计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(6):75−81 [doi: 10.15888/j.cnki.csa.008008]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

分布式关系型数据库研究与金融行业应用①

朱哲哲, 赵振海, 李　鹏, 吴海洋, 向小佳

(光大科技有限公司, 北京 100049)
通讯作者: 赵振海, E-mail: zhaozhenhai@ebchinatech.com

摘　要: 数据库作为金融信息化建设的重要组成部分, 需要面对持续的业务量增长、高度可用性和扩展性等挑战, 而以 MySQL、Oracle 等为代表的传统数据库单点架构, 在可用性、扩展性和存储能力上已经无法满足当前的金融服务要求. 分布式数据库的出现, 旨在解决单机数据库所面临的各种挑战, 提供更加灵活的架构, 保障系统稳定运行. 为此, 本文在结合实际的金融业务需求下, 研究实现了具有分布式事务支持、分布式 SQL 引擎、混合事务分析处理等特点的分布式数据库. 系统采用全组件的冗余设计, 通过类 Raft 增强一致性算法保证了存储层高可用和数据强一致, 同时利用基于 Zookeeper 的集群调度方案保证调度层的高可用.

关键词: 分布式系统; 数据库; MySQL; 金融业; 高可用

引用格式: 朱哲哲,赵振海,李鹏,吴海洋,向小佳.分布式关系型数据库研究与金融行业应用.计算机系统应用,2021,30(6):75–81. http://www.c-s-a.org.cn/1003-3254/8008.html

Distributed Relational Database Research and Its Application in Financial Industry

ZHU Zhe-Zhe, ZHAO Zhen-Hai, LI Peng, WU Hai-Yang, XIANG Xiao-Jia

(Everbright Technology Co. Ltd., Beijing 100049, China)

Abstract: As an important part of financial informatization, the database faces the challenges of continuous business growth and high availability and scalability, while the traditional single-point architecture of databases, represented by MySQL, Oracle, etc., fails to meet the current requirements of financial services in terms of availability, scalability and storage capacity. Distributed databases are designed to address the challenges faced by single-site databases and provide the more flexible architecture, ensuring stable system operation. To this end, this study, subject to actual financial service requirements, researches and implements a distributed database equipped with a distributed SQL engine, which is capable of distributed transaction support and hybrid transactional/analytical processing. The system is designed with full component redundancy. In addition, high availability of the storage layer and strong consistency of the data are ensured by the Raft-like enhanced consistency algorithm, while high availability of the scheduling layer is guaranteed by the Zookeeper-based cluster scheduling scheme.

Key words: distributed system; database; MySQL; finance industry; high availability

数据库的发展经历了从传统的关系型数据库、NoSQL (Not only SQL)[1] 数据库到近几年新出现的分布式 NewSQL 数据库, 整个趋势由单机逐渐向分布式方向发展. 传统关系型数据库的问题在于, 对可扩展性以及高可用性的支持不足, 单机数据库容量有限, 单表容量有限, 无法支撑庞大的数据查询处理需求. 一种解决办法是通过分库分表将负载分散到多个表中, 缓解数据库压力, 或者增加磁盘和 CPU 容量, 但不能从根本上解决问题. 而 NoSQL 数据库虽然原生支持扩展性和高可用性, 数据模型更加灵活, 但是对数据的强一致

① 收稿时间: 2020-10-19; 修改时间: 2020-11-18; 采用时间: 2020-12-18; csa 在线出版时间: 2021-06-01

性却无法保证, 不支持原子事务 (原子事务保证了跨多个节点事务操作的一致性, 但是节点间协调会增加延时, 基于性能原因的考虑, 当前 NoSQL实现的都是基于单一键的事务), 而且 NoSQL 不兼容传统的 SQL 语法, 造成原有系统的迁移障碍. 分布式 NewSQL 数据库的概念产生来源于 Google 于 2012 年发表的 Spanner[2] 数据库, 该论文将传统关系模型以及 NoSQL 数据库的扩展性相结合, 使数据库同时支持分布式又具有传统 SQL 的能力. 除了 Spanner 数据库, 国外的 CockroachDB[3] 以及国内的 TDSQL、MyCAT、TiDB[4]、OceanBase 等都是新兴的分布式数据库产品. 分布式数据库从实现上可以分为两类: 一类是在现有数据库之上以中间件代理的形式, 提供自动分库分表、故障切换、分布式事务等支持, 以 MyCAT、TDSQL 等为代表. 一类是原生的分布式架构, 通过共识算法实现高可用性和数据一致性等支持, 以 TiDB、OceanBase 等为代表.

分布式数据库常见的特点包括高可用、高可扩展和数据强一致性等. 对于可用性和数据一致性, 根据分布式 CAP[5] 理论可知, 在保证分区容错性的前提下, 势必要在数据一致性和可用性中做出权衡. 其中可用性的保证可以通过复制实现, 通过在多台机器上保存数据副本, 提高系统可用性, MyCAT 以及 TDSQL 均支持主从复制的方式. 传统主从复制方式的问题在于虽然保证了可用性, 但是数据的强一致性却无法保证, 如果主库故障, 可能会出现多个节点成为主库 (脑裂问题[6]), 导致数据丢失或损坏. MySQL 在 5.7 版本推出了 MySQL Group Replication 功能, 实现了基于 Paxos 共识算法[7] 的可用性和数据一致性保证, TiDB 基于 Raft 共识算法[8] 保证了可用性和数据一致性. 扩展性的保证可通过分区的方式, 将原有单个节点的压力分散到多个节点, 提升系统性能. 分区面临的问题是如何将数据和查询负载均匀分布在各个节点, 常见的解决办法有基于 Hash 的分区和基于 Range 的分区, TiDB 使用 Range 的方式分区, 而 OceanBase 两种都支持. 分布式数据库的另一个特点是对事务的支持, 分布式场景下保障事务的 ACID 原则常见的办法有 2PC 协议、TCC 协议以及 SAGA 协议, TiDB、OceanBase 等均使用 2PC[9] (两阶段提交协议) 来实现跨多个节点的事务提交. 除了对以上特点的支持, 分布式数据库还具有 HTAP、SQL 引擎、兼容性等特点.

对金融行业来说, 在数据库系统可用性、数据一致性、交易延时等方面有更严格的要求, 系统可用性要求 99.99% 以上, 账务交易对事务要求零容忍. 本文基于国内某银行原有金融业务的分布式改造进行研究, 银行原有数据库架构采用 Oracle RAC 部署的方式, 这种方式的问题在于硬件成本高昂, 无法有效横向扩展, 随着业务量的增大, 系统性能以及存储能力严重不足, 故障切换时间长, 更无法实现秒级切换, 导致系统运行风险增大. 基于银行业务对数据强一致性、高可用、横向扩展等要求, 并结合银行自身业务系统的特点, 本文在 MySQL 集群作为数据节点的基础上, 利用自研的调度组件 Grid, 实现了具有高可用、动态扩展、分布式事务、HTAP 混合、兼容 MySQL 等特性的分布式关系型数据库. 该数据库通过类 Raft 增强一致性算法和 Zookeeper 自动选主实现了整个系统组件的高可用, 并通过 Grid 组件调度, 实现了自动分库分表、SQL 路由、分布式事务等功能的支持.

1 架构

在架构设计上, 本文的分布式数据库由调度节点、数据节点和配置节点构成, 如图 1 所示. 其中, 数据节点基于 MySQL 集群方案, 负责数据存储、本地事务管理、本地结果集计算等功能, Grid 调度节点负责全局事务管理、分布式执行计划的生成和调度、集群扩缩容以及数据节点的高可用调度等功能, 而配置节点基于 Zookeeper[10] 实现, 负责集群运行态元数据的存储同步以及配置管理.

在数据节点的实现上, 与 TiDB 以及 OceanBase 等分布式数据库不同的是, 本文使用了 MySQL 集群的方案. MySQL5.7 版本之前, 实现高可用一般的做法是主从复制的方式, 常利用半同步保证节点间数据的一致, 但仍无法满足高可用的要求[11]. 本文采用了基于 Raft 实现的增强一致性算法 (将在第 3 节存储中详细阐述), 通过增强 Raft 算法的部分细节实现, 提供数据库集群节点的强一致保证, 解决了传统主从复制容易出现的脑裂问题.

调度节点 Grid 主要用来进行 SQL 解析, 实现分布式的 SQL 语句执行优化. 对于混合事务分析处理 (HTAP)[12] 的需求, 除了原生支持分布式事务之外, 对于在线分析处理需求, 通过插件的形式提供 Spark SQL 集成, 将复杂语句通过 SQL 引擎路由到 Spark 端处理分析, 而 Spark 的数据则直接通过数据节点进行抽取. 对于分布式事务的实现, 采用了两阶段提交协议的标准实现——

XA 事务. 本文的分布式数据库中所有组件都是高可用和高扩展的, 以调度节点为例, 在对外服务中利用 F5 提供负载均衡支持[13], 组件本身基于配置节点实现分布式选主, 实现多活高可靠.

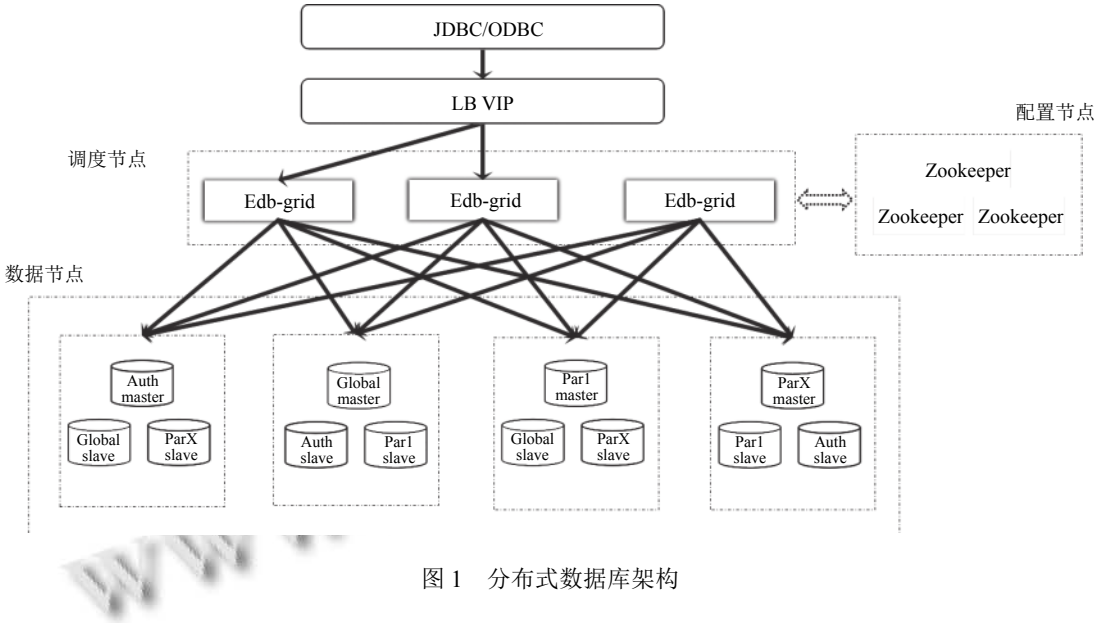

图 1 分布式数据库架构

配置节点层主要用来存储调度层元数据, 提供调度节点的“领导选举”.

除了上述提到的组件, 本文也在数据库安全上做了相应支持, 通过黑白名单检查、登录认证和 ACL 认证等保护数据库安全.

下面的章节本文将会对调度层 SQL 引擎、事务和锁的实现、存储层实现和高可用等作更深入的阐述.

2 调度

为了解决大规模数据量以及高负载下的数据分布问题, 本研究通过中间件的形式提供了对分库分表、SQL 引擎、事务与锁等功能支持. 调度层 Grid 采用了 XA 两阶段事务以及对悲观锁和全隔离级别的支持. SQL 引擎通过词法分析对执行语句进行分布式改造, 实现查询下推和结果汇聚.

2.1 SQL 引擎

分布式场景下 SQL 语句执行[14] 的一个要点是如何跨多个数据分片计算并收集结果, 如何实现跨节点 JOIN. 解决方案在于通过某一种维度进行切分操作, 分步 JOIN 之后再将结果聚合. Grid 组件在 SQL 引擎的实现上对语句进行了并行改造, 通过对语句添加后缀将其路由到不同的数据分片实现并行计算. 执行过程如图 2 所示.

对于一个查询语句来说, SQL 引擎首先对语句改造后生成查询计划, 然后最大限度的将查询下推, 在多个节点进行并行查询计算, 并进行结果归并. 结果归并的过程采用了流式归并, 其与数据库原生的返回结果集的方式最为契合. 流式归并将每一次从结果集中取到的数据, 通过逐条获取的方式得到正确的单条数据. 具体来说, 一个查询语句经过 SQL 引擎的步骤如下:

(1) 通过词法语法解析获取 SQL 语句语义信息.

(2) 对 SQL 语句进行必要的分布式并行执行改造.

(3) 尽可能地将查询计算下推.

(4) 当涉及到多个数据分片, 查询会并行下发到多个数据分片进行并行计算.

(5) 根据执行计划对并行计算的结果集进行归并.

(6) 归并过程尽可能采用流式归并.

(7) 对于复杂的跨节点 JOIN, 通过分析语义将逻辑上独立的子查询块拆分出来并行执行.

对于上述步骤描述的并行执行计算, Grid 在实现上采用了物理与逻辑结合的方式, 通过物理表拆分与 SQL 逻辑拆分相结合的数据分发及并行处理技术, 如图 3 所示, 实现在数据分布式部署和并行查询执行计划两个层面的性能优化, 相对于单数据库实例, 查询效率提升 N 倍 (N=逻辑拆分数×虚拟分片数).

跨节点 JOIN[15] 作为分布式数据库实现的难点之一, 需要对普通表和分区表同时进行支持, 跨节点 JOIN 指的是多个节点的联表查询, 比如“SELECT…FROM t1, t2 WHERE”查询语句, 涉及到两张表以上的查询. 如

何对多个节点数据多个表进行 JOIN, 调度层 Grid 采用的做法是通过引擎将复杂 JOIN 拆分为多个子句, 将子句划分为表组 (Table Group), 生成执行计划, 通过创建临时表的方式进行迁移[16], 如图 4 所示. 以子句 “SELECT … FROM tb1, tb2, tb3 WHERE”为例, 首先会根据可以 MERGE 的 tb1 和 tb3 表生成一个表组, tb2 为另一个表组, 然后创建临时表迁移. 迁移的过程包含以下步骤:

(1) 在大表 tb2 中创建记录 tb1, tb3 的临时表.

(2) 生成 tb1、tb3 的 SELECT 计划 1 数据下推.

(3) 生成 tb2 的 SELECT 计划 2 同时创建临时表.

(4) 执行 SELECT 计划 1 移动到 tb2 中的临时表.

(5) 执行 SELECT 计划 2.

对于普通表和分区表的 JOIN, 会在分区表的每一个节点创建临时表, 然后将普通表迁移到分区表每个节点的临时表中, 对于有等值分区列的 JOIN 条件, 则会将普通表按照分区表的分区算法散列到不同节点. 如果子句中的表都是分区表, 限制查询 WHERE 语句必须包含分区列, 将表按照分区散列算法迁移到多个节点. 迁移完成后, 在分区节点中并行计算 JOIN 结果并返回.

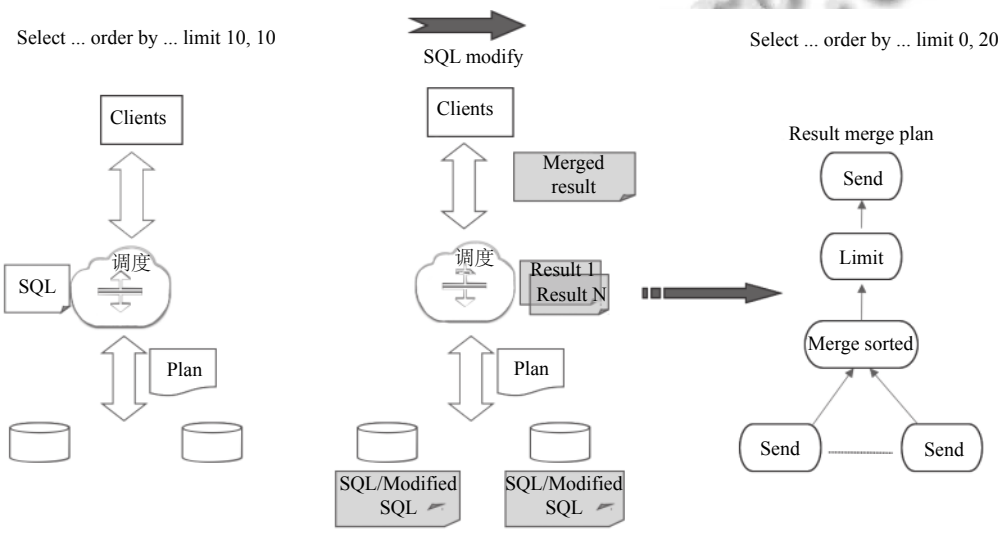

图 2 SQL 执行过程

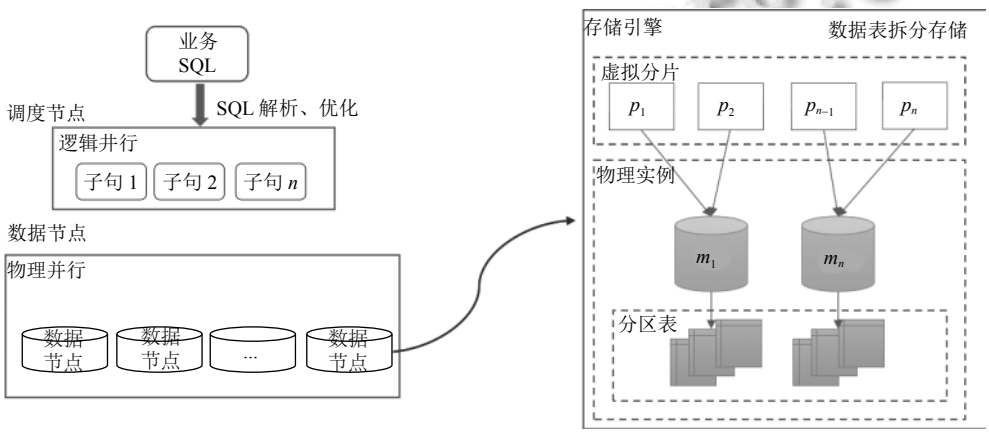

图 3 并行查询

2.2 事务与锁

对事务的支持是分布式数据库和 NoSQL 数据库的一个主要区别, 事务在数据库中代表一系列操作要么全部完成, 要么全部失败[17], ACID 规定了事务操作的原子性、一致性、隔离性和持久性. 在分布式情况下要保证 ACID, 常见的做法是使用两阶段提交算法 (2PC). 本文的分布式事务借鉴了 XA 两阶段事务的实现, 通过事务管理器协调多节点之间的事务, 整个事务

被分为准备和提交两个过程[18].

2PC 存在事务管理器故障导致数据库节点无法获知事务提交状态的问题. 在事务提交时, 事务管理器崩溃或者网络出现故障, 而数据库节点无法获知当前事务是否提交成功或者回滚, 只能处于等待状态. 常见做法是将事务提交或者终止的状态写入日志, 在事务管理器恢复后, 可以通过日志信息继续事务过程. Grid 在实现上将分布式事务日志存储在数据节点的元数据表中, 同时提供主备数据副本支持. 事务日志在数据节点中使用类 Raft 增强一致性同步协议确保数据强一致 (类 Raft 协议的实现将在存储一节详细介绍), 事务日志中包含了关联节点的信息和提交状态, 在出现故障时, 通过关联事务日志列表和状态日志汇总结果计算出事务提交列表以及回滚列表, 将结果应用到对应的节点上.

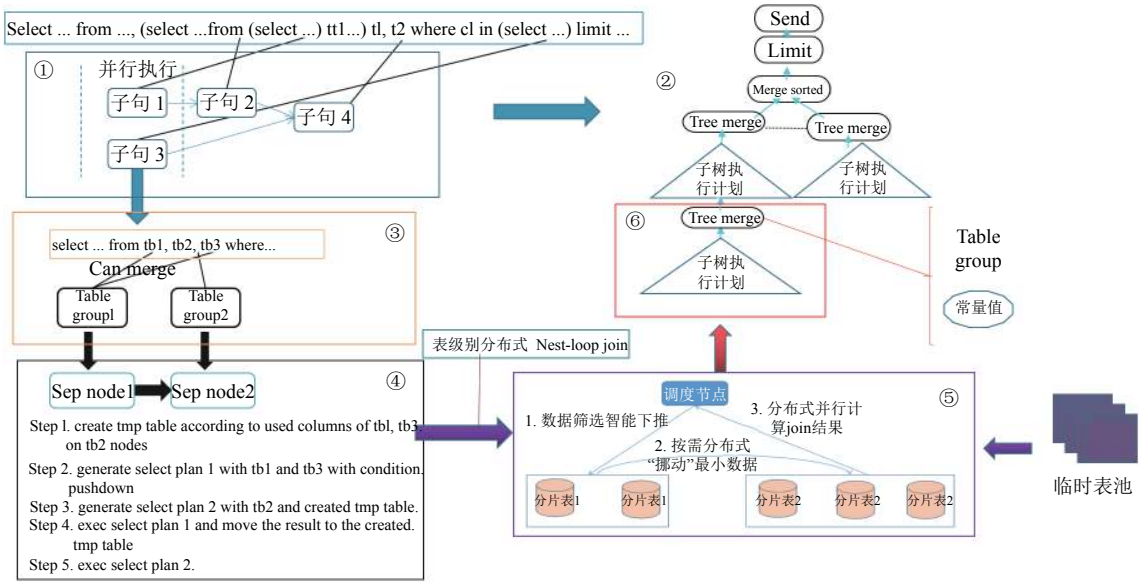

图 4 跨节点 JOIN

在锁的支持上, Grid 实现了与 MySQL 一致的支持, LOCK TABLE 会导致隐式提交, 而 DDL 语句导致事务的隐式提交, 同时支持显式行锁以及 FLUSH TABLE 全局锁.

3 存储

分布式数据库存储层常见的方案有使用关系型数据库 (MySQL 等) 和非关系型的键值数据库, 比如 TiDB 和 CockroachDB 都使用键值数据库 RocksDB[19] 作为其存储层实现. 键值数据库常见的底层存储模型包括 LSM-Tree[20]、B-Tree 和 Hash 模型, LevelDB 及其变种 RocksDB 皆使用 LSM-Tree 模型, 内存型数据库 Redis 使用的是 Hash 模型, 对于关系型数据库来说, 其底层存储模型常使用 B 树或者其变种, 比如 MySQL 使用的就是 B+树. 本文在底层数据库中使用 MySQL 数据库作为数据集群, 这样可以最大限度的兼容原有 MySQL 数据库, 而以 TiDB 等为代表的数据库需要在存储层和 MySQL 的兼容性方面提供转换层, 对 SQL 语法的兼容性也不够好. 在集群的实现上, 原生 MySQL 主从复制实现的可用性方案有诸多问题, 如果为同步复制, 存在从节点失去响应后导致主节点无法处理写入操作的问题. 另外主节点故障后如何选择新的领导者 (共识问题), 还可能出现脑裂的现象. 本文使用类 Raft 的增强算法来保证数据的强一致性.

Raft 作为经典的共识算法, 在 2014 由 Ongaro 根据 Paxos 算法简化而来[8], Raft 通过领导选举机制、日志复写和安全性机制保证集群节点在状态转换间的一致. 除了对原有的算法实现外, 本文在具体的算法细节上也做了相应增强, 在实现上与 Raft 算法中元素的对应关系如表 1 所示.

表 1 Raft 算法对应实现

Raft元素	对应实现
节点角色	Leader: 主副本; Follower: 备副本
Raft日志	基于Binlog实现
状态机	表引擎

在领导选举的实现上, 通过数据副本的复制指向关系进行投票, 领导者在任期内的 Term 通过 master_version 进行维护, 如果主节点发生故障, 调度节点 Grid 会通过 RequestVote RPC 进行冗余节点探测来触发新投票过程, 在投票过半数的情况下, 首先遍历所有存活候选从节点, 选取 master_version 最大的节点, 并且检查候选节点是否有巨大的复制延迟, 首先选择从节点版本最新的节点作为主节点, 如果无可用节点的话, 则从延迟过大的节点中选择. Raft 中主节点使用 AppendEntries RPC 来向从节点同步日志和检测从节点状态, 本文的 AppendEntries RPC 实现基于半同步复制, 从节点直接对日志进行应用, 同时基于闪回处理故障恢复节点[21,22]. 具体过程如图 5 所示.

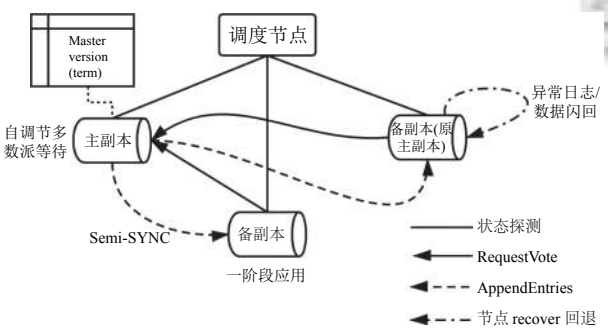

图 5 类 Raft 增强一致性算法

4 高可用

在分布式数据库的设计上, 本文通过全组件的冗余设计, 保障了数据库系统的高可用. 在调度节点通过 LVS 或 F5 对外提供服务, 而调度节点本身基于配置节点进行分布式领导选举, 保证节点可用性. 数据节点的高可用保证通过采用类 Raft 增强一致性同步协议实现. 配置节点基于 Zookeeper 实现了原生高可靠的支持, 支持双机房切换恢复.

调度节点作为分布式数据库的核心, 依赖配置节点存储调度信息. 调度节点为主从模式, 其关键在于主节点故障时如何在从节点中选取新的领导者, 同时由于调度节点相对于数据节点不涉及业务数据, 只有少量元数据需要同步, 因此调度节点 Grid 直接使用 Zookeeper 中的 Zab 算法[23] 来保证. 调度节点之上基于 F5, 从硬件层面保证负载均衡实现多活高可靠. 如图 6 所示.

数据节点的高可用通过 MySQL 数据库集群, 使用类 Raft 增强一致性算法实现单节点宕机后其他节点也能正常提供服务, 保障高可用, 同时也保证了节点间数据的一致, 具体实现细节在存储一节有介绍.

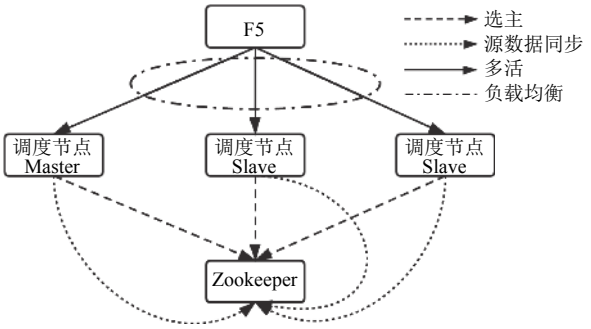

图 6 高可用

除了高可用外, 在扩展性方面, 系统中的所有节点均可动态添加和删除, 不影响系统运行. 其中, 调度节点依赖 Zookeeper 同时实现了集群节点的动态增减, 保证了整个集群吞吐性能的横向扩展, 通过从节点的在线添加与在线删除实现节点在线滚动升级. 而数据节点的扩展, 同时支持逻辑和物理上的在线单调重分布.

5 实验与应用

为了更直观的了解本文分布式数据库的性能, 实验采用标准的 TPC-C 数据库基准, 使用国产鲲鹏服务器集群对分布式事务性能 (TPMC) 进行测试. 其中, 每台鲲鹏服务器配置为泰山服务器 2280 v2、两个 KunPeng920 64 核 CPU、内存 64 GB、系统盘和数据盘分别为两个 600 GB SAS 盘、12 个 800 GB SAS SSD. 实验通过控制集群节点数的增加, 测试数据库在事务处理上的性能, 测试结果如图 7 所示.

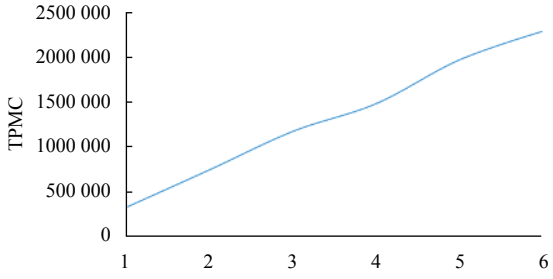

图 7 TPMC 1000 仓库

从图 7 可以看出, 本文的分布式数据库在 3 节点的集群上每分钟最高可以达到 110 w 的事务处理, 随着节点的数的增加, 事务处理能力也线性上升, 在 6 节点集群中事务处理可以达到每分钟 220 w, 横向扩展比大于 0.95. 证明了该数据库在性能上以及扩展上的优秀表现.

目前, 分布式数据库已经在国内某银行业务系统中使用, 经过技术方案、前期业务功能试点, 该系统全

部缴费业务单元已经逐步迁移到分布式数据库系统中.

原有数据库经过分布式改造之后, 在架构上更加清晰, 分布式数据库对外仍然像单机数据库一样提供服务. 在系统部署架构上, 采用双机房运行的方式, 利用两台备份服务器和两台逃生服务器保障数据完整, 在数据节点的配置上, MySQL 集群节点采用一主一从和两从节点分机房运行的方式, 通过配置保障同机房和跨机房数据强一致性, 调度节点各机房使用三台服务器为系统提供调度保障.

经过对分布式数据库实际运行中的测试, 银行业务系统在联机同步交易单笔查询和缴费响应的时间小于 100 ms, 在可用性方面, 系统整体可用率达到 99.99%, 并且数据不易丢失和损坏. 相较于原有数据库系统, 现有分布式数据库有更灵活的扩展能力, 集群可动态扩展, 各个功能节点发生故障可实时切换, 系统发生故障的次数也大幅度下降.

6 结论

本文研究并实现了基于 Raft 的增强一致性算法和 Zookeeper 集群调度的分布式数据库系统, 通过全节点的冗余设计以及调度层对 SQL 引擎和分布式事务的研究实现, 为大规模金融数据的高可用以及强一致性提供了有力支持. 在数据库测试中表现出了优异的性能, 在实际金融业务系统的稳定运行也表明该系统在金融行业应用的优势. 不过, 本文的分布式数据库在设计与实现上也有不足之处, 比如限于研发难度使用多种一致性算法, 增加了运维难度, 后续会在这方面进一步改进.

参考文献

1 申德荣, 于戈, 王习特, 等. 支持大数据管理的 NoSQL 系统研究综述. 软件学报, 2013, 24(8): 1786–1803. [doi: 10.3724/SP.J.1001.2013.04416]

2 Corbett JC, Dean J, Epstein M, *et al*. Spanner: Google's globally distributed database. ACM Transactions on Computer Systems, 2013, 31(3): 8.

3 Taft R, Sharif I, Matei A, *et al*. CockroachDB: The resilient geo-distributed SQL database. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Portland, OR, USA. 2020. 1493–1509.

4 Huang DX, Liu Q, Cui Q, *et al*. TiDB: A raft-based HTAP database. Proceedings of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

5 Gilbert S, Lynch N. Perspectives on the CAP theorem. Computer, 2012, 45(2): 30–36. [doi: 10.1109/MC.2011.389]

6 Kleppmann M. Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017.

7 de Prisco R, Lampson B, Lynch N. Revisiting the PAXOS algorithm. Theoretical Computer Science, 2000, 243(1–2): 35–91.

8 Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. Proceedings of the 2014 USENIX Annual Technical Conference. Philadelphia, PA, USA. 2014. 305–320.

9 CORPORATE X/Open Company Ltd. X/Open CAE Specification: Distributed Transaction Processing: CPI-C Specification, version 2. Prentice-Hall, 1996.

10 Junqueira F, Reed B. ZooKeeper: Distributed Process Coordination. Sebastopol: O'Reilly Media, 2013.

11 康文杰, 王勇, 俸皓. 云平台中 MySQL 数据库高可用性的设计与实现. 计算机工程与设计, 2018, 39(1): 296–300.

12 姚入榕. 面向 HTAP 的大规模分布式数据库混合存储引擎 [硕士学位论文]. 成都: 电子科技大学, 2020.

13 赵中英, 李斌, 王敏. F5 负载均衡综述. 现代信息科技, 2019,3(2):60-61.[doi:10.3969/j.issn.2096-4706.2019.02.022]

14 Widenius M, Axmark D, Arno K. MySQL Reference Manual. O'Reilly & Associates, 2002.

15 茅潇潇. 分布式数据库并行连接查询的实现及优化 [硕士学位论文]. 上海: 华东师范大学, 2018.

16 杨飞. 分布式数据库中间件 DBScale 的设计与实现 [硕士学位论文]. 哈尔滨: 哈尔滨工业大学, 2015.

17 Delcambre JML, Lisboa ET. Transaction management in a distributed database management system. Proceedings of the 3rd Conference of the European Cooperation in Informatics Trends in Information Processing Systems. Munich, Germany. 1981. 224.

18 Gray J. The Transaction concept: Virtues and limitations (invited paper). Proceedings of the 7th International Conference on Very Large Data Bases (VLDB). Cannes, France. 1981. 144–154.

19 Borthakur D. The history of RocksDB. rocksdb.blogspot.com. 2013.

20 O 'Neil P, Cheng E, Gawlick D, *et al*. The log-structured merge-tree (LSM-tree). Acta Informatica, 1996, 33(4): 351–385. [doi: 10.1007/s002360050048]

21 张晨东, 郭进伟, 刘柏众, 等. 基于 Raft 一致性协议的高可用性实现. 华东师范大学学报 (自然科学版), 2015, (5): 172–184.

22 陈陆, 黄树成, 徐克辉. 改进的 Raft 一致性算法及其研究. 江苏科技大学学报 (自然科学版), 2018, 32(4): 559–563.

23 Junqueira FP, Reed BC, Serafini M. Zab: High-performance broadcast for primary-backup systems. Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems & Networks. Hong Kong, China. 2011. 245–256.