在 2ms~5ms 之间,节点越少,选主时间越短.与 Raft 算法相比,由于去掉了投

票时间,因此选主用时比 Raft 算法短,而且用时相对稳定.

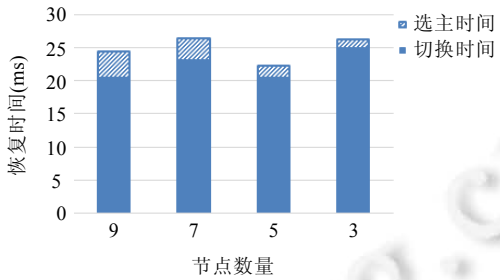

Fig.9 Master recovery time

图 9 主节点恢复时间

实验 5:评估不同的网络延迟和丢包率的情况下,日志复制的效率.通过使用工具人为增加网络延迟和丢包率,在 YCSB 的 workloada 负载(读 50%写 50%)下,测试系统的吞吐量.在网络延迟增加的情况下,日志复制延迟变高,而事务提交需要等待其日志至少复制给一个节点,因此,系统事务处理速度会变慢.而丢包会影响日志复制,有些日志包会丢失需要非主节点主动获取缺失日志,因此也会影响系统的事务提交速度.实验结果如图 10 所示:在网络延迟小于 1ms 和丢包率小于 5%时,对性能影响不大;当网络延迟超过 10ms 或者丢包率超过 10%,对系统有较大影响.因此,本文的日志复制策略能够抵御一定的网络延时和丢包,使得系统吞吐率不会显著下降,提高了系统的可用性.

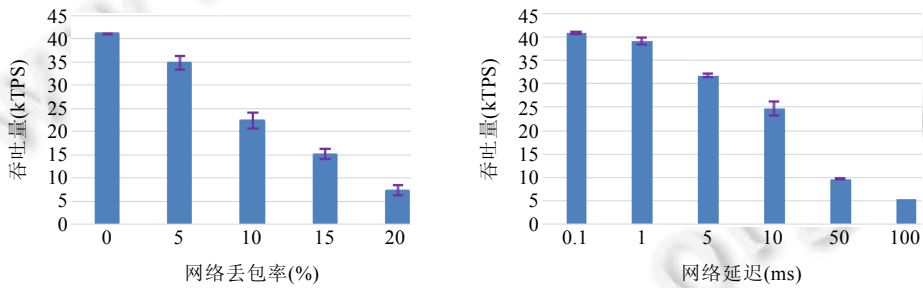

Fig.10 Throughput in the scenario of different network latency and package loss

图 10 不同网络延迟和丢包率的情况下系统吞吐量

实验 6:验证系统在高负载下的可用性.实验使用了 YCSB 的 workloada 负载,如图 11 所示:在系统平稳运行一段时间(0s~50s)后,在某一时刻(50s)杀死主事务处理进程,系统的吞吐量迅速降低为 0,并在随后进行了租约过期检测、重新选主和切换这 3 个步骤;4s 之后(54s),新的主事务处理节点恢复完成,系统重新开始正常运行.由此证明了在较高负载下,本文的恢复机制能够检测到异常并快速恢复,有较高的可用性.

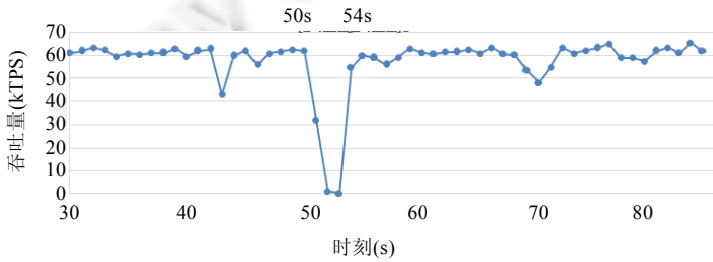

Fig.11 Impact of master transaction processing node failure under high workload

图 11 高负载下主事务处理节点异常对系统的影响

- 对比测试

实验 7:将本文的日志复制策略与原主备日志同步策略的性能进行对比,如图 12 所示.从图中可以看出:在高读负载下,对于相同的读一致性策略,两种日志同步策略性能几乎相同.这是因为日志复制策略对读事务没有影响;而在高写负载下,本文提出的日志复制策略相比原来的同步策略性能有所下降,大约下降了 16%.这是由于相比之下,原主备同步只需要将日志写入本地的主节点,事务便可以提交;而为了提供了可靠的异常恢复机制,本文的日志复制策略需要确保多数节点将日志写入磁盘后,事务才能提交,因此事务执行的时间变长,吞吐量受到了影响.但是这样做带来的好处是保证了提交事务日志不会丢失,提高了系统的可用性和一致性,牺牲了一小部分性能是值得的.

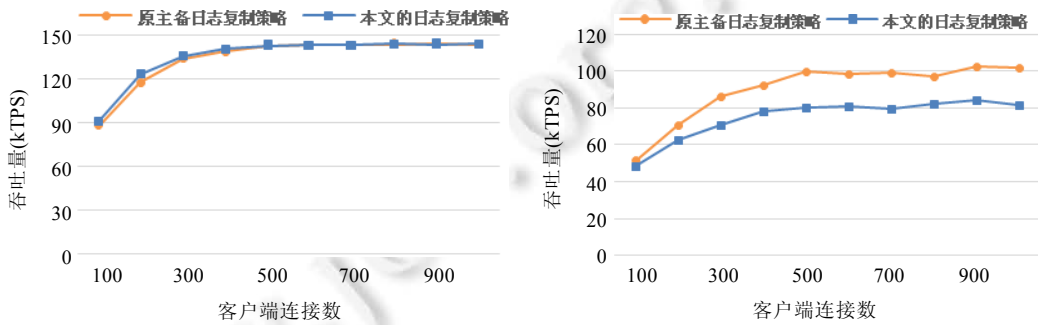

Fig.12 Comparison between our log replication strategy and master-slave strategy

图 12 本文日志复制策略与原主备日志复制策略的性能对比

7 结 论

本文提出了一种集群环境下分布式数据库中的日志复制和恢复机制,能够在故障情况下保证事务日志的完整性和一致性,实现了强弱两种一致性读策略,提高了系统的扩展性.此外,本文提出了一种轻量级的选主算法,能够快速稳定地选主,避免了经典一致性算法选主算法中活锁、双主和频繁选主等问题,提高了系统的可用性.最终,通过实验分析各种异常情况下对系统性能的影响,证明了日志复制和恢复的高效率.

本文设计的日志复制和恢复机制已经应用于某银行数据库系统中,日志复制和恢复算法具有很好的通用性,因此也容易应用于其他类似的分布式事务系统.本文只实现了在非主节点弱一致性读的策略,而如何在非主节点实现强一致性读,是今后研究的重点.

References:

[1] Özsu MT, Valduriez P. Principles of Distributed Database Systems. 2nd ed., Berlin: Springer-Verlag, 1999.

[2] Yang ZK. The architecture of OceanBase relational database system. Journal of East China Normal University (Natural Science), 2014,9(5):141−148,163 (in Chinese with English abstract).

[3] Lamport L. Paxos made simple. ACM SIGACT News, 2001,32(4):18−25.

[4] Burrows M. The Chubby lock service for loosely-coupled distributed systems. In: Proc. of the Symp. on Operating Systems Design and Implementation. USENIX Association, 2006. 335−350.

[5] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. Stanford University, 2013.

[6] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems, 2008,26(2):205−218.

[7] Yang CH. Large Scale Distributed Storage System. Beijing: China Machine Press, 2013 (in Chinese).

[8] Rao J, Shekita EJ, Tata S. Using Paxos to build a scalable, consistent, and highly available datastore. Computer Science, 2011,4(4): 243−254.

[9] Thalmann L, Ronstrom M. MySQL Cluster Architecture Overview. 2004.

[10] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Mruthy R. Hive□□A warehousing solution over a map-reduce framework. Proc. of the VLDB Endowment, 2011,2(2):1626−1629.

[11] Chandra T, Griesemer R, Redstone J. Paxos made live: An engineering perspective. In: Proc. of the 26th ACM Symp. on Principles of Distributed Computing (PODC 2007). 2007. 398−407. [doi: 10.1145/1281100.1281103]

[12] 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. ACM Sigops Operating Systems Review, 2007,41(6):205−220.

[13] Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R, PNUTS: Yahoo!'s hosted data serving platform. Proc. of the VLDB Endowment, 2015,1(2):1277−1288.

[14] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. ACM Sigops Operating Systems Review, 2010, 44(2):35−40.

[15] 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 HY, 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):251−264.

[16] Skeen D. A quorum-based commit protocol. In: Proc. of the Berkeley Workshop on Distributed Data Management & Computer Networks. 1982. 69−80.

[17] Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. ACM Trans. on ACM Sigact News, 2002,33(2):51−59.

[18] Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: Wait-Free Coordination for Internet-scale Systems. 2010. 653−710.

[19] Robertson A, Robertson A. The Evolution of the Linux-HA Project. 2004.

[20] Gray C, Cheriton D. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. ACM Sigops Operating Systems Review, 1989,23(5):202−210. [doi: 10.1145/74851.74870]

[21] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In: Proc. of the ACM Symp. on Cloud Computing. ACM Press, 2010. 143−154. [doi: 10.1145/1807128.1807152]

附中文参考文献:

[2] 阳振坤.OceanBase 关系数据库架构.华东师范大学学报(自然科学版),2014,9(5):141−148,163.

[7] 杨传辉.大规模分布式存储系统.北京:机械工业出版社,2013.

王嘉豪(1993－),男,硕士生,主要研究领域为海量数据管理与分析,分布式数据库.

蔡鹏(1978－),男,博士,副教授,主要研究领域为可扩展高性能事务处理.

钱卫宁(1976－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为互联网环境下的数据管理,大数据管理系统评测基准,社交媒体数据分析,知识图谱构建与应用.

周傲英(1965－),男,博士,教授,博士生导师,CCF 杰出会员,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.