文章编号:1000-5641(2014)05-0103-14

OceanBase 一致性与可用性分析

周 欢[1], 樊秋实[1], 胡华梁[1,2]

(1. 华东师范大学 软件学院,上海 200062; 2. 浙江理工大学 经济管理学院,杭州 310018)

摘要: OceanBase 作为一个面向海量数据查询的分布式数据库,支持关系查询和跨行跨表事务,同时保证了一致性和可用性. 本文在详细阐述了一致性与可用性的背景基础上,根据传统数据库和分布式数据库保证一致性和可用性的协议和策略,对 OceanBase 的一致性和可用性架构进行分析. 最后,探讨了基于 OceanBase 架构演变而来的 3 种架构的实现方案.

关键词: 分布式数据库; 一致性; 可用性; OceanBase

中图分类号: TP392 **文献标识码**: A **DOI**:10.3969/j.issn.1000-5641.2014.05.009

Consistency and availability in OceanBase

ZHOU Huan[1], FAN Qiu-shi[1], HU Hua-liang[1,2]

(1. *Software Engineering Institute, East China Normal University, Shanghai* 200062, *China*;
2. *School of Economic Management Zhejiang Sci-Tech University, Hangzhou* 310018, *China*)

Abstract: OceanBase as a distributed database, not only supports for cross-table relational query and interbank transactions but also ensures consistency and availability. Based on the study of traditional database architectures and distributed database architectures, this article analyzed the architecture of consistency and availability in OceanBase, and finally discussed the implementations of the three architectures evolved from OceanBase.

Key words: distributed database; consistency; availability; OceanBase

0 引 言

宽带、无线和移动终端的普及,方便了用户通过浏览器和移动 APP 来访问各类信息系统,也使得信息系统的用户规模和负载情况发生了巨大的变化. 例如,数千万用户可以通过手机 APP 随时访问和操作自己的银行账户;传统商用数据库系统的并发处理能力和系统可扩展性,已经难以满足互联网时代对数据存储和处理能力的需求;热门理财产品,短时间内可能出现上百万的并发访问请求.

近年来,采用分布式架构的具有良好扩展性和灵活性的数据管理技术和产品不断涌现,并且被广泛应用于各种互联网应用中,例如 Big Table[1]、Spanner[2]、OceanBase[3] 等. 虽然

收稿日期:2014-06

基金项目:浙江省自然科学基金(LY12F02044)

第一作者:周欢,女,博士生,研究方向为分布式数据库. E-mail: zhouhuan9026@gmail.com.

通信作者:胡华梁,男,副教授,硕士生导师,研究方向为数据库. E-mail: jmxyandy@126.com.

这些系统多数拥有存储和管理海量数据的能力,但在数据一致性和事务处理能力上,同传统的关系数据库产品还有差距,很难被直接应用于银行业务系统等传统的大型信息系统中.与论坛、微博等互联网应用不同,以银行业务为代表的金融领域信息系统对于数据库的可用性和一致性有非常高的要求.Brewer 的 CAP 理论[4]指出,分布式系统不可能同时满足一致性(Consistency)、可用性(Availability)和分区容错性(Partition Tolerance),最多只能同时较好地满足其中两个.在选择和应用一个分布式数据库产品时,对于该产品在一致性、可用性和分区容错性之间的取舍进行衡量,确认该产品能在多大程度上保证数据库的强一致性和高可用性,是决定这个产品能否被应用于金融领域信息系统的关键.

OceanBase 是阿里巴巴集团研发的开源分布式关系数据库产品,具有可扩展性、高可用性和低成本等特征.目前,该产品已经被成功应用于阿里巴巴集团的淘宝和天猫等平台上的业务系统,还将被应用于支付宝等金融类业务系统.分析和研究 OceanBase 系统的一致性和可用性,将有助于该系统的进一步研发和应用,有利于推动各类信息系统建设中的数据管理技术的发展.本文作者参与了使用 OceanBase 作为数据库平台的应用系统开发,对于 OceanBase 有一定的了解.本文试图通过分析和研究 OceanBase 系统的源代码和文档,对该系统的一致性和可用性做出客观评价,以期能够对其他研究和开发人员有所帮助.

为了便于理解和阅读,本文首先在第 1 节中介绍了数据库系统一致性和可用性的概念,以及用于保障一致性和可用性的协议和技术;第 2 节对 OceanBase 两个最新版本(0.4.2和0.5)的架构和实现技术进行了介绍,并对其各自实现一致性和可用性的策略进行了分析;针对 OceanBase 的单点写性能的问题,第 3 节对 OceanBase 未来可能的架构演变方案进行了探讨,同时对架构演变可能对系统一致性和可用性的影响进行了分析;第 4 节对全文进行了总结,并根据 OceanBase 系统的现状提出了对应用开发者的建议.

1 一致性与可用性背景

可用性是指系统在面对各种异常时可以提供正常服务的能力.系统的可用性可以用系统停止服务时间与正常服务时间的比例来衡量;例如,某个系统的可用性为 5 个 9(99.999%),相当于系统一年停止服务的时间不能超过 $365 * 24 * 60/100\ 000 = 5.25$ min.为了保证系统的高可靠和高可用,数据在系统中一般存储多个副本,而多个副本又可能带来数据不一致的问题.一致性是指对同一客户端而言,读操作总能读取到最新完成的写操作结果.一致性和可用性是相互矛盾的.为了保证数据一致性,各个副本之间需要时刻保持强同步;但是当某一副本出现故障时,可能阻塞系统的正常写服务,从而影响系统的可用性;如果各副本之间不保持强同步,虽然系统的可用性相对较好,但是一致性却得不到保障,当某一副本出现故障时,数据还可能丢失.因此,数据库设计时需要权衡系统的一致性和可用性.

传统数据库依赖高端服务器采用共享存储或者主备库同步的方式实现系统的一致性和可用性.主备库同步包括 3 种模式[5]:

1) 最大保护模式 事务必须先同步到备库,主库才能应答客户端成功;

2) 最大性能模式 事务异步同步到备库,只要主库执行成功就可以应答客户端;

3) 最大可用模式 事务尽力同步到备库,当主备库之间网络正常时,采用最大保护模式;当主备库之间网络出现故障时,退化为最大性能模式.

3 种模式支持了不同程度的一致性和可用性,最大保护模式能够保证主备库数据的一

致性，但是当备库或者主备库之间出现网络故障时，系统将无法提供服务；最大性能模式能够保证系统的高可用性，但是主备库的数据不能保证完全一致，当主库出现故障时，则可能导致数据不一致；最大可用性模式介于最大保护模式和最大性能模式之间，当主备库之间网络正常时保证数据一致性，当网络出现故障时保证系统的可用性，但是当主备库之间的网络和主库同时出现故障时，主备库的数据可能不一致. 在应用生产中，为了兼顾系统的一致性和可用性一般选择最大可用模式.

随着数据规模越来越大，Oracle、微软、SAP 等传统数据库系统软件公司自主研发或收购了一批分布式数据库管理系统，如 Oracle Berkeley DB[6]、SAP HANA[7]、VoltDB[8] 等，来解决传统数据库在处理互联网应用时的低并发处理能力和不易扩展等问题. 这些高性能数据库管理系统大多采用存储多个数据副本的方式来保证可用性和可靠性，使用不同的一致性协议和并发控制技术来保证一致性. 在分布式环境下，数据通过分区和复制的方式分布在不同的 PC 服务器上，分布式数据库的一致性包括以下两个方面.

1）副本一致性　同一数据的多个副本之间的一致性；

2）事务一致性　在分布式并发事务场景下，不同数据之间事务的一致性.

要保证数据一致性，需要所有副本有一个本地操作执行更新的确切时间；例如，副本可以使用 Lamport 时间戳[9]来决定执行操作的全局顺序，或者用一个协调器来分配这样一个顺序. 然而，CAP 定理证明了一致性、可用性、分区容错性三者无法兼得. 在大规模分布的系统中，由于网络分区无法避免，因此为了保证系统的可用性和分区容错性，提出了多种一致性模型[10]，通常分为三种.

1）强一致性　数据在任一副本上被更新成功后，后续任何对该数据的读取操作都能得到最新的值.

2）弱一致性　在一定的时间窗口内，对某一数据的读取操作得不到最新的值.

3）最终一致性　是弱一致性的特例，系统确保最终所有访问都能得到更新的数据.

根据应用对数据一致性的不同容忍程度，系统可以选择不同的一致性模型. 在分布式系统领域，实现副本一致性的协议有很多. 例如，Paxos[11] 协议和基于主副本的强同步复制协议[10]实现了副本的强一致性，HDFS 采用的 NRW 协议[12]实现了副本最终一致性. 保证事务一致性的协议和技术也有很多. 例如，两阶段提交协议(2PC)[13]保证了分布式事务的原子性；向量时钟(Vector Clock)技术[10]、分布式加锁机制、Chubby 互斥锁机制[14]、全球时钟同步机制 TrueTime 技术[2]和多版本并发控制(MVCC)解决了分布式环境下的并发冲突保证了数据正确性. 两阶段提交协议是阻塞协议，执行过程中需要锁住其他更新并且有大量的网络通信开销，为了提高保证系统的可用性和高性能，大多分布式系统选择尽量避免分布式事务. 为了更好地兼顾可用性和一致性，在实际应用中，常常通过将应用在逻辑上进行分区，在分区内实现强一致性；在分区间通过应用层的处理解决网络分区故障带来的一致性问题，从而保证系统的可用性.

在数据库实现中，为了保证并发事务操作之间相互不受影响，采用两阶段封锁实现事务的串行化；但事务串行化执行将导致很多资源得不到充分利用，大大降低了系统并发能力. 为了提高读写性能，数据库支持在数据的不同粒度[15]上进行加锁并且放松释放锁的时间；例如在行粒度上进行加读锁和写锁，读锁可以在读操作结束之后立即释放，不用等到事务结束之后才释放. 在保证事务写操作串行执行的基础上适当放宽读操作的要求，从而产生了 4

个事务隔离级别.

1) 序列化(Serializable):提供严格的事务隔离.要求事务序列化执行,即事务只能一个接一个执行,不能并发执行.实现序列化需要锁定整张表,并且读锁和写锁在事务提交之后才能释放.

2) 可重复读(Repeatable Read):要求在一个事务中可以多次读取某些数据,且每次读取的结果相同,并且只能读到已提交的数据.但是有时会产生幻读,即重复读取的结果集中存在一条新插入的记录.实现可重复读是在一定范围内加读、写锁,并且读锁和写锁在事务提交之后才能释放.

3) 读已提交数据(Read Committed):要求每次读取必须是已提交的数据.但是可以出现不可重复读和幻读.读已提交数据是在单行记录上加读、写锁,并且读锁在读取单条记录结束之后立即释放,写锁要在事务提交之后才能释放.

4) 读未提交数据(Read Uncommited):允许读取没有提交的数据,但不允许更新丢失.读未提交数据是在单行记录上加写锁.写锁需要等到事务提交之后才能释放,这里的写锁排斥其他写事务,但允许读取该行数据.

采用多版本并发控制实现的分布式数据库还提供另一种隔离级别——快照(Snapshot)读,即读取某一时间戳的数据.数据研究和设计者发现,实现高可用的分布式数据库无法提供序列化和可重复读的隔离级别,因此大多主流数据管理系统都将读已提交(Read Committed)作为缺省隔离级别,如 MemSQL 1b[16]、Oracle 11g[16]、SAP HANA 等.

2 OceanBase 的一致性与可用性

OceanBase 是阿里巴巴集团研发的,具有高可用性、高性能、强一致性和支持跨行跨表事务等特点的分布式数据库.OceanBase 将整体架构划分为 4 个模块:主控服务器 RootServer、更新服务器 UpdateServer、基线数据服务器 ChunkServer 以及合并服务器 MergeServer.其主要功能如下.

1) RootServer 提供服务器管理和集群管理的功能,存储集群的管理信息,一般与 UpdateServer 部署在同一台服务器中;

2) UpdateServer 存储系统最近一段时间的增量更新数据,是唯一的写入模块,一般与 RootServer 部署在同一台服务器中;

3) ChunkServer 存储系统的基准数据,一般与 MergeServer 部署在同一台服务器中;

4) MergeServer 接收并解析用户的 SQL 请求,经过词法分析、语法分析、查询优化等一系列操作后,将物理执行计划发到相应的 ChunkServer 或者 UpdateServer,一般与 ChunkServer 部署在同一台服务器中.

OceanBase 采用读写分离架构将共享数据分为基准数据和增量数据.基准数据通过分区和复制的方式分布在多台 ChunkServer 中,并且只提供读服务,增量数据存储的是所有基准数据的更新操作结果并集中式存放在一台 UpdateServer 上.增量数据通过定期合并操作融合到基准数据中,避免了单台 UpdateServer 的存储容量成为瓶颈,实现了良好的扩展性.这种架构将写事务集中在一台服务器上,巧妙地避免了复杂的分布式事务,高效地实现跨行跨表事务,同时大大降低了实现分布式数据库一致性和高可用性的难度.

2.1 OceanBase 0.4.2 的一致性与可用性

目前，阿里巴巴开源的是 OceanBase 0.4.2，该版本采用传统的主备双集群架构来保证系统的一致性和可用性. 每个集群里有多台 ChunkServer，多台 MergeServer，两台 RootServer，两台 UpdateServer. 其双集群架构如图 1 所示.

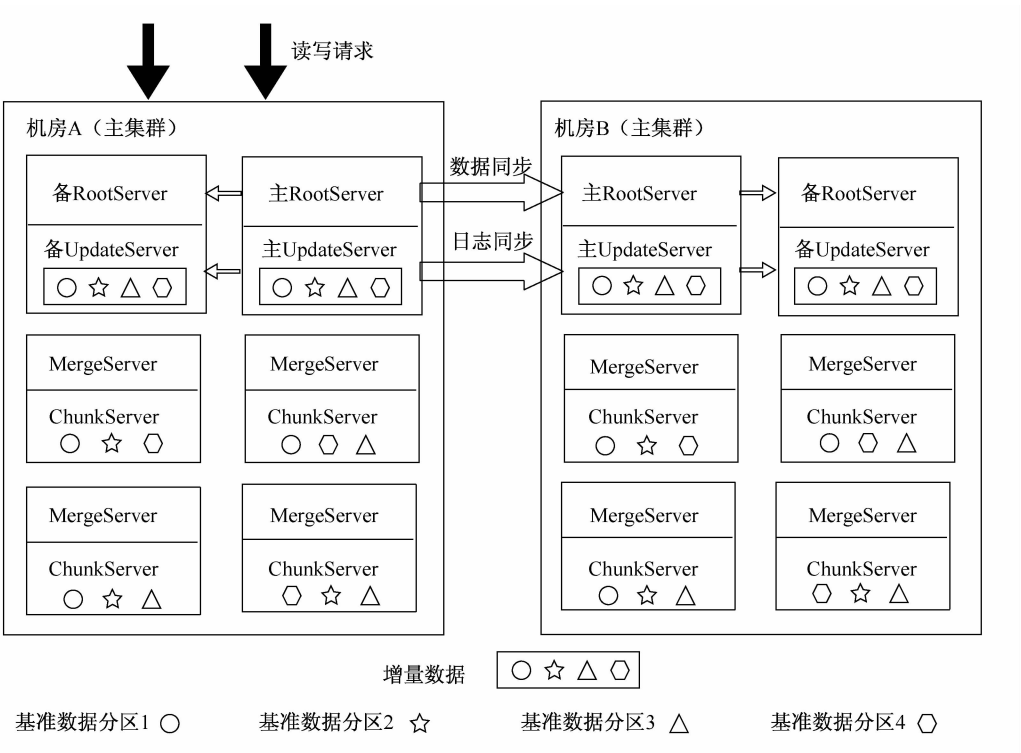

图 1　OceanBase 0.4.2 版本的架构

Fig. 1　The architecture of OceanBase 0.4.2

双集群通过同步 RootServer 的管理信息和 UpdateServer 的更新日志来保证系统的一致性并采用最大可用模式来保证系统的可用性. 每个集群内 RootServer 和 UpdateServer 均采用主备高可用模式. 主备 RootServer 之间通过数据强同步来保证一致性并且使用 Linux HA 来实现可用性. 通常情况下，集群中的 ChunkServer、UpdateServer 和客户端通过 VIP(Virtual IP)访问主 RootServer，当主 RootServer 出现故障时，Linux HA 检测到该故障并将指向主 RootServer 的 VIP 漂移到备 RootServer，备 RootServer 的后台线程检测到 VIP 漂移到自己机器上后自动切换为主机提供服务. 为了确保增量数据一致性，RootServer 通过租约(Lease)机制为每个集群选取一台主 UpdateServer 对外提供写服务，租约有效期一般为 3～5 s. 正常情况下，RootServer 会定期给主 UpdateServer 发送命令延长租约有效期. 如果主 UpdateServer 出现异常，RootServer 等待主 UpdateServer 的租约过期后才能选择其他的 UpdateServer 为主 UpdateServer 继续提供写服务. UpdateServer 采用主备同步的最大可用性模式(Maximum Availability)来保证 OceanBase 的高可用性，主备之间通过日志同步来保证数据副本的一致性. 其工作流程如下：

1) 主 UpdateServer 将更新操作日志(redo 日志)发送到备 UpdateServer；

2) 将操作日志写入主 UpdateServer 硬盘；

3) 将操作日志应用到主 UpdateServer 的内存表(MemTable)中;

4) 等备 UpdateServer 返回写入成功后主 UpdateServer 返回客户端写入成功.

OceanBase 的高可用机制保证主 UpdateServer、备 UpdateServer 及主备之间网络三者之中的任何一个出现故障都不会对客户端产生影响. 然而,如果三者之中的两个同时出现故障,则系统的可用性将受到影响. 主备 UpdateServe 之间的副本不是时刻保持强同步,当主备之间网络出现故障时,主备 UpdateServer 之间采用异步方式来同步日志,那么当主 UpdateServer 出现故障时,备 UpdateServer 的数据往往落后于主 UpdateServer,如果将服务切到备 UpdateServer,可能会丢失一部分提交的事务,导致数据不一致. 因此 OceanBase 是牺牲增量数据副本的一致性来保证系统的高可用性.

UpdateServer 作为一个单点支持跨行跨表写事务,使用加锁机制和多版本并发控制(MVCC)来实现事务的并发控制. UpdateServer 内部使用 MemTable 存储结构作为对外提供统一的读写接口,MemTable 内存结构包括两个部分:高性能内存 B 树索引结构和行操作链表,B 树的叶子节点存储表的主键值,每行的更新操作按时间顺序构成一个行操作链表挂在 B 树的叶子节点上. 处理写事务时,首先会在 B 树的叶子节点上申请写锁避免并发时的写写冲突,然后根据事务提交时的系统时间戳生成一个事务版本号,最后将事务版本号和修改操作一起存入行操作链表. 处理读事务时不需要申请读锁,直接根据版本号读取某个版本的快照数据,而且每次都能读到已提交的最新数据,由此可以看出 OceanBase 支持事务的强一致性并提供 Read Committed 和快照(Snapshot)读两种事务隔离级别.

OceanBase 0.4.2 采用主备同步最大可用性模式的高可用架构,牺牲副本间的强一致性来满足系统的可用性. OceanBase 0.4.2 的可用性能达到 4 个 9(99.99%),单集群内主机发生故障时可以通过 Linux HA 和租约机制自动切换到备机提供服务,但是双集群间主集群发生故障时需要手动切换到备集群提供服务. 在事务处理方面,OceanBase 0.4.2 采用分布式读、集中式写的架构避免了大量的读写冲突和分布式事务,降低了处理分布式系统事务的难度,并采用不加读锁,在行粒度上加写锁和 MVCC 策略实现事务的并发控制,在保证可用性和高性能的基础上提供了 Read Committed 和快照(Snapshot)读两种事务隔离级别. 虽然 OceanBase 0.4.2 提供了高可用性,事务强一致性和数十万 TPS、数百万 QPS 的访问量,但处理读写请求时,需要先从 ChunkServer 中读取基准数据,再带着基准数据去 UpdateServer 执行读或写操作,处理过程中存在着大量的网络开销降低了读写性能,同时 UpdateServer 单点也限制了 OceanBase 集群整体的读写性能.

2.2 OceanBase 0.5 的一致性与可用性

在生产应用中发现,OceanBase 0.4.2 可能存在因数据不一致而导致系统故障的情况. 为了提高 OceanBase 的一致性和可用性,OceanBase 0.5 版本提出了三机房的集群架构:每个机房里有多台 ChunkServer,多台 MergeServer,一台 RootServer,一台 UpdateServer. 我们将这种三机房的集群称为大集群. 其机房架构如图 2 所示.

在大集群中,RootServer 和 UpdateServer 均采用一主多备的架构,同一时刻只有一台主 RootServer 和一台主 UpdateServer 对外提供服务. 在多台 RootServer 之间,使用基于 Paxos 的分布式选举协议,保证有且只有一台 RootServer 被选为主 RootServer 对外提供服务,从而保证了系统的一致性. 备 RootServer 定期从主 RootServer 同步内容,从而保证数据副本的一致性;当主 RootServer 出现故障时,分布式选举协议将选举出一台可靠的备 Root-

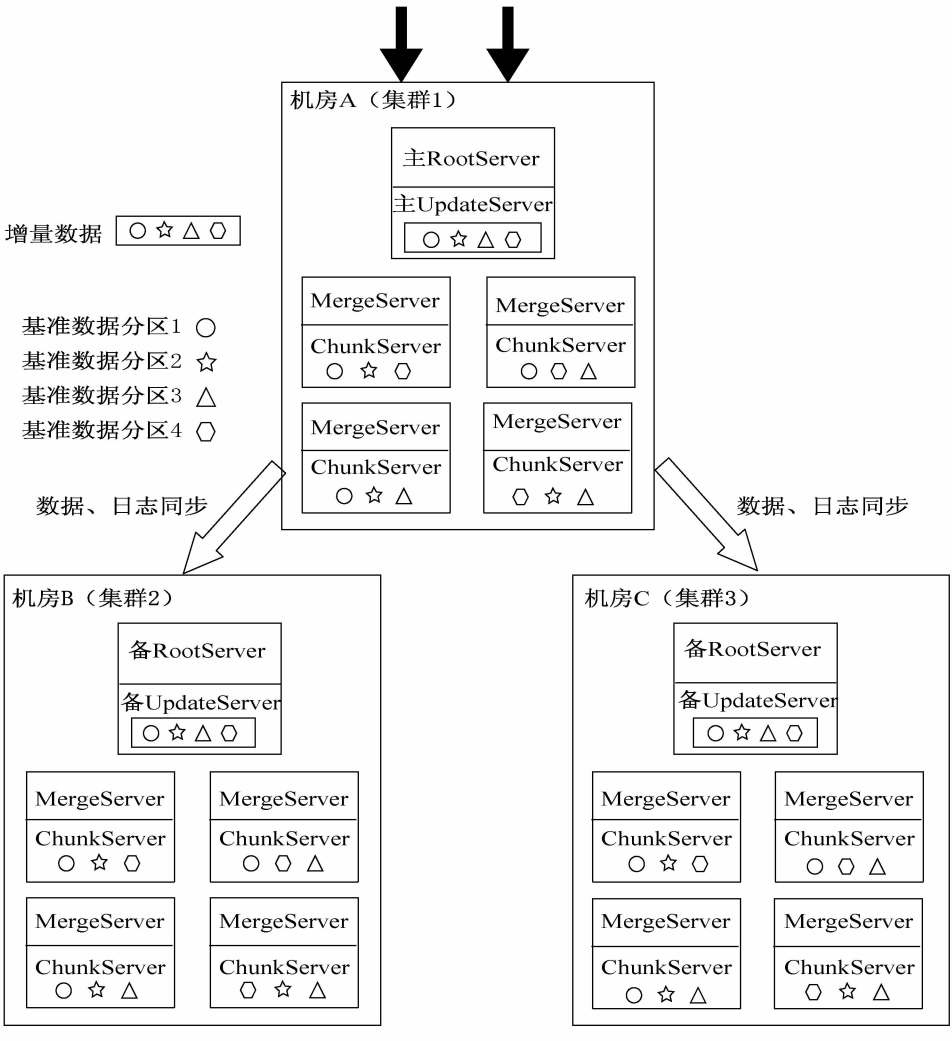

图 2　OceanBase 0.5 版本的架构

Fig. 2　The architecture of OceanBase 0.5

Server 作为主 RootServer 继续提供服务，保证了系统的可用性. 基于 Paxos 的分布式选举协议的基本准则包括以下三个方面：

1）成员不说假话（非拜占庭式）；

2）单个成员说话不自相矛盾：就某一决议投票给了 A，就不能再投票给 B；

3）任何修改需要多数成员同意.

OceanBase 0.5 在实现分布式选举协议时对 Paxos 协议进行了简化，将原生态 Paxos 的“异步模式”改进为“同步模式”，不仅实现上变得简单同时能够保证系统只有几秒到十几秒的时间不能提供服务，但是 OceanBase 选举协议依赖时钟同步，要求多台服务器之间的时钟偏差不能过大，底层的 NTP 服务需要严格保证时钟同步. 多台 UpdateServer 之间由主 RootServer 使用租约（Lease）机制为大集群选举一台主 UpdateServer，当主 UpdateServer 出现故障时，RootServer 将从剩余两台备 UpdateServer 中选择日志时间戳最大的作为主

UpdateServer 继续提供服务,从而保证了 UpdateServer 的可用性. 主备 UpdateServer 之间使用 Paxos 协议同步操作日志从而保证了数据副本一致性,主 UpdateServer 采用异步请求方式向备 UpdateServer 同步日志,但是写本地硬盘采用同步的方式,即前一个日志块没有响应不会写下一个日志块,并且任何一份数据需要在三台 UpdateServer 中的两台(超过半数 UpdateServer)确认写盘成功主 UpdateServer 才认为写请求成功,保证了任何一台机器出现故障时系统都不会丢失数据,可以继续提供服务,从而兼顾了系统的一致性和可用性. 在事务处理方面,OceanBase 0.5 仍使用分布式读集中式写的事务处理架构,采用行粒度上加写锁和 MVCC 来实现并发控制,保证事务强一致性并提供 Read Committed 和快照(Snapshot)读两种事务隔离级别.

OceanBase 0.5 采用主备强同步策略和分布式选举保证了数据副本一致性和高可用性. 虽然在每次更新时保证至少有一个备副本与主副本完全同步,但是所有副本之间仍满足最终一致性. 因此,这种实现方案仍是牺牲了副本的强一致性来保证高可用性. OceanBase 0.5 的可用性可以达到 4 个半 9. 事务处理方面仍采用 OceanBase 0.4.2 的实现方法保证了事务一致性并提供供 Read Committed 和快照(Snapshot)读两种隔离级别. 虽然 OceanBase 0.5 提供了更高的可用性,但集群中仍只有一台 UpdateServer 提供写服务,UpdateServer 单点影响系统整体读写性能的缺陷仍然存在,同时处理读写请求时的网络开销也仍然存在.

3 多 UpdateServer 架构的一致性与可用性

OceanBase 虽然实现了一致性和可用性,但单点 UpdateServer 和处理读写请求的网络开销限制了系统的读写性能. 为了提高 OceanBase 集群的读写性能,需要多个 UpdateServer 同时提供写服务. 改善单点性能的方法有两种:第一是数据镜像,把所有数据同步到多个服务器,服务器间提供无差别的服务;第二是数据分区,把数据按照一定规则进行分区存放在服务器上.

3.1 数据镜像架构

此架构采用三机房的集群:每个机房里有多台 ChunkServer,多台 MergeServer,一台 RootServer,一台 UpdateServer. 数据分为基准数据和增量数据,基准数据采用分区和复制的方式存放在一个机房多台 ChunkServer 上,然后再镜像拷贝到其他机房;增量数据集中式存放在一个机房的 UpdateServer 上,然后再镜像拷贝到其他机房. 两两集群间的基准数据和增量数据作为无差别的数据副本对外提供服务,数据副本通常分为两种架构即 Master-Slaver[17] 和 Master-Master[17],Master-Slaver 架构是指多个数据副本中存在一个主副本和多个备副本并且同一时刻只有主副本对外提供服务,Master-Master 架构是指多个数据副本都可以对外提供服务,各个数据副本之间是平等的关系不存在主备之分. OceanBase 0.5 中增量数据使用的是 Master-Slaver 数据副本架构,而此架构采用 Master-Master 数据副本架构,即同一时刻三台 UpdateServer 都可以接受写请求. 数据镜像架构如图 3 所示.

在这种架构模型下,RootServer 采用一主多备架构,同一时刻只有一台主 RootServer 提供服务,多台 RootServer 之间仍采用 OceanBase 0.5 的分布式选举协议来保证一致性和可用性. UpdateServe 的增量数据采用 Master-Master 架构,三台 UpdateServe 同时接收写请求的情况下,保证增量数据一致性可以采用 NRW 协议[17],这里的 N 表示总的副本数,R 表示每次读请求需要读取的副本数,W 表示每次写请求需要写入的副本数. R、W 需要满足

以下条件：

1）$R+W>N$；　　2）$W>N/2$.

至于 R、W 取值多少，可以根据应用需求来设置. 这种一致性协议虽然能保证每次读取到的都是最新值，但是所有副本满足最终一致性. 在事务处理方面，采用分布式锁机制和 MVCC 来实现并发控制. 一种简单的加锁方法是，采用集中式封锁机制维护一张全局锁表，锁表存储每个副本加锁的信息. 这种方法的明显缺点是，存在单点性能，当存放锁表的机器出现故障时，系统将不能提供服务；另一种加锁方法是，在每个 UpdateServer 都维护一张全局的锁表. 这种方法虽然避免了单点故障，却增加了保证锁表一致性的难度；另一种加锁方法是使用 NSX 封锁机制[10]，其中 N 表示总的副本数，S 表示事务获得数据项 A 的全局共享锁时必须以共享方式封锁的 A 的副本数，X 表示事务获得数据项 A；只要满足 $2X>N$ 且 $S+X>N$，就能满足数据项 A 上只能有一个排他锁：不会既有一个排他锁又有一个共享锁. 还有一种方法是，采用向量时钟（Vector Clock）来解决写写冲突. 但这种方法依赖集群内部节点之间的时钟同步算法，不能完全保证事务一致性.

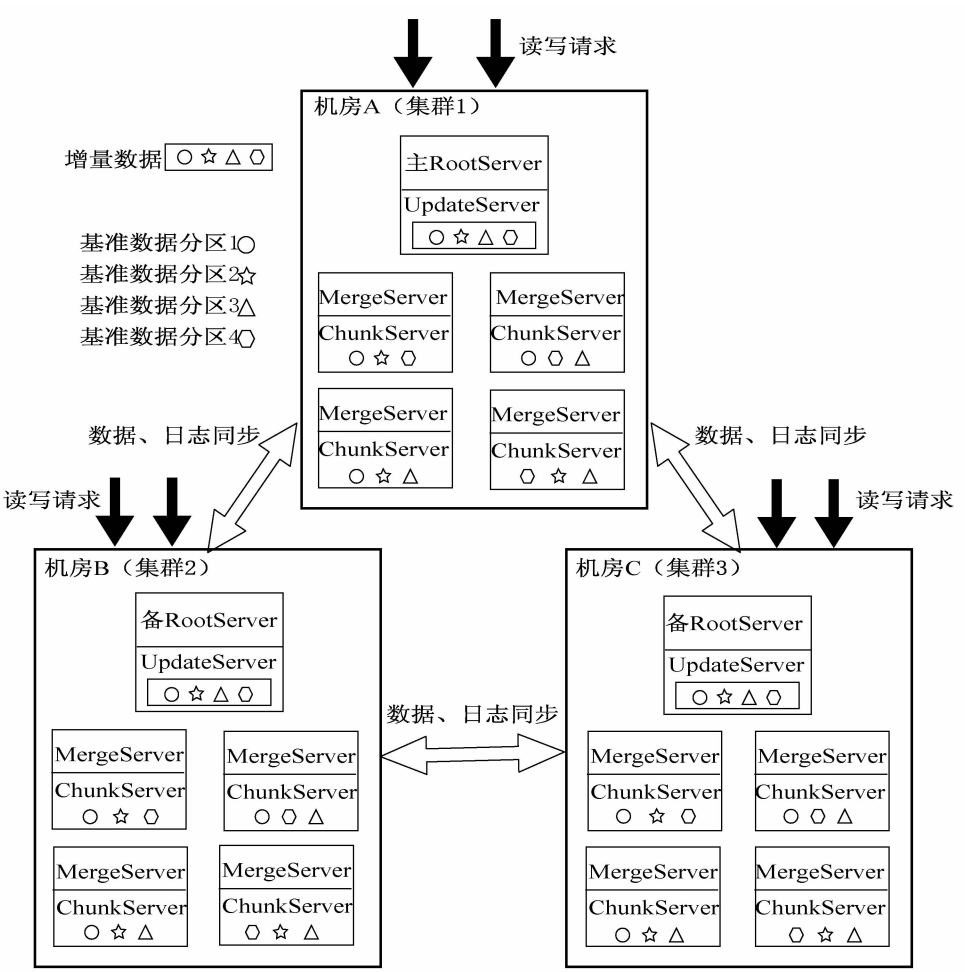

图 3　数据镜像架构模型

Fig. 3　The architecture of data mirroring

这种架构通过 UpdateServer 采用 Master-Master 架构提高了系统整体的读写性能和系统的扩展性,但是保证数据一致性却需要花费很大的代价.实现方案中数据副本只保证最终一致性,如果多个 UpdateServer 之间的更新顺序不一致,客户端可能读不到预期的结果.而且系统的可用性也不是很高,当某台 UpdateServer 或者客户端与 UpdateServer 之间的网络出现故障时,R、W 不能满足预定的值则不能提供服务,从而降低了系统的可用性.总的来说,提高了系统读写性能却要承担数据不一致的风险.在实际生产应用中,不提倡这种架构.

3.2 数据分区架构

此架构采用三机房的集群架构,每个机房里有多台 ChunkServer,多台 MergeServer,一台 RootServer,多台 UpdateServer.数据分为基准数据和增量数据,基准数据采用分区和复制的方式存放在一个机房的多台 ChunkServer 上,然后再镜像拷贝到其他机房;增量数据采用分区和复制的方式存放在三个机房的多台 UpdateServer 上,两两集群间基准数据是一样的,但增量数据可能不一样,其架构如图 4 所示.

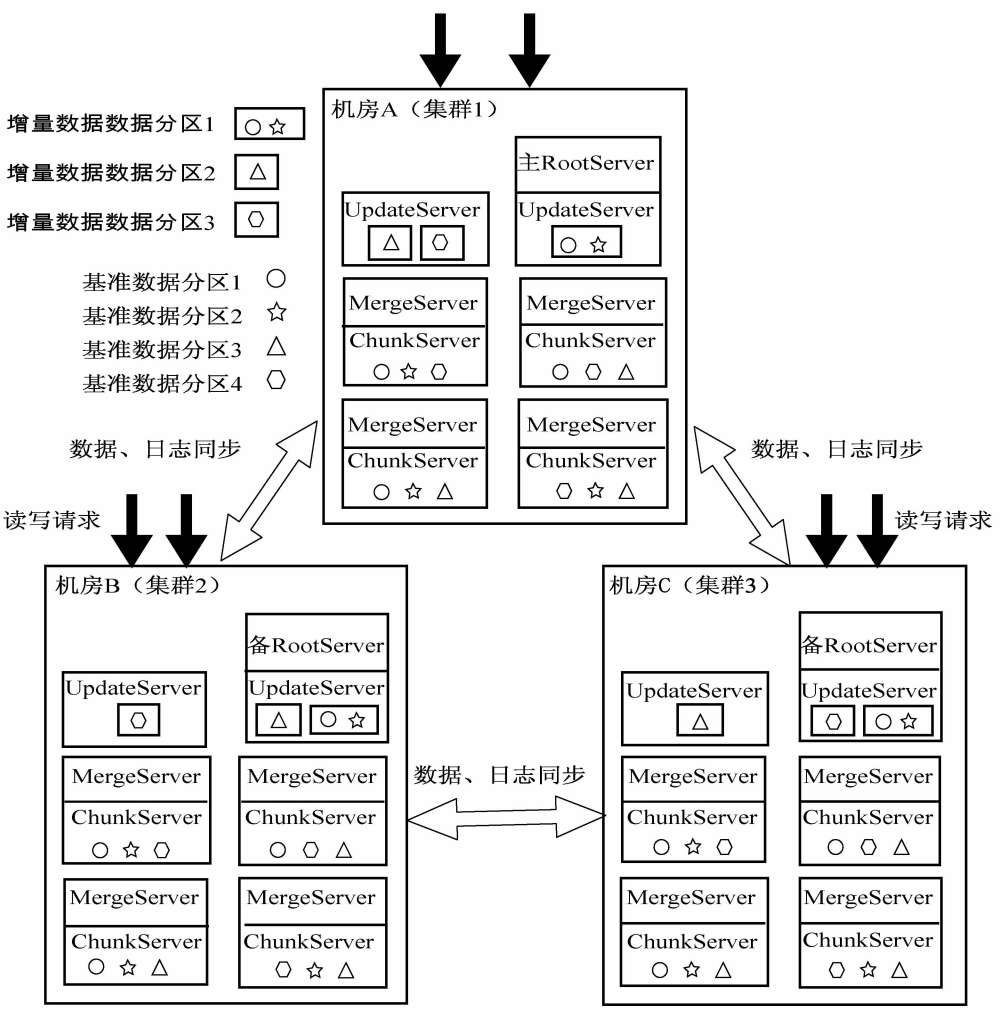

图 4 数据分区架构模型

Fig. 4 The architecture of data partition

在这种架构模型下，RootServer 采用一主多备的架构：同一时刻只有一台主 RootServer 提供服务，多台 RootServer 之间仍采用 OceanBase 0.5 的分布式选举协议来保证系统的可用性. UpdateServer 增量数据采用 Master-Slaver 架构：同一数据分区副本之间由主 RootServer 选出唯一的主分区副本并对外提供服务，主备分区副本之间通过 Paxos 协议同步操作日志来保证分区副本一致性. 在事务处理方面，由于增量数据分布在多个 UpdateServer 上，可能存在跨多个节点的分布式事务，此时需要使用两阶段提交(2PC)协议来保证分布式事务的原子性，使用分布式锁机制和 MVCC 来实现并发控制. 两阶段提交协议是阻塞协议，在执行过程中需要锁住其他更新并且需要大量的网络通信，这样将直接影响系统的读写性能. 提高性能的唯一办法是减少分布式事务，减少分布式事务则需要根据应用场景将大多数事务涉及的数据划分到一个数据分区中并且将大多数事务涉及的数据分区存放在同一个物理机器上. 通过分区规则使得 95％的事务都是集中式事务，只有 5％的事务是分布式事务. 此架构下能够保证事务一致性并提供 Read Committed 和快照(Snapshot)读两种事务隔离级别.

这种架构采用数据分区主副本强同步和分布式选举来保证系统的一致性和高可用性，并且采用数据分区的方式改善了单点 UpdateServer 的读写性能和 UpdateServer 的扩展性. 在事务处理方面，采用较好的数据分区规则避免大量的分布式事务影响系统的读写性能，采用两阶段提交协议、加锁机制和 MVCC 来保证事务原子性、一致性并提供 Read Committed 和快照(Snapshot)读两种隔离级别. 这种架构虽然保证了系统的高可用性、事务一致性和较好的读写性能，但是读写分离架构在处理读写事务时会有大量的网络开销，这将直接影响系统的读写性能.

3.3　节点对等架构

此架构采用三机房的集群架构，每个机房里有多台 ChunkServer，多台 MergeServer，一台 RootServer，多台 UpdateServer，将 ChunkServer、MergeServer 和 UpdateServer 部署在一台物理机上. 增量数据和基准数据采用分开存储，每台物理机上的增量数据存储该台物理机上基准数据的更新修改结果. 增量数据采用分区方式存于一个机房的 UpdateServer 中，基准数据采用分区方式存于一个机房的 ChunkServer 中，集群之间采用镜像拷贝存储基准数据副本和增量数据副本，提供无差别的服务. 其架构如图 5 所示.

在这种架构模型下，RootServer 采用 OceanBase 0.5 中实现 RootServer 一致性和高可用性的架构和策略，同一时刻选出唯一的主 RootServer 对外提供服务. UpdateServer 增量数据采用 Master-Slaver 架构，由主 RootServer 从多个增量数据分区副本中选出一个主副本对外服务，副本之间采用 Paxos 协议同步更新操作日志保证增量数据分区副本的一致性和可用性. 在事务处理方面，同一个增量数据分区内的事务采用加行锁和 MVCC 来保证事务的一致性，不同增量数据分区之间的分布式事务采用两阶段提交协议、分布式加锁和 MVCC 来保证分布式事务的一致性. 为了提高系统性能，需要运用较好的数据分区规则，以尽量避免应用中的分布式事务. 在保证系统高可用、数据一致性和事务一致性的基础上，此架构可以提供 Read Committed 和快照(Snapshot)读两种事务隔离级别.

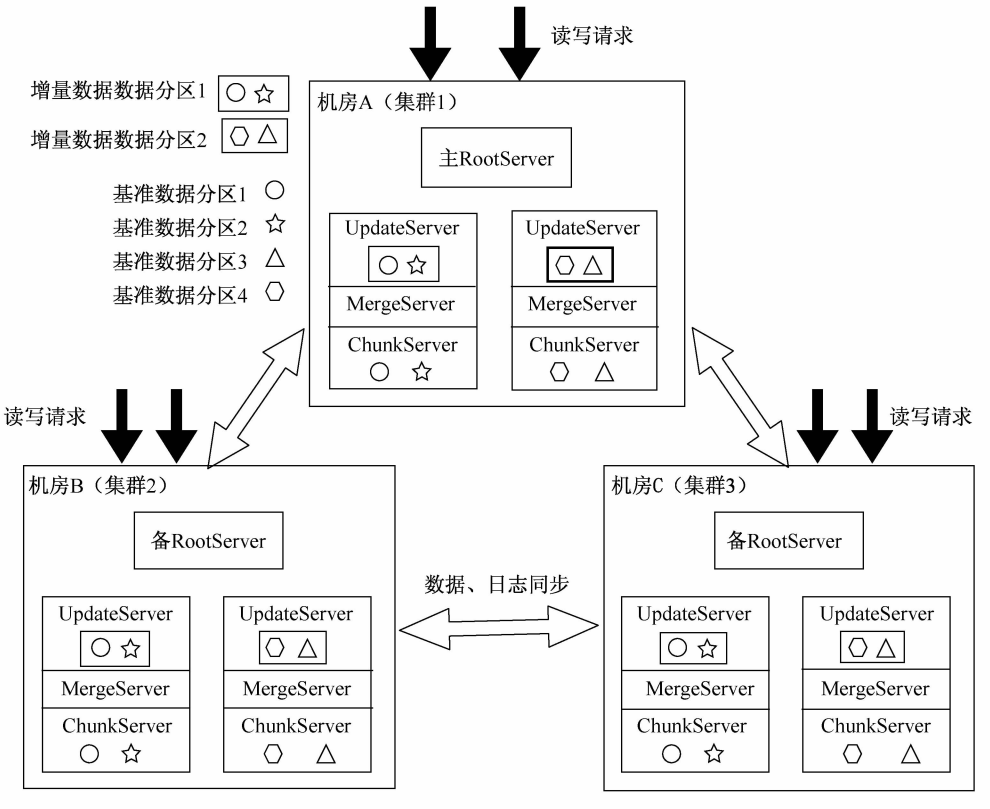

图 5 节点对等架构模型

Fig. 5 The P2P architecture

这种架构将增量数据和基准数据存放于同一物理机上，在处理读写请求时减少了基准数据与增量数据之间的网络开销，大大提高了系统的读写性能. 在提升系统性能的基准上采用增量数据分区主副本强同步和分布式选举来保证系统的一致性和高可用性，并且采用数据分区的方式改善了单点 UpdateServer 的瓶颈. 在事务处理方面，依然需要采用适应业务的分区规则避免大量分布式事务的存在，采用两阶段提交协议、加行锁机制和 MVCC 来保证事务原子性和一致性并且能够提供 Read Committed 和快照(Snapshot)读两种隔离级别. 这种架构不仅可以实现系统的高可用性和事务强一致性，而且能够保证系统的高性能和高扩展性，但是数据副本之间是最终一致性.

4 结 论

本文根据传统数据库和分布式数据库一致性和可用性技术，分析了 OceanBase 0.4.2 和 OceanBase 0.5 的一致性和可用性. OceanBase 0.4.2 和 OceanBase 0.5 分别采用了不同的高可用架构和一致性协议来保证系统的高可用性和事务一致性，为了兼顾系统性能数据副本之间满足最终一致性. 虽然 OceanBase 具有高可用性、一致性、扩展性并能支持跨行跨表事务，但单点 UpdateServer 处理写请求和处理读写请求时基准数据与增量数据之间的网络开销将大大影响系统的读写性能. 本文针对 OceanBase 的缺陷提出了 3 种新的架构，并探

讨了新架构一致性和可用性的实现方案. 5 种架构总结参见表 1.

表 1　5 种架构总结

Tab. 1　Summary of architectures

架构	数据副本一致性	可用性/%	事务隔离级别	是否支持跨行跨表事务	存在的缺陷
OceanBase 0. 4. 2	弱一致性	99. 99	Read Committed、Snapshot Isolation	是	单点 UpdateServer 瓶颈、大量网络开销
OceanBase 0. 5	弱一致性	99. 99～99. 999	Read Committed、Snapshot Isolation	是	单点 UpdateServer 瓶颈、大量网络开销
数据镜像架构	最终一致性	99. 99	Read Committed、Snapshot Isolation	是	一致性很难保证、大量网络开销
数据分区架构	弱一致性	99. 99～99. 999	Read Committed、Snapshot Isolation	是	大量网络开销
节点对等架构	弱一致性	99. 99～99. 999	Read Committed、Snapshot Isolation	是	副本弱一致性

OceanBase 目前支持的一致性和可用性还不能达到金融领域对数据管理系统 99. 999% 的可用性和事务可串行化隔离级别的要求，并且 OceanBase 支持的 SQL 查询不适用于金融行业复杂的交易. 在使用 OceanBase 分布式数据库过程中发现 OceanBase 支持的 SQL 数据类型和函数并不完整，不支持二级索引和复杂 Join，因此 OceanBase 目前只适用于读多写少的简单查询应用，如果要运行复杂业务时，需要对业务逻辑进行改造来提高查询效率. 虽然 OceanBase 分布式数据库要完全应用于金融领域还存在着很大的距离，但它实现一致性和可用性的策略与技术对分布式数据库的研究和实现有很大的参考价值. 在运用和学习 OceanBase 时，应该注意以下几点：

1) OceanBase 不支持复杂 Join，并且两张大数据量表连接时的效率很低，在业务应用时需要考虑效率问题；

2) OceanBase 不支持查询优化，如不支持二级索引，不支持 in 主键前缀查询；

3) OceanBase 没有存储过程和游标；

4) OceanBase 代码是由几个版本融合在一起的，存在废弃的接口和数据类型；

5) OceanBase 有自己的内存管理机制，在开发时不要盲目申请内存空间，以免导致内存泄露.

[参　考　文　献]

[1] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: a distributed storage system for structured data[C]// Proceedngs of the 7the symposium on OSDI, 2006: 205-218.

[2] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Trans Comput Syst, 2013, 31(3): 8.

[3] 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.

[4] BREWER E A. Towards robust distributed systems (abstract). PODC 2000:7.

[5] 黄剑. 基于 Oracle Data Guard 懂得容灾策略设计与实现[J]. 科技广场，2006(11)：71-73.

[6] Oracle Berkeley DB: Performance Metrics and Benchmarks, [EB/OL]. 2006[2014-08-31]. http://ww. oracle. coms/database/berkeley-db. html.

[7] SIKKA V, FÄRBER F, GOEL A K, et al. SAP HANA: The evolution from a modern main-memory data plat-

form to an enterprise application platform[C]. PVLDB,2013,6(11):1184-1185.

[8] STONEBRAKER M, WEISBERG A. The VoltDB Main Memory DBMS[J]. IEEE Data Eng Bull, 2014, 36(2): 21-27.

[9] LAMPORT L. Time, clocks, and the ordering of events in a distributed system[J]. Commun ACM (CACM), 1978, 21(7):558-565.

[10] TANENBAUM A S, VAN STEEN M. Distributed Systems: Principles and Paradigms[M]. 2nd ed. [s. l.]:Prentice Hall, 2008.

[11] LAMPORT L. The Part-Time Parliament[M]. ACM Transaction on Computer Systems, 1998, 16(2):133-169.

[12] BORTHAKKUR D. HDFS Architecture Guide [EB/OL]. 2013-02-13.

[13] Two-phase commit protocol. http://en. wikipedia. org/wiki/Two-phase_commit_protocol.

[14] BURROWS M. The chubby lock service for loosely-coupled distributed systems[C]//7th USENIX Symposium on OSDI 2006: 335-350.

[15] GRAY J N, LORIE R A, PUTZOLU G R, et al. Granularity of Locks and Degrees of consistency in a Shared Data Base[R]. San Jose, California: IBM Research Laboratory.

[16] BAILIS P, DAVIDSON A, FEKETE A, et al. Highly available transaction: Virtues and Limitations[C]//Proceedings of the VLDB Endowment, 2014,7(3):181-192.

[17] DEAN J, GHEMAWAT S. MapReduce: Simplified Data Processing on Large Clusters[C]//7th USENIX Symposium on OSDI,2004.

(责任编辑 王善平)

(上接第 102 页)

[10] SHAFER J, RIXNER S, COX A L. The hadoop distributed filesystem: Balancing portability and performance [C]//Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on. IEEE, 2010: 122-133.

[11] BORTHAKUR D. The hadoop distributed file system: Architecture and design[EB/OL]. Hadoop Project Website. 2007-11-21[2014-08-30]. http://hadoop. apache. org/core.

[12] ENGLE C, LUPHER A, XIN R, et al. Shark: fast data analysis using coarse-grained distributed memory[C]. SIGMOD, 2012:689-692.

[13] HE Y Q, LEE R B, HUAI Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems[C]. ICDE, 2011:1199-1208.

(责任编辑 赵 伟)