文章编号:1000-5641(2015)05-0172-13

基于 Raft 一致性协议的高可用性实现

张晨东, 郭进伟, 刘柏众, 储佳佳, 周敏奇, 钱卫宁
(华东师范大学数据科学与工程研究院,上海 200062)

摘要: 随着互联网的快速发展和大数据时代的来临,传统数据库的局限性开始逐渐显现,而支持海量数据存储和高并发访问的分布式数据库系统越来越流行.在此背景下阿里巴巴集团研发了一款适用于海量数据存储的分布式数据库系统(OceanBase),并提供单集群和多集群两种部署模式.但多集群部署模式下的可用性较低,无法满足关键性应用的需求,包括:发生故障时不支持主备集群的自动切换;主备集群之间无法保证日志的强同步.针对上述问题,本文分析了传统数据库的高可用方案,针对 OceanBase 架构的特点,结合了 Raft 算法的思想,设计并实现了基于时间戳的分布式选举模块、自动化的集群切换模块和基于 QUORUM 策略的日志强同步模块.经实验验证,以上模块的实现能够提高系统整体的可用性.

关键词: 分布式数据库; 高可用性; Raft 一致性协议; 日志同步
中图分类号: Q948 **文献标识码**: A **DOI**:10.3969/j.issn.1000-5641.2015.05.015

High availability implementation based on Raft

ZHANG Chen-dong, GUO Jin-wei, LIU Bo-zhong, CHU Jia-jia,
ZHOU Min-qi, QIAN Wei-ning
(*Institute for Data Science and Engineering at East China Normal University*, *Shanghai* 200062, *China*)

Abstract: With the rapid development of Internet and the up-coming Big Data era, the limitation of traditional database has been emerged and enlarged. The distributed database system based on massive data storage and high concurrent accesses has become more and more popular. Alibaba group developed a distributed database system suitable for mass data storage named OceanBase, which supports two deployment modes, i.e., single cluster and multiple clusters. But the availability of multiple clusters mode is not efficient and can't satisfy the requirement of some critical applications, where it does not support the automatic switch between master cluster and slave cluster when a failure occurred and the inconsistent log is also generated during switching under multiple clusters mode. To address these problems, we analysis the high availability solutions of the traditional database, aiming at the characteristics of OceanBase architecture, combining the idea of in Raft, and then designs and implements the distributed election module based on the timestamp of logs, the automatic clusters switching module and the strong synchronization logs module based on QUORUM. The experimental results showed that the above ap-

收稿日期:2015-06
基金项目:国家自然科学基金重点项目(61332006);863 项目(2015AA015307)
第一作者:张晨东,男,硕士研究生,研究方向为分布式数据库. E-mail: 51131500048@ecnu.cn.
通信作者:周敏奇,男,副教授,硕士生导师,研究方向为内存数据库. E-mail: mqzhou@sei.ecnu.edu.cn.

proachescould improve the availability of the whole system.

Key words: distributed database; high availability; Raft consistency protocol; log synchronization

0 引　　言

数据库管理系统是存储和访问具有内在关系的数据的系统软件，例如 DB2，Oracle，SQLServer 等数据库系统，它们被广泛应用于银行、商品交易、电信、电力等对可用性和可靠性要求非常高的领域，在生产和生活中起着至关重要的作用[1].

随着移动互联网的普及和发展，数据库的数据量和访问量呈指数级增长，传统数据库已难以满足高并发、海量存储的需求. 因此许多互联网公司将高可扩展的分布式技术引入到数据库系统中，研发出许多分布式数据库系统，例如 Google 的 Bigtable[2]，Spanner[3]，Yahoo! 的 Dynamo[4] 系统等. 同时一批开源的 NoSQL 分布式数据库也逐渐开始流行起来，例如 HBase、MongoDB 等，这些数据库系统通过分布式的数据存储机制保证了数据的可靠性和可扩展性，使用了一些常见的容错机制来保证系统的可用性.

技术发展的同时，信息安全也越来越被人们重视. 作为存取数据的基础软件设施，数据库管理系统在金融、政府等机构的信息系统中处于至关重要的地位. 然而数据库系统产品的市场长期被国外垄断，在信息安全方面存在着巨大的隐患，这种现状促使国产软件的发展上升为了国家战略[5].

在这样的市场需求和国家战略的背景下，阿里巴巴集团推出了一款支持海量数据存储的高性能分布式数据库系统 OceanBase[6]. 它实现了数千亿条记录、数百 TB 数据上的跨行跨表事务，能满足当前海量存储需求的应用，在可扩展性、可用性和数据一致性方面都有非常好的表现，并且已经成功应用到了阿里集团内部很多业务系统中. OceanBase 的高可用性并没有依赖于底层的高可用硬件，而是采用了价格低廉的普通 PC 服务器，在软件层面上实现了系统内部故障对外部使用者不可见，这种方式可以在不牺牲强一致性和高可用性的情况下有一个低成本的使用方案[7].

OceanBase 最新的开源版本(0.4.2)有两种集群部署模式：单集群模式和多集群模式. 单集群模式下所有的机器都必须在一个机房内，它没有防范区域性自然灾害的能力. 多集群模式下，各个集群可以部署在不同地区的机房内，具有容灾的能力，但这种部署模式的可用性并不高. 当主集群出现故障导致系统不可用时，必须通过人工介入的方式来将备集群切换为主集群. 即使人工介入了也有可能发生由于日志不一致而导致的主备集群切换失败的情况，这是由于主备集群间的不可靠日志同步策略导致的.

针对 OceanBase 的多集群模式的可用性不高的缺陷，本文基于 Raft 一致性协议[8]，设计并实现了一些功能来提高它的可用性：

①实现了基于日志时间戳的选主协议，使之适用于 OceanBase 的使用场景；

②实现了自动切换主备集群的功能，当系统得到选举结果后，能够实时感知到主备集群的状态，从而自动切换主备集群，保证了数据库服务的持续可用性；

③设计并实现了基于 QUORUM 策略[9] 的日志强同步技术，在日志同步过程中兼顾了同步日志的效率和数据的可靠性，在多集群的部署模式下日志同步能够保证集群的异常恢

复机制,最大化整个系统的可用性,并且保证了在异常恢复的过程中数据的一致性和完整性;

实验证明基于日志时间戳的选主协议以及集群切换过程是正确且高效的,日志同步能够保证数据的完整性和一致性.

本文第一部分介绍了传统数据库高可用方案以及 OceanBase 的架构和相关一致性算法背景.第二部分介绍了选主协议实现、主备集群切换的实现以及 UpdateServer 的日志强同步的实现.第三部分对这些已实现的功能进行测试实验.第四部分是本文的总结以及对未来的发展方向的阐述.

1 相关工作

可用性是指系统在面对各种异常时依然可以提供正常服务的能力.数据库系统一般采用冗余硬件和数据的策略来实现高可用性.然而,数据一致性问题随之产生.以下将分三个部分介绍本文的相关工作.

1.1 传统数据库的高可用方案

作为一个典型的传统数据库,Oracle 提供了以下两种高可用解决方案[10]:真正应用集群(Real Application Cluster,RAC)[11],数据卫士(Data Guard,DG)[12-13].

在 RAC 方案中,为了达到最高级别的服务器保护模式,一般有两台以上的服务器架构在共享存储设备之上,所有服务器均可直接访问共享存储中的数据.这种方式使得在低成本服务器上构建高可用性数据库系统成为可能,但是 RAC 方案的缺陷在于系统的可用性受制于共享存储的可用性.

在 DG 方案中,一个主服务器提供服务,多个备服务器作为热备份,主服务器需要将日志数据实时同步到备服务器上.DG 方案提供了三种数据保护模式:

(1) 最大保护模式:任何事务日志必须先成功同步到所有备服务器上,最后才会在主服务器上执行成功,然后应答客户端成功;

(2) 最高性能模式:任何事务只需在主服务器上执行成功,就可以应答客户端成功;

(3) 最大可用模式:最大可用模式介于前两者之间,在正常情况下,它和最大保护模式一样,但一旦备服务器出现故障,就立即切换成最高性能模式,但主服务器不会关闭.

在最大保护模式下,当备服务器与主服务器间的网络出现故障时,系统将无法提供服务,失去可用性.在最高性能模式下,主备服务器之间的数据无法保证一致,当备服务器切换为主服务器时,无法保证可靠性.在最大可用模式下,当主服务器出现故障时,备服务器无法保证数据的强一致性.

1.2 OceanBase 的高可用方案

不同于传统的数据库,OceanBase 是在分布式的网络环境下实现了高可用性[14-15].

根据系统中的功能和角色不同,可以将 OceanBase 分为四个模块:主控服务器(RootServer)、更新服务器(UpdateServer)、基线服务器(ChunkServer)以及合并服务器(MergeServer).各个模块通过网络传输数据,其中 RootServer 负责集群中的机器管理和数据分布的管理,UpdateServer 存储增量数据,ChunkServer 存储基线数据,MergeServer 负责 SQL 解析和执行并对外提供服务.

在 OceanBase 系统中,增量数据存储在 UpdateServer 的内存中,基线数据存储在 ChunkServer 的磁盘上,当增量数据达到一定阈值或者在每天的固定时刻会发生数据合并,

将增量数据合并到基线数据上. 这种将基线数据和增量数据分离开的方式减少了写放大对系统性能的影响，也提高了数据的可靠性和系统的可用性[6].

OceanBase 可以部署为单集群模式，也可以部署为多集群模式. 当 OceanBase 部署为多集群模式时，如图 1 所示，这些集群可以部署在不同地区的机房中，在发生自然灾害或者整个机房发生停电等大规模区域性故障时，仍然可以保证数据的安全和数据库服务的可用性. 在多集群模式下，有一个主集群和多个备集群，每个集群中可以有一主一备两个 RootServer 和 UpdateServer 保证单集群内的可用性. 主集群可以对外提供强一致性的查询服务，备集群也可以对外提供查询服务，但是只能提供弱一致性的查询服务. 多集群在运行的时候，主集群一旦无法提供服务，需要通过人工手动的方式将备集群切换为主集群. 这种人工手动切换的方式风险大，耗时长，不能保证系统的持续可用性.

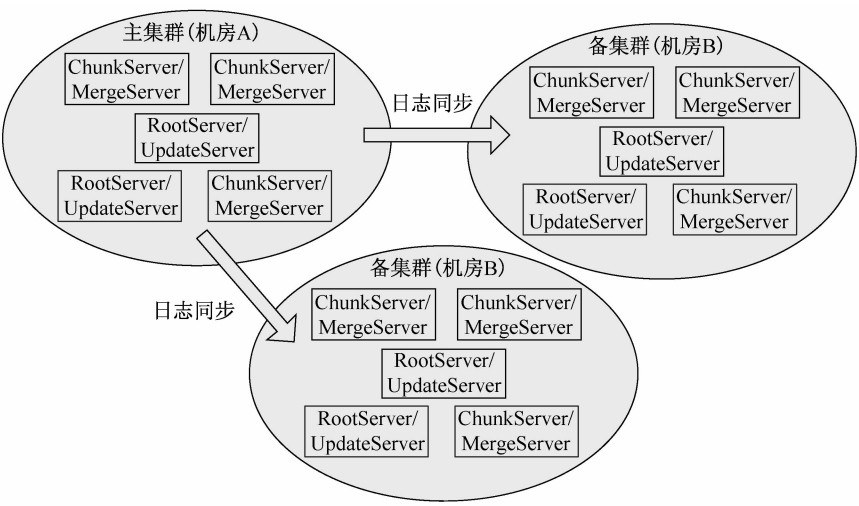

图 1　OceanBase 多集群模式的系统架构

Fig. 1　The system architecture of OceanBase under multi-clusters

1.3　分布式系统一致性算法

数据一致性是指各数据副本的数据总是一致的，表现在同一客户端下，读操作是否总能读取到最新完成的写操作结果. 而在分布式系统中，数据的一致性尤为重要. 为此，许多研究者为解决分布式环境下数据一致性问题，针对性地提出多种算法，其中较为典型的有 Paxos 一致性算法和 Raft 一致性算法.

(1) Paxos 一致性算法

Paxos 算法[16]是 Leslie Lamport 于 1990 年提出的一种基于消息传递模型且具有高度容错特性的一致性算法，目的是解决如何在一个不可靠的网络环境和不可靠的节点之间就某个值达成一致. 在一个分布式数据库系统中，不考虑消息传递过程中可能出现的拜占庭错误，如果各个节点的初始状态是一致的，在经过相同的操作序列后，它们最终一定能达到一致的状态. 为了保证每个节点都能经过相同的操作序列，每执行一条操作指令，都要使用一次 Paxos 算法来达成一致.

上述描述的是 Basic Paxos. 除此之外，还有很多基于 Paxos 的变种，如 Multi-Paxos[17]，Cheap Paxos[18]，Fast Paxos[19]等，它们都对 Basic Paxos 算法进行了不同程度的优化. Pax-

os 算法虽然能解决分布式环境下多个节点就某个值达成一致的问题,但是算法本身的逻辑过于复杂,很难被人理解,实现十分困难.

(2) Raft 一致性算法

针对 Paxos 难于被理解的问题,斯坦福大学的 Diego Ongaro 和 John Ousterhout 提出了一个管理日志复制的一致性算法 Raft[7],它是由 Paxos 算法演化而来的,它的首要设计目标就是为了使得它易于理解和实现.

为了加强算法的可理解性,Raft 将达成一致性的关键步骤分离开,例如 Leader 的选举、日志复制、安全性保证等这些关键步骤,通过减少需要考虑到的状态来保证了更强的一致性. Raft 的可理解性经过实践验证是有效果的,它能很容易被理解和实现.

2 问题定义

由于 OceanBase 开源至今已先后发布多个版本,未来可能还会有新的版本发布,为避免混乱,本文中所提及的 OceanBase 版本特定为 0.4.2 版本. 该版本的 OceanBase 系统没有自动选主及自动切换主备集群的功能,并且日志的同步策略会引起主备切换发生数据不一致的问题.

OceanBase 为了能够支持异地容灾,提供了多集群部署模式. 当主集群出现故障时,需要选择一个备集群切换为主集群,此过程通过人工判断,一般选取数据最新的为主集群. 由于切换过程是由人工手动去执行的,这是一个较为耗时并且有操作失误风险的过程.

OceanBase 主备集群的日志同步策略会引起手动切换主备不成功的问题. OceanBase 的数据分为基线数据和增量数据,由于基线数据是由增量数据转换得到的,所以 OceanBase 要保证全量数据的同步只需实时同步增量数据即可,由主集群中的 UpdateServer 将增量数据以日志的形式发送到各个备集群中 UpdateServer 上. 日志同步策略类似于 Oracle Data-Guard 方案中的最大性能模式,同步日志时并不需要保证任何一个备 UpdateServer 回放日志成功,这种日志同步策略会使主 UpdateServer 的日志数据经常会超前于备 UpdateServer 的日志数据,当主 UpdateServer 发生故障宕机时,备 UpdateServer 的数据会落后于发生宕机的 UpdateServer,从备机中重新选出的主 UpdateServer 必然会丢失一部分数据,此时如果重启原先宕机的 UpdateServer,就会发生数据不一致的问题.

OceanBase 在设计多集群部署模式时并没有考虑到主集群不可用时人工介入的不足,日志同步的模式也会导致主备集群的切换会发生数据不一致的问题,因此 OceanBase 无法应用于对可用性要求较高的场景. 考虑到 OceanBase 本身架构的特点,并结合 Raft 算法的思想,本文实现了:(1)基于日志时间戳的选举协议;(2)基于 QUORUM 策略的日志强同步技术.

3 高可用方案的实现

为了提高 OceanBase 的可用性,我们在深入了解 OceanBase 系统架构的基础上,借鉴了 Raft 算法的思想,设计并实现了如下的一些功能:

(1) 基于日志时间戳的选主协议,使它能够在多集群模式下的实现成为可能;

(2) 全自动化的集群切换功能,当 Raft 选举出新的主后能够快速自动切换主备集群;

(3) 基于 QUORUM 策略的日志强同步技术,保证了数据的完整性和可靠性.

在 OceanBase 0.4.2 版本的基础上,本文设计了如图 2 所示的集群架构,并实现了如图 3 所示的集群自动切换过程.

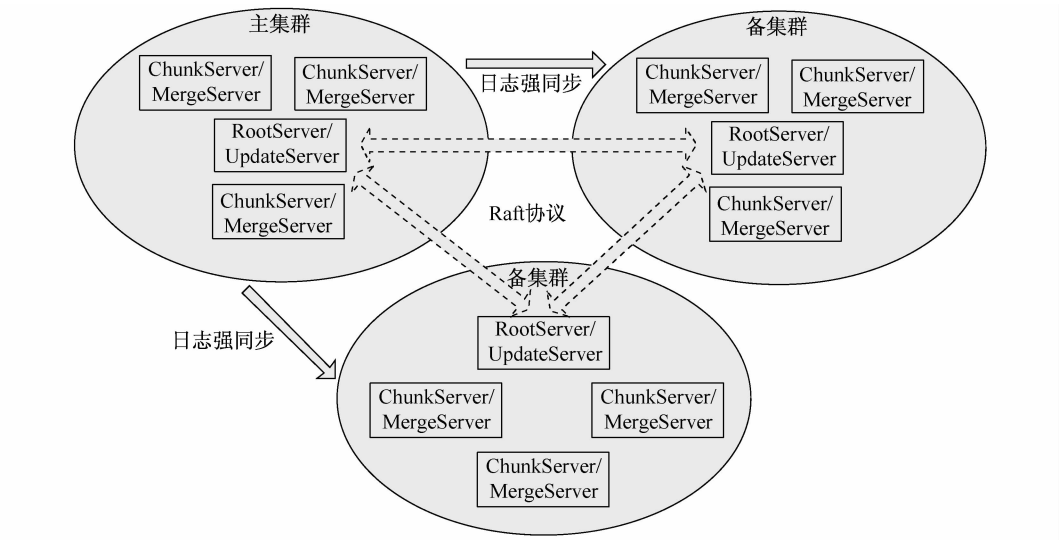

图 2　多集群架构

Fig. 2　The architecture of multi-clusters

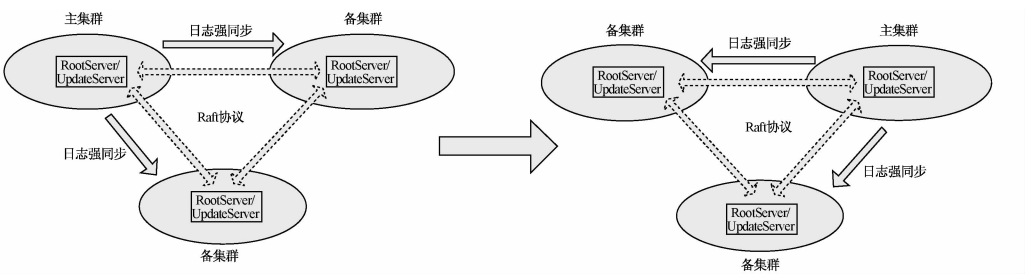

图 3　集群切换过程

Fig. 3　The process of clusters-switching

在图 2 所示的多集群架构中,每个集群中有且仅有一个 RootServer 和一个 UpdateServer,并且多集群中只有一个主集群.其中在主集群中的 RootServer 和 UpdateServer 都是主机的角色,备集群中的 RootServer 和 UpdateServer 都是备机的角色.

主备集群中的所有 RootServer 之间通过选举协议保持始终只有一个主 RootServer 存在,当主机发生故障时会有一个新的主 RootServer 被选举出来,并发生集群角色的相应切换过程(图 3).切换过程中的数据一致性和可靠性是由日志强同步来保证的.

3.1　基于日志时间戳的选主协议

为了保证系统的持续可用性,多集群模式下必须有一个主集群存在,当主集群出现问题不可访问时,应当有一个新的主集群能够快速被选出来并提供服务,这就需要通过选举过程的一致性算法来实现.我们选择 Raft 一致性协议算法作为基础,对它进行了定制,使它适用于 OceanBase 系统,在 RootServer 模块中实现了基于日志时间戳的选举功能.

基于 Raft 一致性协议中的描述,我们为每个 RootServer 指定以下三种角色的其中之

一:领导者(Leader),跟随者(Follower),候选者(Candidate)[7]. 领导者是全局唯一的,包含领导者的集群就是主集群,在这个集群中的 UpdateServer 是主 UpdateServer,并且是全局唯一的主 UpdateServer,它是所有集群中唯一能够接受写操作的 UpdateServer. 跟随者能够接收领导者发送过来的日志,或者给满足条件的候选者投票. 候选者会主动发起选举,角逐领导者的角色,当跟随者经过了随机等待时间后仍然没有选出主,则它会将自己变为候选者,并向其他节点发送拉票请求.

当 RootServer 刚启动时,它的选举模块的功能不会开启,必须通过人工手动开启或者通过接收其他 RootServer 的选举广播消息来开启,可以由数据库管理员来决定第一次启动时主所在的机器,便于管理员的管理. 当所有的 RootServer 都是刚启动的状态,此时它们的选举模块都是被屏蔽的,并且没有主 RootServer 存在,这时需要人工手动指定主 RootServer 来打开所有的 RootServer 的选举模块功能. 在此之后若有 RootServer 发生故障重启,不论它重启之前是主 RootServer 还是备 RootServer,都会接收到其他的 RootServer 的广播消息,从而开启选举功能.

只有选举功能是开启的 RootServer 才能进行正常的选举过程. 选举的流程与 Raft 一致性算法中描述的相同,每个跟随者必须先各自随机等待一段时间才能发起选举,这是为了减小选举冲突导致选举失败的概率. 当跟随者在这段时间内没有收到主 RootServer 的广播信息,那么它会变为候选者向其他 RootServer 发起投票,当接收到选票数超过半数则当选为领导者,否则重新变为跟随者. 与 Raft 算法不同的是,其他 RootServer 接收到投票请求时会比较 UpdateServer 的最大日志时间戳,以此作为条件来判断是否同意该候选者成为领导者. 在每个集群中的 UpdateServer 上都会记录最新数据的日志时间戳,时间戳越大表示数据越新,若候选者的最大日志时间戳大于自己的最大日志时间戳,则表示候选者的数据比自己的数据更新,同意该投票请求,若候选者的最大日志时间戳小于自己的最大日志时间戳,表示候选者的数据旧于自己的数据,则拒绝该投票请求.

当一个节点成为领导者之后,它和其他每个节点之间都会维护一份租约信息,包括它自己,在租约快要到期的时候会给每个节点续约. 在这个租约有效期内跟随者会一直认为该领导者是有效的,如果超过了这个租约有效期并且一直未成功续约,可能是由于领导者宕机、跟随者宕机或者是连接的网络出现中断故障等原因造成的. 如果是第一种情况,领导者宕机,跟随者会随机延迟一段时间然后重新发起选主. 如果是第二种情况,跟随者宕机,这时领导者会检查当前仍然在线的跟随者的数量,若少于一半就放弃主的身份,与其他在线的跟随者节点重新选主. 如果是第三种情况,网络连接出现中断,这时领导者会在自己与自己的租约时间到期后检查当前和它保持在线的跟随者数量是否超过半数,如果少于半数则自动放弃主的身份,重新选主.

如图 4 所示为三集群中使用 UpdateServer 上的最大日志时间戳为条件进行选主的过程,其中 $T_{max\text{-}log\text{-}1} < T_{max\text{-}log\text{-}2} < T_{max\text{-}log\text{-}3}$. 集群 2 的 RootServer 的随机等待时间首先到达,它会发起选举,由于 $T_{max\text{-}log\text{-}2} < T_{max\text{-}log\text{-}3}$,集群 3 的 RootServer 会拒绝它的拉票请求,从而集群 2 的 RootServer 从候选者转变为跟随者. 再经过一个随机的等待时间后,集群 3 的 RootServer 再成为候选者发起投票,由于其中 $T_{max\text{-}log\text{-}1} < T_{max\text{-}log\text{-}2} < T_{max\text{-}log\text{-}3}$,从而集群 3 的 RootServer 当选为领导者,它会将自己的信息通知给集群 1 和集群 2 的 RootServer,并在这之后与这两个 RootServer 一直保持租约.

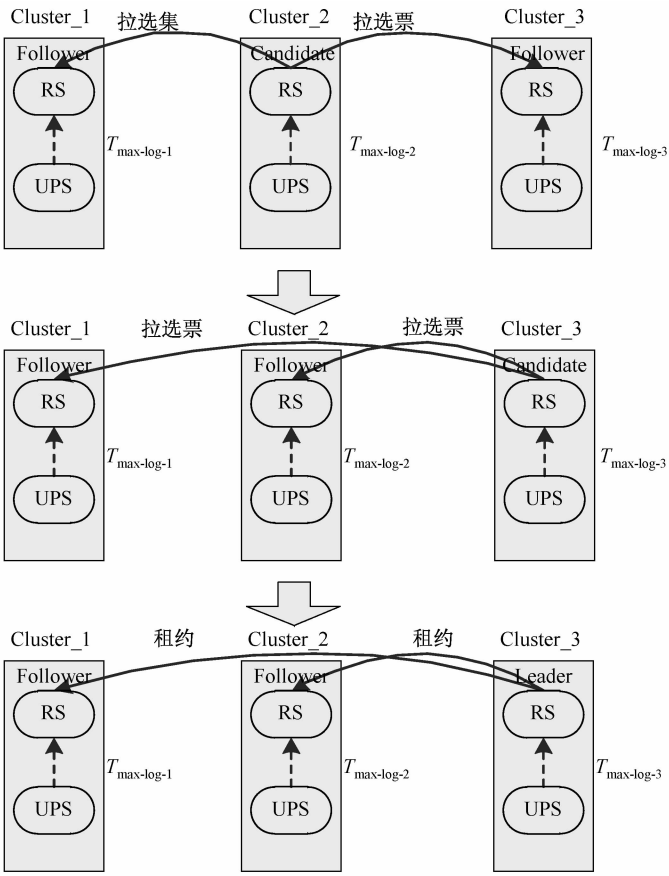

图 4　Raft 一致性协议选主过程

Fig. 4　The process of Raft election

3.2　集群切换功能

RootServer 中选举模块完成选主后，会更改当前 RootServer 的角色，当角色发生变化时系统就会进行相应的集群角色切换，此小节内容介绍在选举模块角色发生变化时主备集群切换的过程.

多集群部署模式下每个集群也会有自己的角色，可以是主集群角色(OBI_MASTER)、备集群角色(OBI_SLAVE)或者初始集群角色(OBI_INIT). 其中主集群是唯一的，它能提供强一致性的查询服务；备集群可以有多个，它们只能提供弱一致性的查询服务；初始化集群不能提供任何查询服务.

选举模块中的角色变化会引起集群角色的变化，进而调起集群切换的流程. 当所有的 RootServer 都是处于刚启动的状态时，在选举模块中每个 RootServer 标记的角色都是跟随者，它们的集群角色都是 OBI_INIT，必须通过人工指定选举模块角色为领导者，这时每个集群都会发生集群角色的改变，集群角色从 OBI_INIT 改变为 OBI_MASTER 或者 OBI_SLAVE. RootServer 的主线程每隔 10 ms 检查一次当前的选举模块角色是否发生变化，一旦发生改变，相当于集群角色改变，进而引起集群的切换.

为了衔接选举模块角色变化与集群切换流程，我们使用三个状态来标记 RootServer 所

处的状态:初始化状态(INIT),选举进行状态(DURING_ELECTION),有主状态(AFTER_ELECTION).

以上三个状态之间的迁移关系如图 5 所示:

(1) 初始化状态是指 RootServer 启动直到选举线程的选举功能被开启这段时间所处的状态;

(2) 选举进行状态是指在选举线程的选举功能启用的情况下无领导者的状态;

(3) 有主状态是指选举线程选出领导者,并且它与其他跟随者保持着租约,领导者可能是自己也可能是其他 RootServer.

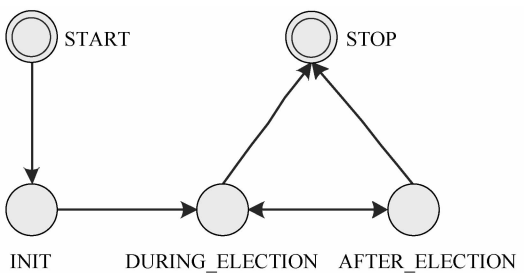

图 5 集群切换的状态迁移图

Fig. 5 The state transition diagram of clusters-switching

RootServer 启动后的状态是初始化状态,它会等待领导者的广播或者被人工手动指定为领导者,然后进入有主状态.

系统中有主之后,每个 RootServer 都会从选举进行状态转换为有主状态.在接下来的过程中,如果选举模块角色没有发生变化,则集群一直保持原来的状态,不会发生切换;如果发生了变化,则会发生集群角色的变化,并引发主备集群切换,主备集群切换过程可以分为两种:①主集群→备集群;②备集群→主集群.

主集群切换为备集群的时候,主 RootServer 首先将自己的集群角色变为备集群角色,再将当前集群中 UpdateServer 的集群角色变为备集群角色.

备集群切换为主集群的时候,新的主 RootServer 首先会将原来的主 RootServer 设定为备集群角色,然后将自己成为主 RootServer 的信息通知给其他所有在线的 RootServer,然后再分别修改它们映射到内部表中集群信息、配置信息,这些信息是集群内部的所有 server 在运行过程都需要用到的.

集群切换成功后 OceanBase java 客户端会感知到集群发生切换,但是用户却不会发觉,因为在 OceanBase java 客户端内部实现了负载均衡算法,它会通过集群之间的流量控制将原主集群中的所有流量全部切换至新主集群上,这样用户就不会发觉主集群发生了切换,从而实现了持续可用性.

3.3 日志强同步功能

主集群是唯一能够接受增量数据的集群,当主集群出现故障时,整个系统将处于不可用的状态,因此需要从备集群中选出一个作为新的主集群,而新选出的主集群要保证与旧集群的数据一致性,才能继续对外提供服务.

OceanBase0.4.2 采用了最大可用模式进行日志同步,即在网络正常时保证主备集群数

据的一致性. 但该日志同步方式无法保证备集群中的数据一定与主集群一致，当系统出现主备切换时，可能导致部分数据的丢失. 为了避免该问题，高可用方案中的日志同步采用了 QUORUM 策略，即主集群将日志发送到包括自己在内的所有集群，仅当收到半数以上响应后才能提交该日志对应的事务，如图 6 所示.

采用 QUORUM 策略日志强同步的具体步骤如下：

(1) 主集群中的 UpdateServer(主 UpdateServer)生成一条日志，将这条日志放入预提交队列中，用异步的方式发送到所有备集群中的 UpdateServer(备 UpdateServer)上；

(2) 每个备 UpdateServer 接收到日志之后将该日志放入回放线程的工作队列中，同时会回复一个当前集群上已经成功刷入磁盘的最大日志号返回给主 UpdateServer；

(3) 主 UpdateServer 接收到这个最大日志号后，判断该日志号之前的日志是否满足 QUORUM 策略，如果满足则从预提交队列中取出该日志号之前对应的事务，并将该事务成功提交，否则继续等待后续的同步回复消息.

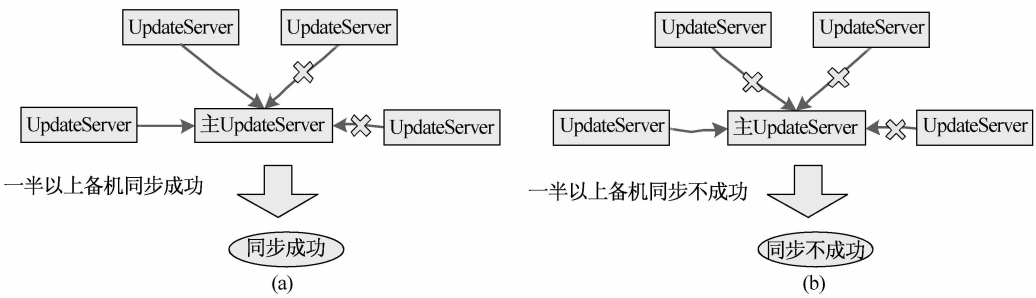

图 6　日志同步策略

Fig. 6　The policy of log synchronization

UpdateServer 在发生故障重启后，日志回放的策略设计如下：

主 UpdateServer 根据备 UpdateServer 返回的最大日志号进行比较，得到可以提交的日志号，称之为已提交点，该点之前的所有日志能够确保在多数派的集群中都是一致的，主 UpdateServer 会将已提交点存储到本地磁盘上. 若当前的主 UpdateServer 宕机重启后，根据重启后所在集群的角色来决定回放日志的策略. 如果集群角色为主集群，则将本地所有日志全部回放；如果是备集群，回放本地日志时只需回放本地日志到已提交点，该点之后的日志从新主 UpdateServer 上拉取，保证了数据的一致性.

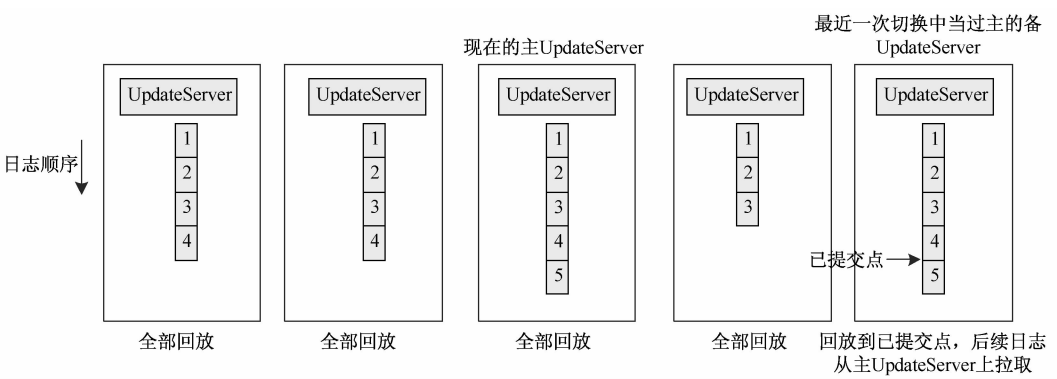

图 7　UpdateServer 重启后回放日志的策略

Fig. 7　The policy of replaying logs after UpdateServer restart

如图 7 所示为当所有的 UpdateServer 都宕机重启后,新的主 UpdateServer 被选举出来,各 UpdateServer 的重放日志的策略. 新的主 UpdateServer 和所有的备 UpdateServer 都是重放所有本地日志. 原来的主 UpdateServer 重启后变为备,它会重放本地日志到它记录的已提交点的位置,后续的日志从新的主 UpdateServer 上拉取.

4 测试实验设计

测试使用的 OceanBase 是在 OceanBase 0.4.2 上经过我们优化的版本,包括 Raft 选举模块功能的实现,集群主备切换实现以及日志强同步功能的实现. 测试环境有两套环境,一套是三集群的 OceanBase 系统,另一套是五集群的 OceanBase 系统. 每个集群中有四台服务器,一台运行 RootServer/UpdateServer,三台运行 ChunkServer/MergeServer,测试集群系统的机器配置参数如表 1 所示.

表 1 硬件配置信息

Tab.1 The information of hardware

Linux 版本	CentOS release 6.5 (Final)
CPU	Intel(R) Xeon(R) CPUE5606,2.13GHz,双核 8 线程
内存	94GB
硬盘	RAID5,2.1TB
网卡	千兆网卡

4.1 实验一

为了验证选举功能模块和主备切换功能模块在 OceanBase 上实现的正确性和效率,可以使用人工模拟 RootServer 的故障来引发选举和集群切换的执行,验证是否能够选出新的主 RootServer 出来,并统计从主 RootServer 宕机到新的主 RootServer 被选出并完成主备集群切换流程所用时间.

实验的步骤如下:

(1)手动杀掉主 RootServer 的,模拟主 RootServer 宕机;

(2)记录新的主 RootServer 选举出来并切换集群成功所用时间;

(3)重启原主 RootServer.

(4)每隔 5 分钟执行步骤 1 至 3,重复 100 次.

在三集群和五集群中分别做此实验,得到的切换耗时如图 8 所示.

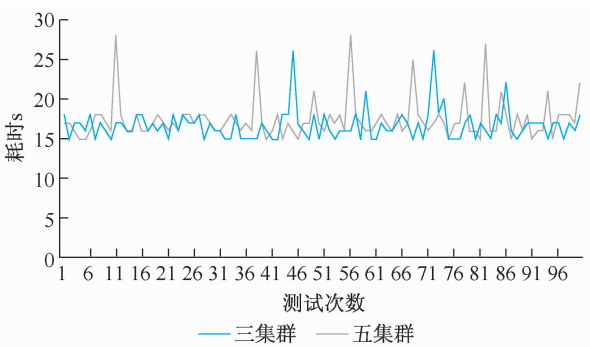

图 8 主 RootSever 故障恢复时间

Fig.8 The recovery time of master RootServer

实验中可以看出在三集群和五集群中选举并切换的成功率均为 100%；三集群中选举并切换的时长的范围在 15 s～26 s，平均时长为 16.7 s，五集群中选举并切换的时长的范围在 15 s～28 s，平均时长为 17.4 s，可见五集群的时长范围更大一些，并且平均耗时更长一些. 这是因为在选举的过程中跟随者会随机等待 1 s～5 s 成为候选者，如果两个跟随者同时发起选举，则会发生冲突从而重新开始选举，那么选举时间就会更长一些. 当 RootServer 的个数越多时，发生冲突的概率就越大，从而选出主的平均时间就会更长一些.

4.2　实验二

此实验是为了验证日志同步的正确性，在主 UpdateServer 出现故障宕机时比较主机上的最大日志号和所有备机上的最大日志号以及已提交点，在正确的情况下半数以上备 UpdateServer 的最大日志号必然大于等于已提交点.

实验步骤如下：

(1)使用多线程测试程序导入数据；

(2)数据导入的同时，手动杀死主 UpdateServer 进程，记录主 UpdateServer 上的最大日志号和所有备 UpdateServer 上的最大日志号，以及主 UpdateServer 上的已提交点，计算已提交点与各 UpdateServer 上的最大日志号的差值.

(3)重启原主 UpdateServer；

(4)每隔 5 分钟执行步骤 1 至 3，重复 20 次.

在三集群部署模式下测试结果如图 9 所示.

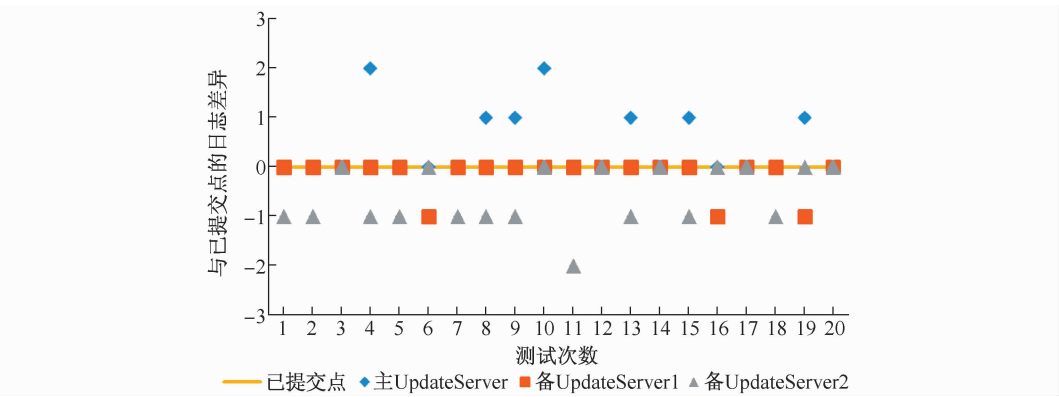

图 9　三集群中测试结果

Fig. 9　The test result in 3-clusters

在五集群部署模式下测试结果如图 10 所示.

由图 9 和图 10 可以看出，每次实验中，主 UpdateServer 上的最大日志号都是大于或等于已提交点，所有备 UpdateServer 上的最大日志号必然有半数以上与已提交点是相同的.

5　总结及未来展望

高可用性是分布式数据库的一个重要性能指标，本文介绍了传统数据库的高可用方案和分布式环境下的几个高可用一致性算法，在分析了 OceanBase 系统的可用性方面的缺陷后，对高可用性方案和分布式一致性算法进行了一些定制和优化，然后介绍了在 OceanBase 数据库上 Raft 一致性协议的实现、集群切换的实现以及日志的同步策略.

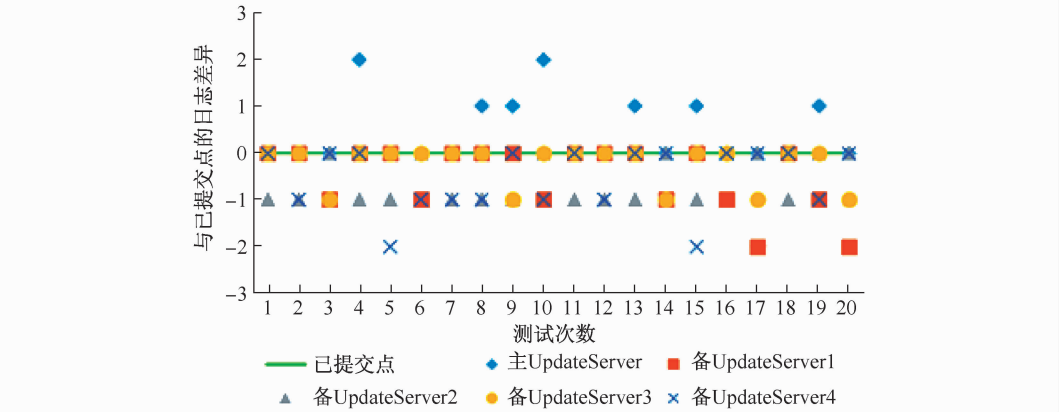

图 10 五集群中测试结果

Fig. 10 The test result in 5-clusters

当前的高可用方案是在多集群模式下实现的,是解决现有问题的最简洁的解决方案,但是在后续的设计中,我们考虑在单集群中也做一些提高可用性的实现,从整体上使系统的可用性更高.

[参 考 文 献]

[1] 阳振坤. OceanBase 关系数据库架构[J]. 华东师范大学学报(自然科学版),2014(5):141-148.
[2] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data[C]//Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation. 2006:205-218.
[3] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally-distributed database[C]//Proceedings of the 10th Conference on USENIX Symposium on Operating Systems Design and Implementation. 2012:251-264.
[4] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: Amazon's highly available key-value store[C]//SOSP'07:205-220.
[5] 吴勇毅. 工信部力挺软件国产化 政策机遇促行业大发展[EB/OL]. [2014-06-05]. http://it.people.com.cn/n/2014/0605/c1009-25108211.html.
[6] OceanBase 开源[EB/OL]. [2014-06-01]. http://code.taobao.org/p/OceanBase/wiki/index/.
[7] 杨传辉. 大规模分布式存储系统:原理解析与架构实战[M]. 北京:机械工业出版社,2013:154-155.
[8] Raft consensus algorithm website[EB/OL]. [2014-02-05]. https://raftconsensus.github.io.
[9] SKEEN D. A quorum-based commit protocol[C]//Proceedings of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks. 1982:69-80.
[10] Oracle maximum availability architecture[EB/OL]. [2014-06-01]. http://www.oracle.com/technetwork/database/features/availability/maa-096107.html.
[11] Oracle Real Application Clusters[EB/OL]. [2014-05-01]. http://www.oracle.com/technetwork/cn/database/options/clustering/overview/index.html.
[12] 黄剑. 基于 Oracle Data Guard 的容灾策略设计与实现[J]. 科技广场,2006(11):71-73.
[13] Oracle data guard[EB/OL]. [2014-05-06]. http://www.oracle.com/technetwork/cn/database/dataguardoverview-091578-zhs.html.
[14] 周欢. OceanBase 一致性与可用性分析[J]. 华东师范大学学报(自然科学版),2014(5):103-116.
[15] 杨传辉. OceanBase 高可用方案[J]. 华东师范大学学报(自然科学版),2014(5):173-179.
[16] LAMPORT L. The part-time parliament[J]. ACM Transactions on Computer Systems, 1998, 16(2):133-169.
[17] CHANDRA T D, GRIESEMER R, REDSTONE J. Paxos made live: An engineering perspective[C]//Proceedings of the 26th Annual ACM Symposium on PODC. ACM, 2007:398-407.
[18] LAMPORT L, MASSA M. Cheap Paxos[C]//Proceedings of the 2004 International Conference on Dependable Systems and Networks. IEEE, 2004:307-314.
[19] LAMPORT L. Fast Paxos[J]. Distributed Computing, 2006, 19(2): 79-103.

(责任编辑 林 磊)