文章编号: 1000-5641(2018)03-0088-09

基于分布式数据库 Cedar 的高效工单管理系统设计与实现

潘宇晨[1,2], 李宇明[1,2], 张春熙[1,2], 张 蓉[1,2], 洪道诚[1,2]

(1. 华东师范大学 计算机科学与软件工程学院, 上海 200062;
2. 华东师范大学 上海高可信计算重点实验室, 上海 200062)

摘要: 随着互联网发展, 企业随之转型, 积极开展基于互联网的业务. 传统业务系统架构基于集中式数据管理系统如 MySQL 之上, 在封闭使用状态向开放使用状态转换的过程中, 逐渐在可用性上暴露出弊端, 故而不能很好地支持业务拓展规模化、分布式处理的要求. 网络业务的开展对工单系统提出了在支持大数据、高并发、高冲突、高可用下保证处理高效性的新需求. 在深入分析当前业务特征的基础上, 利用分布式数据库 Cedar, 基于 Netty 通信框架, 以海尔的工单业务为实例, 设计并实现了支持工单存储、派单以及工程师抢单业务的高效、可扩展工单管理系统, 详述了系统的可靠性和可扩展实现. 经实验表明, 该工单系统在保证高吞吐和低延迟的同时具有良好的可扩展性和可用性.

关键词: Cedar; 分布式计算; 工单管理; share-nothing 架构
中图分类号: TP391 **文献标志码**: A **DOI:** 10.3969/j.issn.1000-5641.2018.03.010

The design and implementation of an efficient order management system based on Cedar

PAN Yu-chen[1,2], LI Yu-ming[1,2], ZHANG Chun-xi[1,2],
ZHANG Rong[1,2], HONG Dao-cheng[1,2]

(1. *School of Computer Science and Software Engineering, East China Normal University, Shanghai* 200062, *China*;
(2. *Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai* 200062, *China*)

Abstract: With the development of Internet, enterprises are increasingly relying on the Internet for core functionality of their business systems. Legacy business systems which were based on centralized data management platforms, such as MySQL, have shortcomings in usability when the systems are used on the Internet instead of for internal business processes. They are not suitable for supporting new business requirements which require scalability with concurrent transactions and conflict resolution. In this paper,

收稿日期: 2017-06-19
基金项目: 华东师范大学信息化软件学研究课题 (41600-10201-562940/018)
第一作者: 潘宇晨, 女, 硕士研究生, 研究方向为数据库. E-mail: 295796676@qq.com.
通信作者: 张 蓉, 女, 教授, 研究方向为数据库. E-mail: rzhang@sei.echu.edu.cn.

we design and implement a scalable order management system which supports order storage, assigning orders, and order rushes using the distributed Cedar database and Netty communication framework. The characteristics are designed on the basis of an in-depth analysis of current business characteristics. We verify our design on real workload of a Haier system. The experimental results show that the proposed order management system has good scalability, high throughput, and low latency.

Keywords: Cedar; distributed computing; order management; share-nothing architecture

0　引　　言

随着互联网的飞速发展, 电商随之新起, 对传统企业造成巨大的竞争压力. 部分传统企业因此转型, 在坚守线下发展的同时, 开始开拓电子商务业务. 传统的电器销售企业, 例如海尔、九阳、美的、海信等纷纷在天猫、京东、苏宁等电商平台上开展电子商务 (电商). 原有的工单系统大多根据电话中心或现有门店接受用户订单, 数据管理系统由内部员工手动操作, 是企业内部的封闭服务系统. 电商业务的开展使得绝大部分的订单由电商渠道产生, 系统的使用从封闭状态转向服务于大众的开放式平台, 这对原有的数据管理系统提出了极大挑战. 比如, 商家为了促销商品会定期举办大型的降价促销活动, 在促销活动当天会产生大量的新增订单; 订单入库之后, 会进行工程师派单甚至抢单业务. 因此, 原有的工单管理系统在面对大规模用户并发下单、买单或消单以及安装维修等派单业务的需求时可用性受到挑战; 其中最具挑战性的任务是支持工程师抢单, 解决短时间内数据库冲突操作. 本文针对这类业务新需求, 设计与实现了基于分布式数据库 Cedar 的高效工单管理平台.

工单业务管理流程如图 1 所示, 现有的业务需求主要分为 3 个场景: 工单入库、派单和抢单.

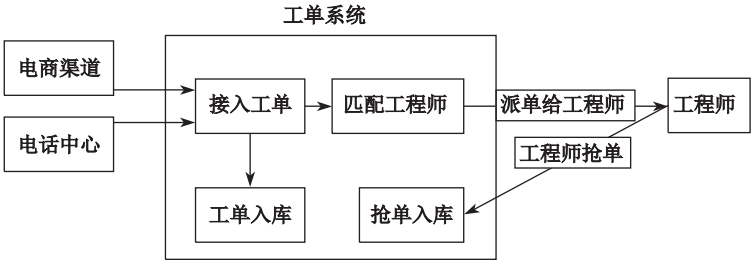

图 1　工单系统业务流程图

Fig. 1　Business requirements for work order system

工单入库指将新增工单信息写入数据库 (数据库存储). 入库工单来自于两个渠道, 电商平台和电话中心. 电商平台主要包括天猫、京东、苏宁、国美、门店和微信等渠道, 根据这些电商渠道的订单生成工单并接入目标系统. 电话中心根据接到的服务请求人工生成工单并接入目标系统. 负载类型是写请求, 将工单信息写入工单表. 通常单日的写入量为几百条, 在“双十一”等促销活动当天单日写入量为百万级别.

数据库管理系统存储订单之后, 需要根据系统指定的应用规则对工程师进行派单. 将所在地区、工程师技能与工单所需、工单时间内工程师是否非忙等信息与工程师进行匹配, 匹配后将工单派送给多位工程师.

工程师在接到派单后, 可以选择是否接收该工单. 存在多个工程师抢同一个工单的情况, 抢单存在失败. 若工程师抢单成功, 需要将该信息写入数据库.

满足以上业务需求的工单管理系统实现存在以下3个难点.

- 工单量大

工单接入服务需要支持高并发. 像在“双十一”这类促销活动当天, 单日工单接入量大, 与平日的工单入库量相比存在量级上的变化.

- 派单量大

面对大量新增工单, 系统根据规则进行派单, 如根据工程师是否在岗、工单请求时间该工程师是否空闲、所在区域、具备技能是否满足等信息将工单派送给指定值的工程师. 高峰约有10万个服务工单请求需要进行派单, 而每个工单为了保证被接单, 派单量通常是工单量的几倍 (如10倍).

- 抢单冲突大

多位工程师在接到派单后, 多人将抢同一个订单. 系统需要支持工程师并发抢单. 因为工单量大, 并发冲突量大.

针对以上难点问题, 新型的工单管理系统要求系统架构具有可扩展性、事务冲突解决具有高效性, 数据库写入满足完备性, 同时系统具有高的可用性和可靠性. 本文工单管理系统实现时采用分布式架构, 处理节点可扩展保障系统可扩展, 处理节点宕机后重启即可照常工作, 路由节点设置一主一备来保障系统可用性和可靠性, 在写入数据库前处理事务冲突, 采用反馈机制保障数据库写入完备性, 详见第2节.

1 相关工作

目前, 国内的如电信企业、电器销售企业等具有安装配置、故障保修等售后服务的企业, 为了满足其迅速发展的业务需求大多开发了自己的工单系统. 如鞍山网通的故障工单系统[1], 主要针对故障处理, 基于B/S结构, 使用Visual Basic Script+ASP技术在ORACLE数据库上开发的; 该系统具有障碍工单管理、查询、派单、回单等功能. 另一个具有代表性的系统是江西电信的综合电子故障系统[2], 它使用B/S结构基于J2EE架构开发, 其表现层使用JSP页面开发交互界面, 持久层依赖于Hibernate框架, 数据库层使用SQL SEVER; 该系统中存在经验库模块, 可以快速查阅相似故障的维修方法, 具有很好的使用价值. 江西联通大工单系统[3]主要是为联通公司各个部门跨部门、跨专业的协同办公, 以实现电子工单的流转、全程跟踪和统计查询功能; 该系统采用B/S结构, 采用J2EE架构, 使用ORACLE数据库. 电信ADSL工单系统[4]是四川省电信公司支持开发的, 系统在接收上层营帐系统的派单后分析处理该订单后回单给营帐系统; 系统采用C/S与B/S相结合的结构模式, 在管理查询部分采用B/S模式, 在工单分析、处理部分采用C/S模式; 系统是在.NET平台上开发, 数据库使用SQL SEVER.

总地来说, 现有的工单系统大多采用B/S结构, 基于.NET、J2EE或ASP等架构开发. 现阶段在业务需求上主要存在以下可扩展性、可用性问题: ①不支持高并发. 在“双十一”、“黑色星期五”这种打折促销活动当日会产生大量的工单, 如何以低成本处理并存储这类高并发订单接入是一个难题. 现有的如基于MySQL工单系统并未考虑到该情况, 在高并发下系统性能降低甚至死机. ②无法有效解决高冲突. 多名工程师抢同一工单时会产生冲突, 在各促销活动当日会产生高冲突. 现有的工单系统无法有效支持高并发情况下有效地冲突解决.

2　可扩展工单管理系统设计与实现

2.1　可扩展工单管理系统架构

如图 2 所示, 该工单系统主要分为 3 层: 应用层 (Application Layer)、数据库层 (Datebase Layer) 和测试层 (Test Layer). 测试层用于模拟真实场景下工程师接到派单并抢单的情况, 仅仅用于实验测试. 该工单系统基于分布式架构, 具有可扩展和高可用的优势.

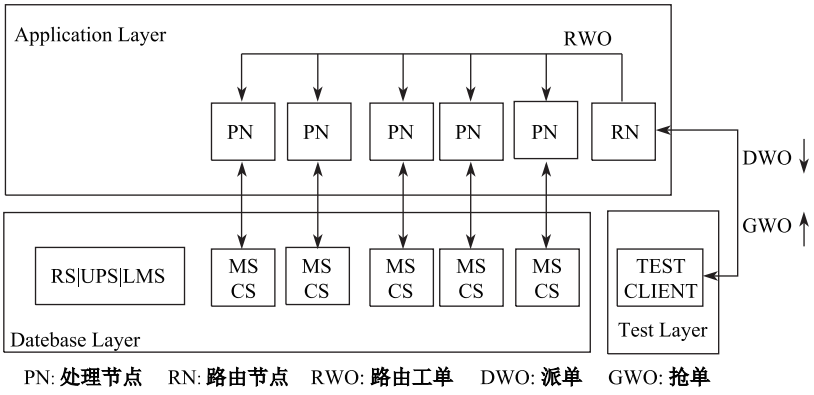

图 2　工单管理系统架构图

Fig. 2　Architecture diagram for work order system

数据库层存储接入工单信息、工单处理状态如抢单结果和工程师信息.

应用层中的节点分为两类, 即路由节点和处理节点. 所有的工单由路由节点接入, 按规则将工单路由到各个处理节点, 由处理节点将工单写入数据库. 处理节点将工单与工程师匹配, 分发给规定名额内的工程师.

测试层用于模拟工程师收到派单后抢单, 并将抢单信息传递给处理节点, 由处理节点将抢单结果写入数据库.

2.1.1　数据库层

数据库层使用的是 Cedar 数据库. Cedar 是基于 OceanBase[5] 的可扩展关系数据库. 如图 3 所示, 数据库层使用 Cedar 单集群+多 CS, 集群中有 1 台主控服务器 (RS) 管理集群中的所有服务器, 1 台更新服务器 (UPS) 存储增量更新数据, 1 台监听合并服务器 (LMS) 查询主备集群的流量分布信息和所有的其他合并服务器的地址列表, 5 台基准数据服务器 (CS) 存储基准数据和 5 台合并服务器 (MS) 接收并解析用户的 SQL 请求转换后转发给相应的 CS 或 UPS.

Cedar 的架构将基线数据和增量数据分开存储. 增量数据主要指最近一段时间的操作并修订的数据, 存储在内存中, 定期与基线数据合并, 然后在合并数据分发存储到基线存储服务器中, 成为新的基线数据. 这使 Cedar 具有良好扩展性, 支持海量数据存储和高效处理. 而且读写操作大多落在增量数据上, 因此以内存计算为主, 硬盘访问次数较少, 能有效保证甚至提高数据库访问、处理性能. Cedar 采用单台更新服务器, 因此写事务都将集中在单台更新服务器上, 避免了复杂的分布式事务, 高效地实现了跨行跨表事务. 另外, 更新服务器上的修改增量能够定期分发到多台基准数据服务器中, 避免成为瓶颈, 实现了良好的扩展性.

Cedar 支持多集群, 每个集群部署一个包含 RS、UPS、CS 以及 MS, 通过强同步的方式保障了高可性. Cedar 支持二集群、三集群、四集群及更高的集群. Cedar 多集群间需要通讯、同步等操作会降低系统性能.

2.1.2 应用层

应用层从各个电商渠道或电话中心接入的工单都由路由节点 (RN) 根据工单区域信息路由到处理节点 (PN) 上. 如图 3 所示, 路由节点根据工单区域信息将工单分类 (如 Hash), 每个处理节点处理对应的区域类. 处理节点在接到工单后, 将该工单写入数据库, 将入库成功的工单信息缓存在内存中, 处理节点把数据库中在岗工程师根据区域信息分类并将同一区域内的工程师按评分排序后缓存在 PN 节点. 该分类方式与工单路由中分类方式相同. 这样使得工单与可以处理该工单的工程师都缓存在同一个处理节点中. 处理节点依次判断工程师在工单请求时间内是否非忙、产品技能是否满足要求, 如果两者都满足则生成派单, 直至生成派单数到设置值. 由于派单操作不对数据库进行任何操作, 可记流水. 工程师的信息是周期性地从数据库中读取缓存在处理节点的内存中. 工程师抢单后将抢单结果传递给处理节点, 处理节点将抢单结果写入数据库. 并且在内存中更新工程师的工作状态 (如某时间段忙碌), 异步地将工程师状态信息在数据库中更新.

图 3 工单处理流程图

Fig. 3 Work order process flow

为了保障系统的高可用性, 路由节点设置为一主一备, 通过心跳[6]保证同步. 主节点路由工单到处理节点上, 并将处理过的工单异步传递给备节点. 如果主节点宕机, 备节点在一段时间内接收不到主节点的心跳, 备节点成为新的主节点. 新的主节点上有旧版本的工单 ID, 路由该 ID 对应的工单 (工单 ID 依次增加) 到处理节点. 可能会有部分工单被多次分发到处理节点, 通过入库主键唯一保障工单入库逻辑上的正确性. 如果该工单已被写入数据库则会由于主键重复写入失败. 最终结果仍是工单一次入库. 而处理节点宕机可恢复. 如果某一处理节点宕机, 只要重启该节点, 并从数据库中读取该区域未派单的工单和该区域的工程师后, 处理节点照常工作, 保障系统可靠性. 并且处理节点可扩展, 可以配置多台处理节点来提升系统性能, 实现系统可扩展.

在实现中, 采用 share-nothing 架构[7]. 各个处理节点都有自己私有的 CPU/内存/硬盘等, 不存在共享资源, 各处理节点之间通过协议通信, 具有良好并行处理和可扩展能力[8].

2.1.3 测试层

测试客户端主要是在接收到处理节点的派单后模拟工程师抢单任务. 通过调整工程师的抢单概率模拟不同的抢单情况, 查看不同冲突规模下系统的性能. 为了测试系统在高并

发、高冲突下的性能, 可以配置多台 Test Client. 各台 Test Client 互不影响, 不共享资源.

2.2　系统实现

如图 4 所示, 工单系统的主要实现流程分为工单接入、派单和抢单两个流程. 接下来将详细介绍其主要流程.

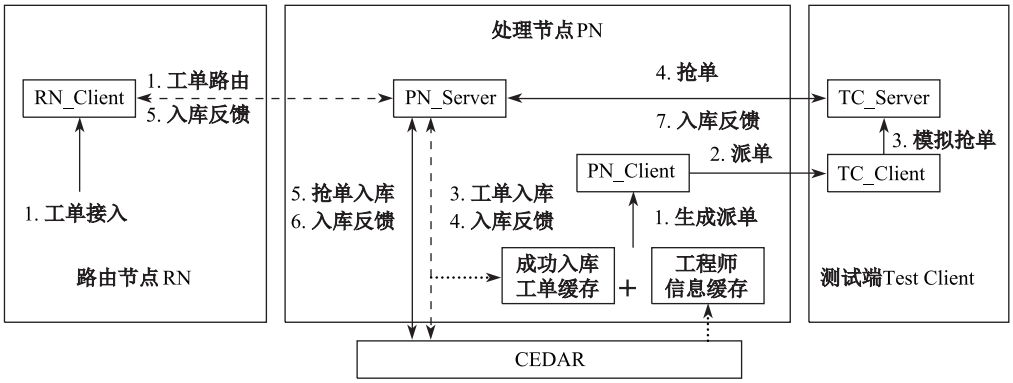

图 4　工单处理系统流程图

Fig. 4　Process flow diagram for order system

2.2.1　工单接入流程

从各个渠道产生的工单从统一入口接入路由节点, 通过路由节点 RN 分类路由到各个处理节点即 PN_Server. PN_Server 将工单写入数据库, 数据库将入库状态信息反馈回 PN_Server, 由 PN_Server 反馈给路由节点. 若接收到入库失败反馈, 或在一段时间内未接到入库反馈, 路由节点将重新路由该工单到处理节点, 保障每一条工单入库成功, 保障数据库写入完备性.

工单的分发处理速度由于网络交互的原因远远落后于工单接入的速度, 因此本系统为了提高处理性能采用双缓冲区的实现方法: 一个缓冲区采用单线程接入工单, 而另一缓冲区完成多线程分发工作. 这种方法既不会堵塞缓冲区处理, 也可以有效地匹配入与出对象的速度.

2.2.2　派单和抢单流程

处理节点将该节点成功入库的工单缓存在内存中, 根据工单 ID 排序. 工程师信息与处理节点中的工单基于服务区域被统一 Hash 函数进行分类并缓存在处理节点, 将工程师信息根据服务能力评价排序. 任意一个工单可以根据指定的规则匹配多位工程师, 然后发送派单给工程师, 即 PN_Client 根据生成的派单将工单派送给 TC_Server.

TC_Server 在接收到派单后, 如有 m 个工程师接到派单, 其中有 n 位工程师会抢单 ($n < m$). TC_Server 将抢单信息发送给 PN_Server, PN_Server 将抢单状态入库. 抢单入库操作具有数据库写冲突 (多人抢同一个工单), 这类冲突会影响数据库性能. 本系统将同一工单的抢单结果路由到同一个数据库操作线程上, 数据库操作线程维护了一个已被抢单的工单 ID 集合, 若抢单事务以及存在于该集合, 则现抢单失败, 否则进行数据库入库操作, 操作成功, 把抢单结果写入集合, 并对抢单线程进行回复. 已抢工单集合采用队列实现, 若队列满, 从一端删除最早入库工单. PN_Server 将入库反馈传递回 TC_Server, 同时该入库反馈若丢失不会对系统造成影响. 该方法保障系统高效解决事务冲突.

2.2.3 强制派单流程

工单系统存在两种强制派单的情况: ①挑单, 推送派单成功, 但是一段时间(如30 min)后无人抢单; ②未派单, 没有满足要求的工程师, 导致该单未派出去.

目前对上上述两类工单的处理方法是, 直接交于人工强制派单, 强制派单事务与抢单事务完全相同.

3 实 验

3.1 数据库schema定义

数据库中表结构如图5所示. WORKORDER表记录工单信息, SERVICEORDER表记录成功的抢单信息, ENGINEER记录工程师信息.

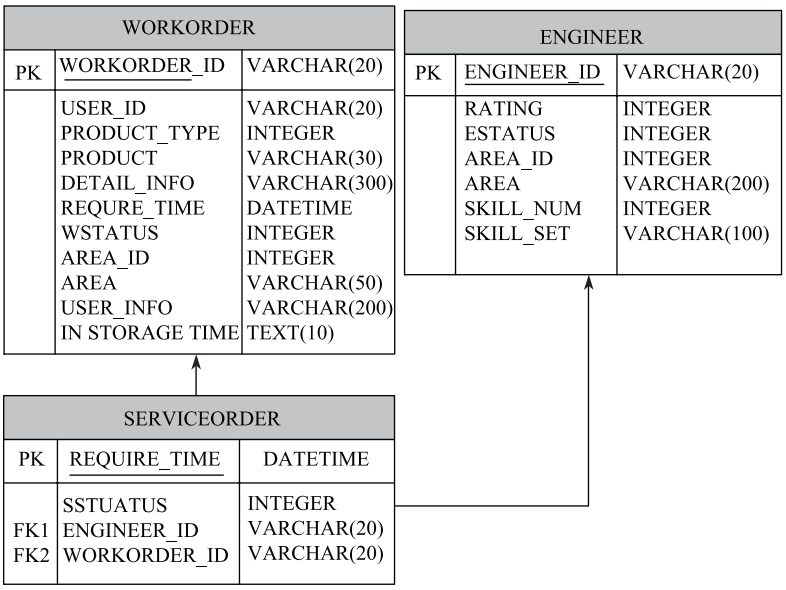

图5 数据库schema

Fig. 5 Database schema

实验环境: 本文中通过一系列对比实验对实现的工单系统进行了性能评估. 实验部署在多台配置相同的服务器. 每台服务器配置有2块Intel(R) Xeon(R) CPU E5-2620, 共12核, 24个线程, 16GB PC3-10600R的内存, 并配有RAID5, 且磁盘阵列含有带电缓存, 其通信网络为千兆以太网. 处理节点、路由节点和测试节点分别部署在不同的服务器上. 实验中模拟了一天的工单量. 实验采用的数据集中工单记录为200万条, 工程师记录为2万条. 在实验一中设置的模拟抢单率为50%, 每单派个10位工程师, 其中5位抢单, 用于模拟在该数据库冲突情况下测试处理节点对性能的影响. 在实验二中设置处理节点数为5个节点, 测试冲突对性能的影响.

度量标准如下.

每秒事务量(TPS-Transaction per Second): 每秒执行的事务数.

查询处理延时(Latency): 查询时延.

每秒派单量: 处理节点每秒向测试端发送派单数.

每秒抢单量: 测试端每秒向处理节点发送抢单数.

3.2　实验结果

在实验中, 数据库分别采用 Cedar 单集群、Cedar 三集群和 MySQL 数据库 (5.6.28). 实验一通过调整处理节点的个数测试系统性能. 实验结果如图 6 所示.

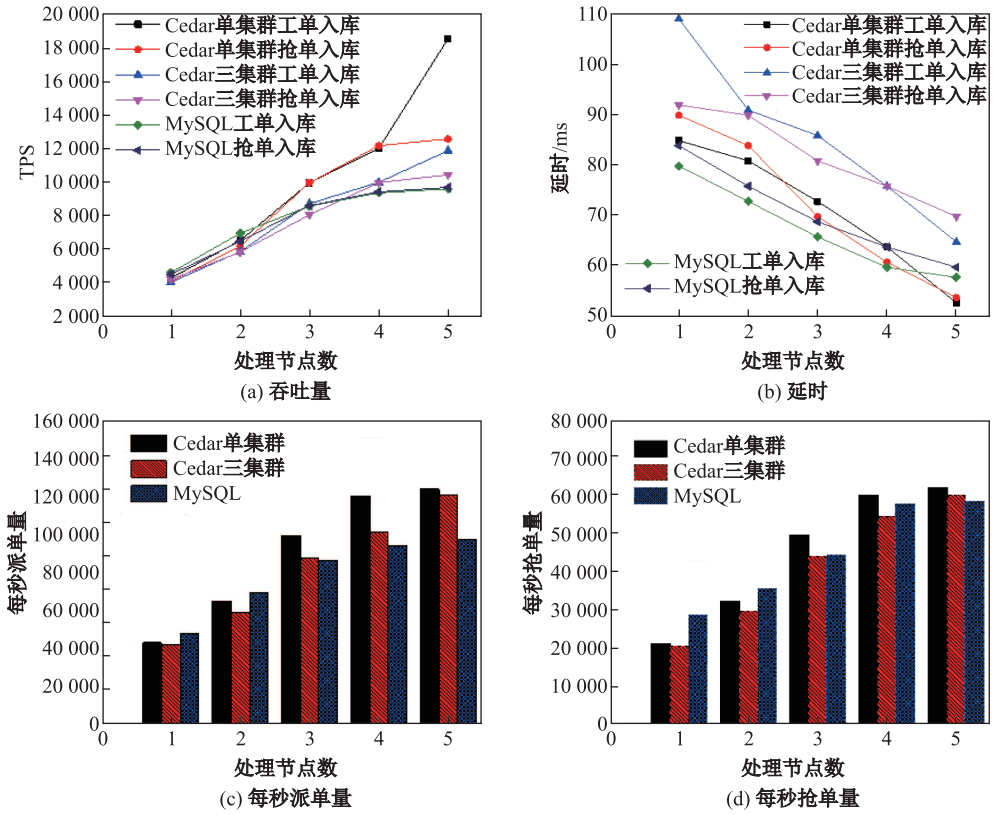

图 6　系统性能

Fig. 6　System performance summary

从图 6(a) 中可以看出, 随着处理节点的增加, 入库的 TPS 量随之急剧增加, 入库的延迟缓慢减少. 处理节点增加, 系统性能提升, 每秒派单量增加, 随之每秒抢单量也增加. 本系统支持的 TPS 远远大于业务需求, 可支持超过 12 000 的 TPS. 并且系统的时延也能较好的满足业务需求, 最大不超过 110 ms. 从图 6(a) 中可以看出, 随着处理节点的增加, Cedar 的 TPS 一直持续上升, 而 MySQL 的 TPS 在节点个数为 4 和 5 差距不大. 这是因为 Cedar 是分布式可扩展的数据库, 在系统增加到 5 个处理节点时, Cedar 的处理能力仍未达到上限. Cedar 单集群在 4 个节点到 5 个节点的过程中性能提升大是因为测试数据是根据真实应用生成的, 当节点数为 4 时, 系统分发规则导致负载均衡不好, 而节点数为 5 时, 负载均衡较好. 系统性能受限于数据库, 从实验中可以推测出 Cedar 随着节点数的增加系统性能会随之增加, 在到达一定节点数后达到性能上限. 而 MySQL 在 4 个处理节点时处理能力已经达到上限. 这是因为 MySQL 不是分布式数据库, 在 4 个节点时已经达到了它的最佳连接数. 再增加 PN 节点增加数据库连接数对系统性能不会有较大提升. 图 6(b) 中展现了 Cedar 单集群处理的性能随着节点数的增加, 延时迅速下降, 并且很快超过 MySQL 的性能; Cedar 多集群由于需要有额外的集群同步的操作, 造成总体延时偏高, 但是也是随着节点数的增加而降低. 在

图 6(c) 和 (d) 中展示了系统每秒的派单量和抢单量. 派单和抢单是系统的操作, 与数据库无关. 随着处理节点的增加, 系统每秒派单量和抢单量也随之增加.

图 7 通过调整抢单率模拟不同冲突规模对系统性能的影响. 抢单率越高系统的冲突率越大, 同一时刻的事务冲突越多. 从图 7(a) 可以看出随着抢单率上升, 抢单入库 TPS 缓慢下降. 图 7(b) 中可以看出随着抢单率上升, 延时缓慢上升, 但延时最大不超过 90 ms. 这是因为系统在应用层良好地处理了抢单冲突, 无论抢单率多大, 落在数据库底层的抢单入库请求不会发生冲突. 总体来说, 单集群吞吐量远远高于 MySQL, 三集群存在数据同步的问题, 但是性能与 MySQL 不相上下.

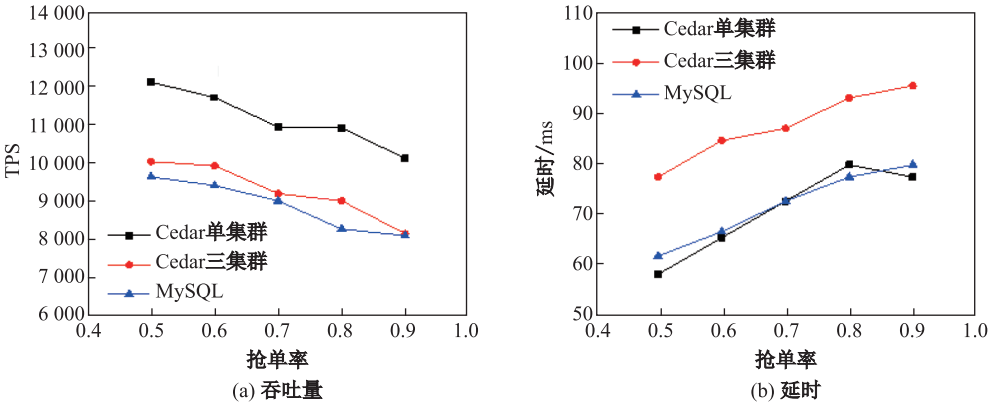

图 7 冲突率与系统性能

Fig. 7 Conflict resolution vs. system performance

为了验证数据库写入的完备性, 对比数据库中 WORKORDER 表与 SERVICEORDER 表数据. 经验证 WORKORDER 条数=SERVICEORDER 条数.

4 总 结

随着电商快速发展, 企业转型, 现有的工单系统已经不能满足日渐增长的业务需求. 本文为满足企业需求, 利用分布式数据库 Cedar, 基于 Netty 通信框架, 设计实现了易扩展、高可用、支持高 TPS、低时延的工单处理系统. 但是负载是否均衡会直接影响系统的整体性能. 在未来的工作中, 我们会通过设计良好的 Hash 函数或人工指定路由规则使得负载均衡分布, 以充分发挥多处理节点的优势.

[参 考 文 献]

[1] 滕瑞远. 鞍山网通障碍工单管理系统 [D]. 长春: 吉林大学, 2006.
[2] 周小飞. 江西电信自动电子工单系统研究 [D]. 南京: 南京邮电大学, 2007.
[3] 齐卉芳. 江西联通大工单系统的研究和实践 [D]. 北京: 北京邮电大学, 2012.
[4] 阮晓凌. 基于. NET 平台的电信 ADSL 工单系统的设计与实现 [D]. 成都: 四川大学, 2006.
[5] 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+163.
[6] 李林锋. Netty 权威指南 [M]. 北京: 电子工业出版社, 2015.
[7] VALDURIEZ P. Shared-Nothing Architecture [M]. New York: Springer, 2009.
[8] Shared Everything 和 Share-nothing 区别 [CP/OL]. (2013-08-29)[2017-05-19]. http://blog.csdn.net/seteor/article/details/10532085.

(责任编辑: 李 艺)