文章编号: 1000-5641(2016)05-0165-08

可扩展数据管理系统中的网络请求服务机制

肖 冰, 郭进伟, 钱卫宁

(华东师范大学 数据科学与工程研究院, 上海 200062)

摘要: 请求服务机制涉及请求的传输和处理, 是分布式数据管理系统中各组件交互并完成任务的重要前提. 本文以可扩展数据管理系统为背景, 抽象系统中的网络服务模型, 介绍系统中的网络请求服务机制. 从数据库的主要实现出发, 分析不同类型的请求在传输以及处理上的不同要求. 以OceanBase为例, 统计各机制在一个可扩展数据管理系统中的服务比重, 并进行相关的分析.

关键词: 可扩展数据管理系统; 网络请求; OceanBase

中图分类号: TP391 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.018

Network request service mechanisms in a scalable database management system

XIAO Bing, GUO Jin-wei, QIAN Wei-ning

(*Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*)

Abstract: Network request service mechanism involves the transmitting and processing of the requests. It plays an important role for interactive components to cooperate in a distributed database management system. Taking scalable database management systems as background, this paper introduces the inside network service model and the network request service mechanisms. On the basis of the primary implementation in database systems, we analyze different demands from different requests in terms of transmitting and processing. The service distribution of each mechanism and associative analysis are presented based on OceanBase.

Key words: scalable database management system; network request; OceanBase

0 引 言

传统单机模式数据库有限的存储能力及其纵向扩展方式已经不适合互联网应用对海量数据的存储和处理所带来的需求. 随着计算机网络与通信技术的发展, 越来越多的分布式数

收稿日期: 2016-05

基金项目: 国家 863 计划项目(2015AA015307); 国家自然科学基金(61432006)

第一作者: 肖 冰, 女, 硕士研究生, 研究方向为分布式数据库. E-mail: bingxiao@stu.ecnu.edu.cn.

通信作者: 钱卫宁, 男, 教授, 研究方向为可扩展事务处理和海量数据管理与分析.
E-mail: wnqian@sei.ecnu.edu.cn.

据库系统应运而生, 如 Megastore[1], Cassandra[2]等. 这些分布式数据管理系统采用横向扩展的方式, 通过向系统中添加机器, 来达到系统支持更高并发服务的目的[3]. 与集中式数据库相比, 分布式数据库系统包含多个在地理上分离而在逻辑上集中的节点, 通过网络互连通信. 每个节点在网络上都是独立的服务器(也可以引申为数据库进程), 能够自主存储并管理本地数据, 但同时所有节点通过系统能够作为整体对外服务[4].

在互联网高速发展的今天, 数据量呈指数增长, 数据类型也展现出多样性, 大量结构化和非结构化数据需要更加高效地存储和处理[5]. 这不仅使得集中式数据库系统愈加难以满足需求, 甚至传统架构的分布式数据库也不再适应. 为了应对这些挑战, 很多企业, 特别是互联网企业开始寻求新的解决方案, 提出新的分布式存储架构和管理方式, 并实现了成熟的、适应海量数据应用需求的分布式数据库[6]. 新型分布式数据库是一种集群式架构的系统, 内部通过高速网络互连, 数据以多副本形式分布存储于部分节点中, 其最重要的特征是高可扩展性, 因此又被称为可扩展数据管理系统, 如 HBase[7], Spanner[8], OceanBase[9-10]等.

而对于这样的可扩展数据管理系统而言, 请求服务机制真正将系统中的各个功能节点联系起来形成一个整体对外服务, 是系统功能的前提和性能的保证. 图 1 所示为一个可扩展数据管理系统中的各功能节点, 其中, 主控节点 MNode(Master-Node)负责管理集群内所有节点的状态信息, 帮助各个节点协同工作, 管理集群、数据分布以及副本; 事务处理节点 TNode(Transaction-Node)负责处理写事务, 并存储增量更新的数据, 是数据库集群中负载较高的节点之一; 基线数据存储节点 SNode(Storage-Node)存储大部分的用户数据, 提供对数据的读访问支持, 能够自动合并基线数据和事务处理节点上的增量数据返回给用户; 客户端管理器 CMgr(Client-Manager)用来管理客户端通信, 对外提供服务接口(此处的客户端指系统用户, 不同于后文中提到的网络模型中的客户端).

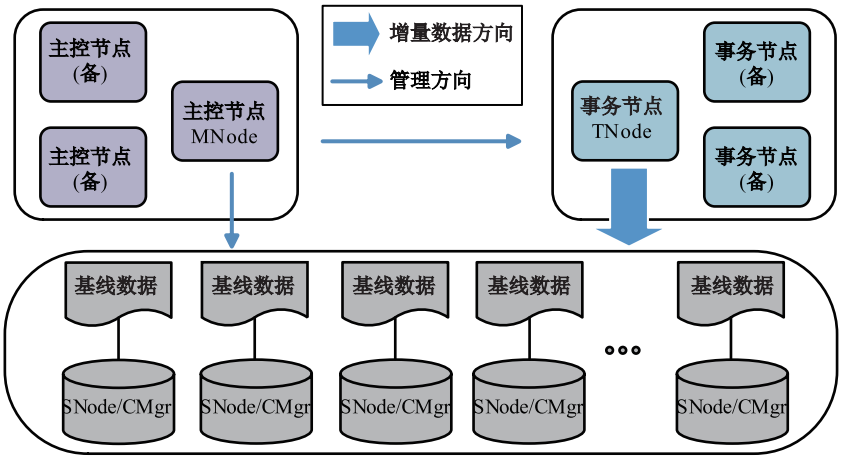

图 1 可扩展数据管理系统架构图

Fig. 1 Architecture of a scalable database management system

为了更好地理解可扩展数据管理系统中的请求服务机制, 本文基于系统架构, 抽象了系统中网络请求服务的 Client/Server 模型, 并基于该模型概括了 3 种请求传输机制和 4 种请求处理机制. 以 OceanBase 为例统计分析了 TPC-C 各类型事务负载下各机制所占的比重, 并总结了不同的请求类型对系统的不同要求.

本文的后续内容组织如下: 第 1 节介绍网络请求服务基础, 以及可扩展数据管理系统在

经典的 Client/Server 模型下包含的几种请求的传输和处理机制; 第 2 节在 TPC-C 多类型的事务负载下, 以 OceanBase 为例, 统计可扩展数据管理系统中各机制处理的请求占整个系统的比重, 分析其与系统架构的关系; 最后, 第 3 节总结全文.

1　网络请求服务基础

1.1　网络请求服务模型

最原始的网络请求服务使用的是阻塞型接口, 需要等待调用结果的返回, 这样的模型一次只能响应一个客户端请求, 无法满足多客户端的应用带来的网络要求. 对此最简单的解决方式是在服务端使用多线程, 让每一个连接都有独立的线程, 这样任何一个连接的阻塞都不会影响到其他的连接. 但是多线程模型会严重占用系统资源, 降低系统对外界的响应效率. 也就是说, 多线程模型中每个线程依然只能处理一路 I/O, 适合处理短连接, 而且是连接的打开关闭非常频繁的情形, 并不适合处理长连接. 因为过多的线程依然会使服务器的性能下降. 一般系统的底层网络服务选用 Linux select 或 epoll 等 I/O 多路复用接口[11], 使用句柄操作, 可以同时从多个客户端接收数据, 同时对多个句柄进行读、写和异常状态的检测, 保证并发. 这个环节只管收, 不处理任务, 把句柄放到一个缓冲区里面, 然后业务处理模型对接线程池, 可以使复杂业务处理上的负担被分担. 但当句柄值较大时, 接口本身需要消耗大量时间去轮询各个句柄.

如果选择异步的方式, 需要在业务处理里使用完全的异步API(Application Programming Interface), 如 OceanBase 底层使用了开源的事件驱动库 libev[12]. 很多业务要保障流程的正确是需要同步操作的. 异步回调和闭包在编程实现上带来了一定的复杂度和风险. UNIX 平台下多进程模型擅长处理并发长连接, 但不适用于连接频繁产生和关闭的情形. 最高效的模型是异步非阻塞 I/O, 而且不用 select/epoll 接口, 因为这两个接口都有着 O(n) 的时间复杂度[11]. 异步非阻塞 I/O 模型下的编程是有一定难度的, 由于 I/O 操作不再阻塞, 需要亲自管理维护每个连接的状态. 为了充分利用 CPU, 还应结合线程池, 避免在轮询线程中处理业务逻辑.

图 2　可扩展数据管理系统中的网络服务模型图

Fig. 2　Network server model of a scalable database management system

图 2 抽象了可扩展数据管理系统中的网络服务模型, 我们说当一台服务机 S1 向另外一台服务机 S2 请求服务时, S1 就被称为客户端, S2 被称为服务端. 服务端接收客户端发送的网络包, 通常会将网络包加入到任务队列中[13]. 接着, 工作线程会从任务队列中不断获取网络包进行处理, 处理完成后应答客户端. 而系统中一台服务机在不同的任务当中往往会充当不同的角色. 系统中的各个节点往往会同时封装客户端和服务端, 在不同的处理流程中, 节点会首先定位自己的角色然后调用相关的处理方法.

1.2　可扩展数据管理系统中的请求服务机制

可扩展数据管理系统中的请求传输机制主要分为两种: 一种是客户端发送请求并需要服务端回复请求, 如查询内存表; 另一种是客户端发送消息不需要服务端回复, 如异步日志回放. 如图 3, 我们将只响应不回复的服务端称为响应端. 其中, 需要回复的请求又分为同步请求和异步请求. 同步请求需等待服务端返回响应, 如获取元数据[14], 而异步请求则不需要,

如心跳. 异步请求时, 客户端将请求包加入到网络发送队列后立即返回, 不等待应答. 同步请求时, 客户端将请求包加入到网络发送队列后开始阻塞等待, 直到网络线程接收到服务端的应答后才唤醒客户端, 从而执行后续处理逻辑. 而不需要回复的请求表现为单向通信, 对方接到消息后立即执行处理逻辑, 请求的发送即为请求的结束.

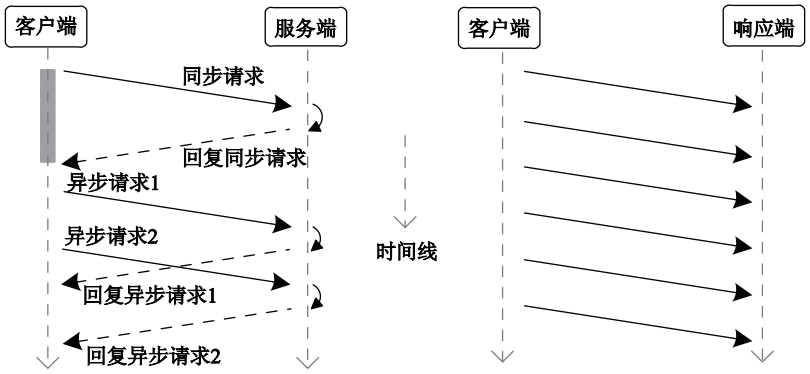

图 3 请求的传输机制

Fig. 3 Request transmitting mechanism

在生产者/消费者模型中, 往往有一个全局任务队列, 生产者将任务加入到任务队列, 消费者从任务队列中取出任务进行处理[6]. 图 4 展示了请求处理机制对于不同的任务有不同的处理[15]. 对于处理时间比较短的任务, 采用 I/O 线程直接处理的方式, 可以减少上下文切换. 基于事件循环的多 I/O 线程能够增加网络的收发能力[16]. 将处理时间比较长的任务加入队列, 工作线程不断地从任务队列取出任务进行处理. 所有的网络 I/O 线程和工作线程操作全局任务队列都需要首先获取独占锁, 这种方式的锁冲突严重, 将导致大量操作系统上下文切换. 为了解决这个问题, 给每个工作线程分配一个任务队列, 网络 I/O 线程按照一定的策略选择一个任务队列并加入任务. 选择策略如随机选择, 通过全局任务计数计算出任务所属的任务队列, 或者选择一个任务个数最少的任务队列. 如果某个任务的处理时间很长, 就有可能出现任务不均衡的情况, 即某个线程的任务队列中还有很多任务未被处理, 其他线程却处于空闲状态. 可以采取一个很简单的策略应对这种情况: 每个工作线程首先尝试从对应的任务队列中获取任务, 如果由于队列为空获取任务失败, 则遍历所有工作线程的任务队列, 直到获取任务成功或者遍历完所有的任务队列为止.

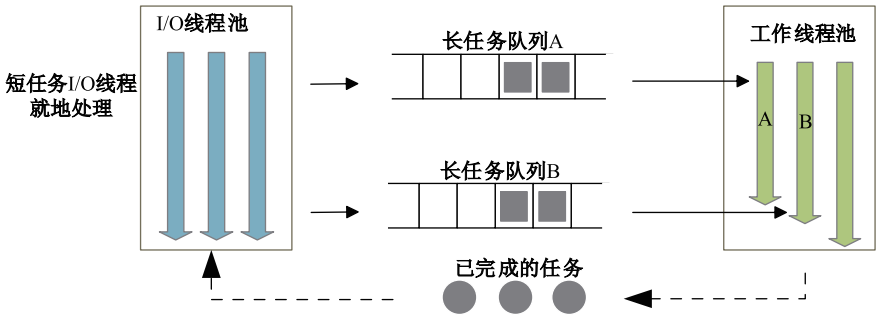

图 4 请求的处理机制

Fig. 4 Request processing mechanism

这样的任务队列在后文归类中我们称之为普通队列.

2　OceanBase的请求服务统计分析

本节以第 1 节中抽象的可扩展数据管理系统中的网络请求服务模型为基础, 在 TPC-C 不同类型的事务负载下分析请求类型和服务机制, 总结不同请求类型对应的传输机制和处理机制, 探索它们之间的关系.

2.1　传输机制与请求类型

表1列出了系统中的传输机制与请求类型的对应关系. 首先, 系统中每个节点通过心跳检测其他节点是否存活, 心跳发出后不必阻塞等待节点的回复, 心跳响应会以不需要回复的方式返回给请求节点. 如 MNode 向 TNode 发送一个心跳请求以确认该事务处理节点仍然存活, 请求发送后立即返回, TNode 随后会向 MNode 发送一个心跳响应的信号, 来告诉 MNode 自己仍然存活. 这一类用来维护节点间关系的请求, 通常是通过异步的方式来发送.

在集群启动时, 其他节点需要向主控节点注册自己的角色, 这样的注册请求需要阻塞等待回复方可进行后续的操作. 当节点需要获取描述系统或系统中用户对象的元数据时, 也需要同步等待数据的返回结果加入输出缓冲区. 如 MNode 获取 SNode 成员链表时, 主控节点的客户端管理器会向其服务端管理器发送请求, 服务端管理器接到请求后立即进行相关的处理, 将 SNode 成员链表信息加入输出缓冲区, 随后立即唤醒 I/O 线程响应客户端管理器. 又如 SNode 向 MNode 获取模式信息时, SNode 的客户端管理器向 MNode 的服务端管理器发送获取模式信息的请求, MNode 接受该类型的请求后立即处理, 将相关模式信息加入输出缓冲区后唤醒 I/O 线程响应 SNode. 同样的同步传输机制还应用于操作 SQL、获取版本、执行事务、处理触发事件等.

表 1　客户端传输机制及请求类型

Tab. 1　Transmitting mechanisms and request types in client

传输机制	请求类型
异步	心跳等
同步	节点注册, 元数据获取等
无回复	日志回放等

还有一类请求是不需要回复的, 客户端发出后即结束, 服务端接收到请求后进行响应即可. 这里我们称服务端为响应端. 通常情况下这一类的请求是不带数据的请求, 而且请求包传输到响应端后生命周期立即终止, 响应端的后续处理由此触发但并不会延续请求的生命周期. 如事务处理节点 TNode 在启动时需要异步回放提交日志, 此时 TNode 客户端管理器会向其服务端管理器发送不带输入输出缓冲区的空请求, 服务端管理器接到请求后立即回放日志, 响应过程结束. 这一个过程与异步传输机制中的服务端向客户端返回结果十分类似, 但是场景不同, 因而这里做一个区分, 并将只处理不回复的服务端称为响应端.

2.2　处理机制与请求类型

服务端接受一个请求后就要进行相应的处理或者响应, 在不同的服务端对应不同类型的请求有不同的请求处理机制. 系统常常使用队列作为客户端和服务器之间的请求和应答的缓冲区[17]. 在一个可扩展的数据管理系统中, 事务处理节点往往处理较核心的业务任务, 也承担着一定比重的负载, 因此 TNode 上的处理机制有多种, 而 SNode 不需要维护队列来管理请求, MNode 只需要简单的队列来管理请求.

对于 SNode 而言, 不管是异步的心跳请求, 同步的获取元数据的请求, 还是建表的请求都直接在 I/O 线程就地处理, 因为这些请求往往不带数据只带报头, 处理时间通常在几十微

秒的数量级, 结束后便将结果返回给客户端. 而 MNode 作为主控节点管理元数据, 会区分请求的类型用不同的队列进行管理, 如数据读取类请求归读队列管理, 数据修改类请求归写队列管理, 节点间协调关系类请求归租约队列来管理. 由于这些请求在 MNode 上的到达和处理顺序随机, 所以这里的队列类型均为普通队列. 然而对于事务处理节点 TNode 而言, 复杂的事务处理和较高的任务负载决定该节点的请求处理机制不仅仅是 I/O 线程就地处理和普通队列+工作线程的方式就能很好地应对的. OceanBase 的 TNode 为了保证事务的提交顺序, 采用 FIFO(First-In-First-Out)的队列来处理事务提交的请求, 而在事务的执行上采用优先级队列的方式, 因为普通队列和 FIFO 队列都不能满足事务并发控制可能带来的优先级要求. 而对于短任务比如异常状态下请求的直接退出、提交异步任务时的日志回访请求, TNode 都会在 I/O 线程直接处理, 不会排队. 对于节点间协调关系类的请求如心跳, TNode 会同 MNode 一样采用普通队列来构建租约队列进行管理. 我们将服务端的请求处理机制与请求类型归纳总结为表 2.

表 2 服务端处理机制及请求类型

Tab. 2 Processing mechanisms and request types in server

处理机制	请求类型
I/O线程就地处理	元数据的获取等
普通队列+工作线程	元数据的修改等
优先级队列+工作线程	事务的执行
FIFO队列+工作线程	事务的提交

可以看出, 对于请求的处理机制, 首要的区别是请求是否需要排队, 其次是对于需要排队的请求, 针对不同的请求类型和节点处理的特点, 选择不同类型的队列.

2.3 多类型事务下的请求服务统计分析

TPC-C是数据库管理系统在线事务处理(OLTP)性能测试的国际标准[18], 已得到广泛认可. 为了在一定程度上反应可扩展数据管理系统中各个请求服务机制的分布情况, 本小节统计了 TPC-C 多类型事务下各机制服务的请求占比. 并进行相关的简要分析.

图 5 所示为同步和异步两种请求传输机制的占比情况, 这里没有考虑无回复的请求, 因为大部分这一类的请求均表现为节点内部的信号, 很大程度上不构成网络 I/O 的开销. 从图 5 可以看出, 在 TPC-C 所有的 5 种事务类型中, 同步传输机制所服务的请求要比异步传输机制服务的请求量高出 1.5 倍到 4 倍. 其中, 对于 Delivery 的事务类型, 同步传输机制服务的请求比重在所有的 TPC-C 事务类型中为最高, 这跟该类型负载中大量的批处理直接相关. 而 OrderStatus

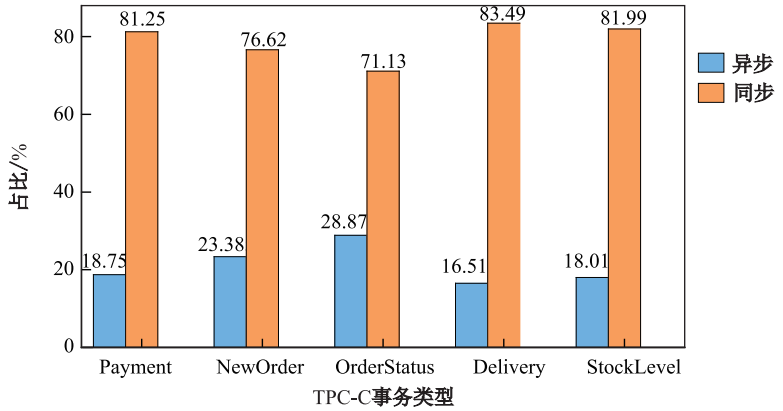

图 5 2种传输机制的比例分布情况

Fig. 5 Proportional distribution of the two transmitting mechanisms

更多的是处理用户的查询, 所以异步传输机制服务的请求量相对较高.

可以看出, 对于可扩展的数据管理系统而言, 同步机制是网络请求传输的主要方式.

由于 OceanBase 底层使用了 libev 事件驱动库, 所以它的同步处理机制是基于底层的异步 I/O 实现的, 这在一定程度上也优化了系统同步机制的发送性能.

而对于网络请求的处理机制而言, 如图 6 所示, 我们可以看到, FIFO 的队列管理方式在一个数据库管理系统中不可或缺[19], 但是在一个可扩展数据管理系统中所能处理的请求已经不再占有较大的比重. 取而代之的是优先级队列. 对于一些对执行顺序不敏感的请求, 系统还采用了随机出入队的普通队列, 进而降低一部分维护成本. 系统中还有大量的短任务可以在 I/O 线程就地处理, 主要表现在 SNode 对请求的转发和系统内各节点的心跳信号.

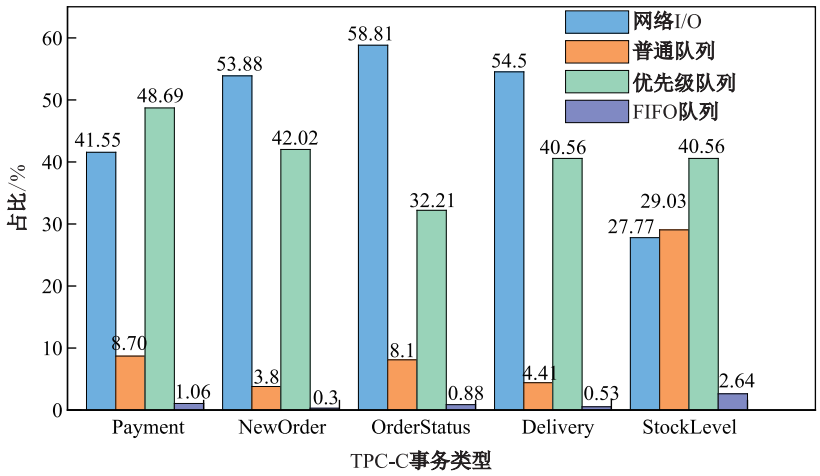

图 6　4种处理机制的比例分布情况

Fig. 6　Proportional distribution of the four processing mechanisms

如果将 TNode 节点视为一个内存数据库, 当所有的读写操作都可以在内存上直接进行时, 不阻塞的事务读取请求可以在 I/O 线程就地处理, 这将提升数据库的请求服务性能.

3　结　　论

网络请求服务机制是可扩展数据管理系统底层支持的重要部分. 本文从系统的架构出发, 抽象网络服务的 Client/Server 模型, 分别以客户端和服务端的角度总结概括了模型中网络请求的传输机制和处理机制, 并将请求传输机制分为 3 种, 将请求处理机制分为 4 种, 关联不同的请求类型进行具体介绍. 以 OceanBase 为例, 文章在 TPC-C 不同类型的事务负载下统计了各机制所处理的请求占系统所有请求的比重, 并分析了不同的请求类型对系统带来的不同要求, 及其与服务机制的关系.

可扩展数据管理系统除了实现集群规模的伸缩性, 也需要提供高性能事务. 为事务类型的请求进行恰当的服务机制的选择是事务处理性能得以保证的前提. 网络请求的发送机制主要分为同步机制和异步机制. 对于数据库而言, 大部分事务相关或者写任务相关的请求都需要采用同步机制. 网络请求处理机制的多样性主要体现在系统的事务处理节点. 传统的先入先出服务类型已经无法满足可扩展架构下的事务请求, 因此, 多种队列各自配合工作线程的服务模型被应用到可扩展数据管理系统中.

[参 考 文 献]

[1] BAKER J, BOND C, CORBETT J. C, et al. Megastore: Providing scalable, highly available storage for interactive services [C]//Proceedings of CIDR. 2011, 11: 223-234.

[2] LAKSHMAN A, MALIK P. Cassandra: A decentralized structured storage system[J]. Operating Systems Review, 2010, 44(2): 35-40.

[3] COULOURIS G, DOLLIMORE J, KINDBERG T, et al. Distributed Systems: Concept and Design[M]. Addison-Wesley, 2012.

[4] ÖZSU M T, VALDURIEZ P. Principles of Distributed Database Systems[M]. New York: Springer-Verlag, 2011.

[5] MARYANSKI F J. A survey of developments in distributed data base management systems[J]. IEEE Computer, 1978, 11(2): 28-38.

[6] 杨传辉. 大规模分布式是存储系统: 原理解析与架构实战 [M]. 北京: 机械工业出版社, 2013.

[7] Apache. Apache HBase [EB/OL]. [2016-06-10]. https://hbase.apache.org.

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

[9] 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, 6(5): 141-148.

[10] OceanBase [EB/OL]. [2016-06-10]. https://github.com/alibaba/oceanbase.

[11] STEVENS W R, FENNER B, RUDOFF A M. UNIX Network Programming, Volume 1: The Sockets Networking API [M]. Addison-Wesley Professional. 2003.

[12] Libev [EB/OL]. [2016-06-10]. http://software.schmorp.de/pkg/libev.html.

[13] GARCIS-MOLINA H, ULLMAN J D, WIDOM J. Database System Implementation [M]. New Jersey Prentice Hall, 2009.

[14] CHANG L, WANG Z, MA T, et al. A massively parallel processing SQL engine in hadoop [C]//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2014.

[15] STONEBRAKER M. Technical perspective-one size fits all: An idea whose time has come and gone[J]. Commun ACM, 2008, 51(12): 76.

[16] JOHNSON R, PANDIS I, AILAMAKI A. Eliminating unscalable communication in transaction processing[J]. VLDB Journal, 2014, 23(1): 1-23.

[17] BERNSTEIN P A, NEWCOMER E. Principles of Transaction Processing[M]. Morgan Kaufmann, 2009.

[18] Transaction Processing Performance Council (TPC). TPC Benchmark C: Standard Specification [EB/OL]. [2016-06-10]. http://www.tpc.org/tpcc/.

[19] HELLERSTEIN J M, STONEBRAKER M, HAMILTON J. Architecture of a Database System [M]. Hanover: Now Publishers Inc, 2007.

(责任编辑: 林 磊)