计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(4):125−130 [doi: 10.15888/j.cnki.csa.006860]

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

一种实时数据库的分布式存储方法①

周　淳, 李贤慧

(南瑞集团有限公司 (国网电力科学研究院有限公司), 江苏瑞中数据股份有限公司, 南京 211106)
通讯作者: 周　淳, E-mail: 364073988@qq.com

摘　要: 随着网络技术迅速发展, 各实时系统产生的数据量呈指数级增长, 各业务应用对海量数据的管理和应用的实时性提出了越来越高的要求, 现有的单机实时数据库技术已无法满足需求, 将分布式思想引入实时数据库领域, 主要从数据分布方式、数据冗余备份、数据一致性等方面做了研究, 并提出一种实时数据系统中的数据分布式存储的设计. 该设计可以为当前实时数据库扩展性及可靠性方面提供技术支撑.

关键词: 实时数据库; 分布式; 分布规则; 一致性; 元数据表

引用格式: 周淳,李贤慧.一种实时数据库的分布式存储方法.计算机系统应用,2019,28(4):125–130. http://www.c-s-a.org.cn/1003-3254/6860.html

Distributed Storage Method for Real-Time Database System

ZHOU Chun, LI Xian-Hui

(China Realtime Database Co. Ltd., NARI Group Corporation (State Grid Electric Power Research Institute), Nanjing 211106, China)

Abstract: With the rapid development of network technology, the amount of data generated by real-time system is increasing exponentially. Business applications are putting forward more and more high demand on the management and real-time requirement. As the traditional stand-alone RTDB technology has met the needs with difficulty, the distributed thinking is introduced into the RTDB field. It mainly studies from the Data distribution mode, data redundancy backup, data consistency, and other aspects, proposes a design of data distributed storage in real-time data system. This design can provide technical support for current RTDB scalability and reliability.

Key words: real-time database; distribute; distribution rule; consistency; metadata table

数据库技术产生于 20 世纪年代后期, 其理论与技术发展极为迅速, 应用也日益广泛, 数据库技术的一个重要分支:实时数据库是采用实时数据模型建立起来的数据库, 实时数据库技术是实时系统和数据库技术相结合的产物[1], 作为解决关系型数据库实时处理能力等问题而产生的一种数据库技术, 实时、高效、稳定是实时数据库最关键的指标.

实时数据库在数据通信、数据存储、数据检索、数据访问、数据处理、数据展现等方面的专业化及产品化, 为构建基于大容量实时历史数据之上的分析应用提供了便捷稳定的数据支撑, 使应用系统可以从更高更深层次充分利用宝贵的生产实时历史数据. 而随着网络技术迅速发展, 实时系统产生的数据量呈指数级增长, 且各业务应用对海量数据的管理和应用的实时性提出了越来越高的要求, 现有的单机实时数据库技术不论在理论还是实际应用中都存在很大的局限性, 已不能完全适应当前需求.

目前业界使用实时数据库基本仍使用单核心集中部署[2,3], 分布式计算和存储技术思想的引入则可以很好的解决单机实时数据库遇到的这些瓶颈, 但如何设计一个满足事务应用处理时效性以及事务吞吐率的需求的分布式实时数据库是一个难点, 本文通过对分布

① 收稿时间: 2018-09-12; 修改时间: 2018-11-12; 采用时间: 2018-11-23; csa 在线出版时间: 2019-03-28

式实时数据库部分关键技术 (数据分布、数据一致性、数据重分布) 的研究, 设计了一种分布式实时数据库管理系统, 可以为实时数据库应用发展提供技术支撑.

1 概述

1.1 总体架构设计

分布式实时数据库部署架构如图 1 所示.

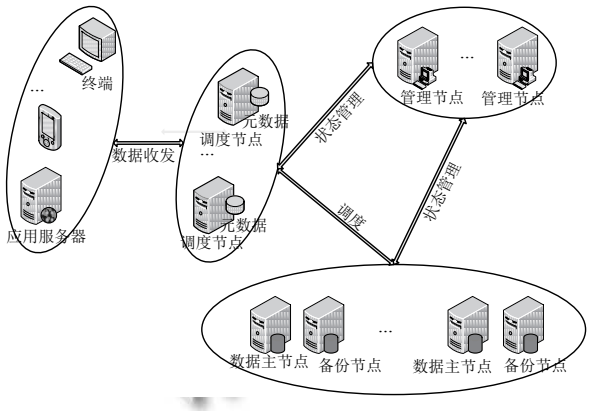

图 1 分布式实时数据库系统部署架构

在部署架构中, 管理节点是整个分布式实时数据库的管理者, 主要存储系统元数据信息, 包括数据分布方式、各节点状态、主备节点一致性状态等关键信息.

调度节点属于分布式访问层, 统一的访问接口使得应用能够将分布式的实时数据库视为一个完整的逻辑整体进行访问; 同时调度节点也属于分布式定位层, 是数据的分布和收集者, 主要负责数据的分发, 查询结果的收集汇总以及任务调度. 在分布式查询访问多个节点上的数据时, 并发式的访问处理能够实现多路数据请求在多个存储节点上的并行处理, 从而实现了高效的分布式数据访问.

数据节点属于分布式存储层. 每一个数据节点运行和管理一个数据库实例. 数据节点负责整个数据库系统数据的实际存储, 接收来自调度节点的数据, 执行经分解的查询任务, 执行结果通过调度节点返回给应用程序. 数据节点的数量仅受限于以太网带宽、机房物理条件等硬性条件. 各数据节点只存储属于相应分区的数据, 逻辑上是对等的. 主备数据节点实现数据在节点间的冗余互备.

1.2 元数据表

实时数据库可以简单地理解为由标签点信息库、值数据库两个部分组成[4]. 如图 2 所示.

标签点信息库含有测点基本信息的一张表, 以标签点标签 (*point_name*) 作为关键字, 一条记录包含一个标签点的基本配置信息, 如标签点描述、压缩算法等. 用户可从此标签点信息库中查询标签点的基本信息. 值数据库包含了实时值缓存和历史数据存储, 每条记录反映了某标签点产生的实时数据的时间戳、数值、质量等信息. 用户可从值数据库中查询实时数据值. 因此实时数据库最主要的两个维度即标签点和数据时间.若需要将实时数据库所有数据分布到多个节点中, 则应该从这两个维度入手.

标签点信息表

标签点name1	标签点描述1	标签点ID1	压缩算法1	...
标签点name2	标签点描述2	标签点ID2	压缩算法2	...
...				
标签点name*N*	标签点描述*N*	标签点ID*N*	压缩算法*X*	...

值数据表

标签点name1	时间	Value	状态码	时间	Value	状态码	...
标签点name2	时间	Value	状态码	时间	Value	状态码	...
...							
标签点name*N*	时间	Value	状态码	时间	Value	状态码	...

图 2 标签点表和数据值表

元数据存储: 标签点表作为元数据表, 存储在调度节点, 每个调度节点拥有一份完整的标签点信息, 多个调度节点互为备份, 通过管理节点控制调度节点状态, 使用同步线程做异常恢复.

1.3 数据流

数据接入数据流如图 3, 数据由应用服务器或客户端发往调度节点, 由调度节点根据分布规则, 将数据发送到不同数据主节点, 数据主节点在存储过程中会将数据转发给对应备份节点.

数据查询数据流如图 4, 查询请求和查询条件由应用服务器或客户端发往调度节点, 由调度节点根据查询条件中涉及到的标签点和时间范围, 通过分布规则, 筛选相关的数据节点, 拆分并重新组织多个子查询请求分配到多个数据节点, 多个数据节点并行处理查询, 完成后将结果返回调度节点, 调度节点待所有已分配的子查询返回结果后做聚集处理.将结果反馈到应用服务器或客户端.

这里涉及到分配主备节点的问题, 每一个子查询只可能被发送到某一个数据主节点或者备份节点, 除非查询异常才会继续由调度节点发往另一个节点做查

询 (同时需要向管理节点汇报节点状态), 调度节点可以通过判断主备节点的繁忙程度来选择分配节点, 数据节点定时向管理节点汇报当前活动状态以及繁忙程度, 调度节点也会定时从管理节点同步所有数据节点状态, 繁忙程度的度量[5]可以从 CPU 平均使用率、平均网络使用率、当前磁盘使用率、当前内存占用率等方便综合测算.

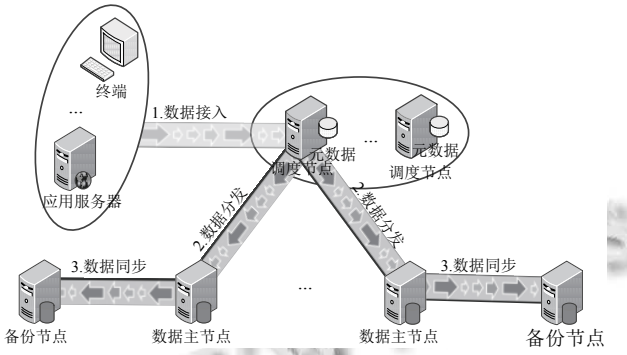

图 3 数据接入的数据流

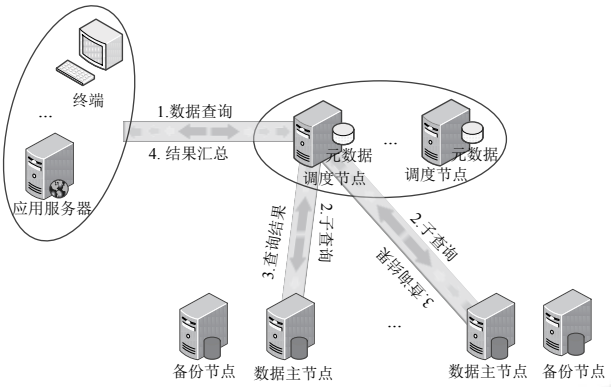

图 4 数据查询的数据流

2 数据分布设计

2.1 分布规则

设计分布规则时, 既要考虑到标签点数据的关联性 (如果能够将部分查询运算放在数据节点内部直接处理, 显然要比将数据传输到调度节点后再集中做运算性能要高的多), 又要考虑分布式架构并行处理带来的效率提升.

可以考虑将连续的一批标签点, 一个时间段的数据分布在同一个数据节点中, 可以考虑通过自定义的 Hash 函数实现, Hash 函数类似如下格式:

$$slice\_id = (w1 \times (hash\_str(point\_name)/b1) + w2 \times (day\_time(time)/b2))\%Hash\_Bucket$$

其中 *slice_id* 为最后函数确定分配的分片号, 通过函数前部计算结果对一个设定的 *Hash_Bucket* 取模获得, *hash_str*(*point_name*) 和 *day_time*(*time*) 分别是对标签点名和时间段的一个量化函数, 考虑到标签点或时间区域内数据的关联性, 设计 $b1$ 和 $b2$ 控制标签点或某相邻时间段内的数据分散程度. $w1$ 和 $w2$ 为系数, 或者称之为未标准化权重, 例如极端情况下设置 $w2$ 为 0, 则表示完全按照标签点来分布数据, 每个标签点的所有时间的数据都存放在同一个数据分片或其备份分片上;同样的, 若设置 $w1$ 为 0, 则表示完全按照时间段来分布数据, 每个时间段的所有标签点的数据都存放在同一个数据分片或其备份分片上.

可以看到, 该等式是以 *point_name* 和 *time* 为自变量, *slice_id* 为因变量的函数, 根据函数定义, 说明任意一组已知的 *point_name* 和 *time*, 可以唯一确定一个 *slice_id*, 也就是说根据接入的数据中的标签点名和数据时间戳, 我们可以分布到一个确定的数据分片.

考虑到系统实际运行中, 节点可能会有调整变动, 若 *Hash_Bucket* 设为初始节点数, 且每个 *slice_id* 对应一个节点, 则节点变动会导致 *Hash_Bucket* 变动, 继而会使得整个数据重分布涉及到系统所有数据文件, 因为需要对每条数据值使用新 Hash 函数计算对应数据节点, 这种方式只能通过提高 Hash 运算以及数据传输的并行度来提高重分布效率.

借鉴一致性 Hash 算法[6]可以有效解决该问题, 管理节点存储一个大小为 *Hash_Bucket* 的数组 (分片映射表), 每个元素存储该元素下标对应数据分片所在的数据节点, 该映射关系可以通过人工设定或者按 *node_num*(数据节点数) 取模的方式自动分配.

图 5 显示了数据是如何根据分布规则分布到不同节点上的, 分布规则包含自定义的 Hash 函数和分片映射表, 通过自定义的 Hash 函数和分片映射表可以将标签点以及数据分布到不同数据节点, 分布规则需要在标签点写入之前确定, 且分布规则一旦确定后, 直到数据节点变动需要重分布才会改变, 且只能改变分片映射表.

若整个分布式实时库系统启动前没有预先设置分布规则, 则启动后分片映射表根据数据节点数自动生成, 接入标签点和数据前, 仍需要通过管理客户端设置 Hash 函数相关参数, 该分布规则直接交由管理节点保存, 调度节点在分配标签点和数据前需要从管理节点

预先获取分布规则.调度节点首次获取到分布规则后会存储到内存中, 无需在每次分配数据时重新获取分配规则, 若系统扩容或缩容等有数据节点的变动, 需要修改分布规则时, 则需要调度节点重新初始化并获取分布规则.

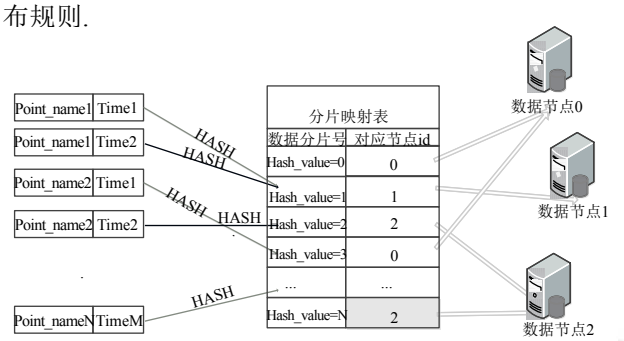

图 5 数据分布规则

2.2 数据重分布设计

随着数据库内数据总量的变动, 可能需要考虑数据节点的扩容或缩容. 数据节点的变动必然需要考虑数据均衡和数据重分布.

前文设计的实时库系统, 采用的分片映射表和 Hash 分布 2 层分布的方式, 重分布策略可以设计为 Hash 函数不做改变, 而只改变分片映射表, 例如重新按照新的节点数自动取模生成新的映射表, 调度节点通过对比新旧分片映射表的差异, 设置每个数据节点相应的分片移动路径并分配到各个数据节点, 数据节点按完整分片目录做数据迁移. 这样以分片为最小移动单位, 无需做数据解压和 Hash 运算, 极大减少 CPU 运算和磁盘 IO 以及网络 IO 的开销.

一个更高效的做法即根据整个数据库系统中数据总量做均分, 图 6 是一个扩容过程, 将数据总量大于平均数据量的数据节点中的部分分片向少于平均数据量的数据节点移动, 这样可以使得数据重分布需要移动的总吞吐量达到最小.

3 单机实时数据库存储设计

实时数据库存储设计尽可能沿用已有的单机实时库设计, 即设计满足单个数据库实例既可以作为独立个体运行, 也可以作为分布式架构中一个数据节点运行, 也就是说单个数据节点需要包含实时库所有模块: 网络收发模块、数据缓存模块、数据处理模块、数据存储模块 (包含索引).

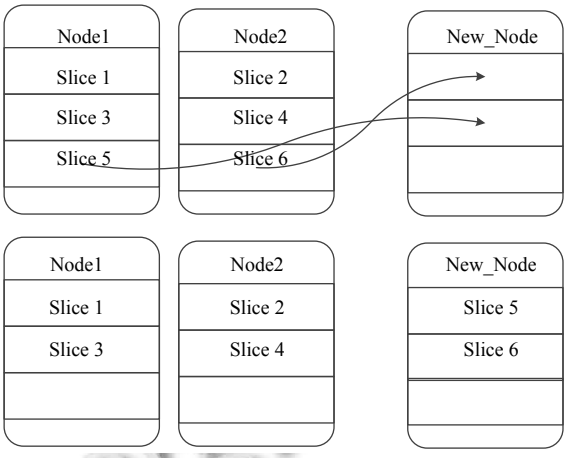

图 6 数据节点扩容时数据分片的移动

这里主要考虑数据存储模块, 数据缓存模块、数据处理模块都是建立在该模块之上, 而网络收发模块只和设计的通信协议有关, 和数据逻辑相对独立.

数据存储按照分片管理, 如图 7 所示, 每个分片号 (*slice_id*) 对应一个文件目录, 若作为单独数据库实例运行, 该实例包含所有分片号, 若作为分布式架构中一个节点, 该实例包含分片映射表中部分分片, 每个分片文件目录下按时间段区分数据文件组存储数据块, 文件组和时间段一一对应, 每个文件组和分片目录各自存储一个版本号 (version), 每次写和修改更新版本号, 查询不做版本号更新[7].

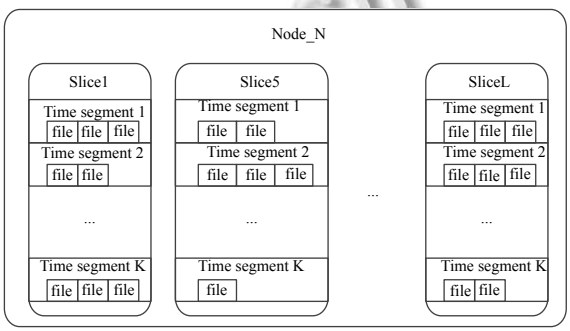

图 7 单数据节点基本存储结构

从整个个分布式系统角度看, 数据节点是系统中最基本的存储和计算单元, 从数据完整性角度来看, 数据节点内部数据文件组是最小粒度的具有自描述性的存储单元, 也是最基本的自恢复和同步单元.

4 数据冗余备份和一致性处理

分布式数据库的一个难点就是如何处理存储数据的一致性, 包括元数据和数据的同步以及异常处理.

4.1 管理节点数据一致性

管理节点一般需要配置多个节点来保障高可用性, 主要存储的是分布规则等关键配置, 以及各主备节点状态等小颗粒信息, 适合基于 corosync 或 zookeeper 类似架构实现, 保证信息同步有序一致, 对于这一类场景目前已经有广泛的应用实践, 例如文献[8].

4.2 调度节点元数据一致性

调度节点除了定时从管理节点同步各节点状态到内存外, 主要存储标签点元数据信息, 对标签点的增删改查以事务方式执行, 需要保证单次操作的原子性和强一致性, 即每次在某一调度节点的操作, 需要保证成功同步到其他所有正常调度节点, 若低于 1/2 的调度节点同步失败, 管理节点将失败的节点做异常标记, 等待后续同步恢复; 若超过 1/2 的节点同步失败, 则在已执行成功节点上执行撤销操作, 若撤销失败, 同样做异常标记, 等待后续同步恢复.

调度节点元数据信息被划分区块, 因为所有操作具有强一致性, 因此正常节点每个区块以及区块顺序都应该是完全一致的. 管理节点定时检测异常的调度节点并进行同步恢复操作.

4.3 数据冗余备份和一致性

通过冗余的镜像机制来保证集群的高可用特性, 设计需要考虑实时性和容错性, 对于主备数据节点数据一致性的选择, 最简单的做法首选强一致性, 但 CAP 理论[9]告诉我们一致性、可用性和网络分区三者不可兼得, 弱一致性也有其适用范围, 特别是在实时性要求较高而对及时一致性要求较低的场景.

主备数据节点的状态由管理节点监控, 调度节点通过管理节点获取主备数据节点的状态, 可以指派数据写入和查询节点, 同时实时任务处理过程中数据节点状态异常也实时反馈给管理节点. 前面已经提到数据写入和查询的的数据流, 写入时调度节点数据直接写入主节点, 由主节点将数据往镜像节点做自动转发同步, 不保证主备节点数据的强一致性, 只保证数据的最终一致性, 查询时主库镜像库分担查询压力, 由调度节点按负载情况分派查询任务.

4.4 数据异常处理

主库节点若发生异常, 备节点会由管理节点指派升级为主节点, 异常节点恢复后会作为备份节点继续运行, 管理节点会在检测到异常数据节点时启动一致性检测, 通过两层版本号检测和文件级同步的方式可以使恢复操作对实时处理性能影响最小化, 且恢复效率高, 具体检测和恢复方式为:通过分析主备数据节点的各分片版本号以及分片内文件组的版本号 (version) 不一致情况, 确定需要同步的索引和数据文件块并进行数据同步, 即对不同版本号文件组直接做替换操作, 数据流通过底层 TCP 传输.

为了不影响整个系统的实时性, 主备数据异常恢复时采用在线同步模式, 即同步过程中不影响数据正常读写, 采用如下策略:数据同步过程中主节点产生的修改记录以 Log 方式记录在主数据节点, 并在同步完成后解析 Log 和更新数据, 解析过程中的数据修改仍然追加到 Log 末尾, 直到所有 Log 解析完毕, 管理节点设置该主备数据节点状态为 Normal.

5 应用实例

如下是一个分布式实时数据库的搭建和设计思路:

(1) 建构系统架构: 创建管理节点集群, 调度节点集群, 数据节点集群, 数据节点服务器的数量和数据量有关, 管理节点因为体量较小, 可以和调度节点合并到同一个服务器集群. 管理节点存储节点元信息, 包括数据分布规则、各节点状态等信息, 调度节点存储测点信息表, 数据节点存储包含测点名、数据时间和数据值的数据表, 每个数据节点通过主备节点实现冗余备份, 管理节点、调度节点和数据节点间分别建立连接, 数据和调度节点定时向管理节点汇报当前活动状态及繁忙程度, 调度节点定时从管理节点同步所有节点状态;

(2) 建立分布规则: 管理节点依据输入参数确定系统唯一的分布规则, 分布规则的设计见 2.1 节;

(3) 测点请求: 调度节点收到测点写请求后, 检测测点合法性, 检测通过执行对测点表的操作, 并向其余所有调度节点做同步, 一致性设计见 4.2 节;

(4) 数据写请求: 调度节点收到数据写请求后根据测点名和数据时间, 依据分布规则, 将数据发送到不同的数据主节点, 数据节点的存储方式见第 3 节, 另外, 如 4.3 节所述, 数据主节点在存储过程中将数据转发给对应的备份节点. 数据存储过程中发生异常会执行恢复流程, 见 4.4 节;

(5) 数据读请求: 调度节点收到读数据请求后, 根据读请求中测点名和数据时间, 通过分布规则筛选数据节点, 拆分并重新组织多个读取子任务分配到多个数据节点并行读取数据, 完成后将结果返回调度节点,

调度节点待所有子任务返回后做聚集处理.将结果反馈到请求端.

为了验证该设计的可行性, 使用 C++语言开发了分布式实时库原型系统, 使用 3 个管理和调度服务器的集群, 5 个数据服务器的集群, 执行了数据读写测试, 并和单机实时数据库做对比. 测试数据来源于电力调度系统模拟数据, 原始数据量均为 10 TB, 测点规模为 1000 万级别, 分别从测点建立, 数据接入效率, 数据实际存储量, 数据随机查询四个方面进行测试对比. 表 1 是对比结果.

表 1 性能测试对比

	测点建立 (万/秒)	数据顺序接入 (万/秒)	随机数据接入 (万/秒)	存储总空间 (TB)	随机数据查询 (万/秒)
单机存储	9.38	120.54	38.62	5.82	1.68
分布式存储	6.68	102.82	74.83	6.8	6.46

测试结果显示, 在磁盘压力较低的情况, 单机存储因为无需做数据分布处理和同步, 性能较高;而在需要做复杂运算或者磁盘 IO 开销大的场景下, 分布式存储比单机存储方式有几倍的性能提升.

6 结语

由于工业领域对实时数据库的性能、可靠性、可扩展性等要求越来越高, 本文通过将分布式技术和实时数据库技术相结合, 提出一种实时数据系统中的数据分布式存储的设计, 同时给出了数据冗余、数据重分布以及数据一致性的一整套方案. 为突破当前实时数据库技术瓶颈提供技术支撑.

参考文献

1 刘云生. 实时数据库系统. 北京: 科学出版社, 2012. 16.
2 崔乐, 石军昌, 张晓华, 等. 实时数据库在工业企业中的应用. 工业仪表与自动化装置, 2015, (4): 73–75.
3 彭晖, 王瑾, 陶洪铸, 等. 适应横集纵贯智能电网调控系统实时数据库的设计. 电力系统自动化, 2016, 40(9): 118–123.
4 剑思庭. 实时数据库和关系数据库的设计特点. 可编程控制器与工厂自动化, 2012, (2): 28.
5 李春晖, 谢永斌. 云计算环境下资源负载均衡调度优化仿真. 计算机仿真, 2017, 34(12): 420–425.
6 Karger D, Lehman E, Leighton T, *et al*. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. Proceedings of the 29th Annual ACM Symposium on the Theory of Computing. El Paso, TX, USA. 1997. 654–663.
7 张伟燕, 吴志杰, 夏涛. 利用版本号保护数据库一致性的 EJB 设计模式. 信息与电子工程, 2004, 2(2): 136–139, 146.
8 倪超. 从 Paxos 到 Zookeeper: 分布式一致性原理与实践. 北京: 电子工业出版社, 2015: 59.
9 Brewer E. Cap twelve years later: How the “rules” have changed. Computer, 2012, 45(2): 23–29.