分布式架构在银行核心业务系统的应用①

金磐石

(中国建设银行股份有限公司 信息技术管理部, 北京 100025)

摘 要: 相对于主机集中式架构, 以 X86 和云计算为基础的分布式架构在扩展性、低成本方面的优势明显, 随着技术的进步, 其可用性也在逐步提升, 已经成为主流的架构方案. 针对在数据库层应用分布式架构, 本文从分库分表、读写分离、数据共享和访问性能优化、高效运维等方面, 提出了一套适用于银行核心业务系统的分布式架构解决方案, 并已取得应用实践的成功.

关键词: 分布式架构; 分库分表; 读写分离

Application of Distributed Architecture in Commercial Bank Core Business Information System

JIN Pan-Shi

(China Construction Bank, Beijing 100025, China)

Abstract: Compared to the host centralized architecture, the distributed architecture based on X86 and cloud computing, has become the mainstream architecture solution, with its characteristics of high extendibility, low coast and high availability. Based on the distributed architecture dedicated to database layer, this article puts forward the solution suitable to the bank core business system, such as database shading, read/write splitting, data sharing, address optimization, high efficient operation and maintenance, etc., which has achieved success in practical application.

Key words: distributed architecture; database shading; read/write splitting

1 分布式系统概念

银行业经过多年的发展, 都已经实现数据的集中, 而且, 随着客户服务的不断提升, 其核心业务系统的账户量和交易量都已经达到了较大规模, 对系统的可用性、可靠性、数据一致性、业务连续性都提出了非常高的要求[1]. 这个过程中, 集中式的架构发挥了重要作用, 各家银行的核心系统基本都构建于集中式架构之上, 尤其以大型主机架构为代表. 集中式架构下, 一般采用纵向扩展的方式, 通过增加单机的资源配置, 或者设备的更新换代, 来提升系统的处理能力. 通过硬件设备和基础软件的集群机制来提升系统的可用性. 随着技术的进步, 以 X86 和云计算为基础的分布式架构逐渐成为技术发展的方向. 分布式架构下, 强调应用的弹性, 一般采用横向扩展的方式, 通过增加服务器的数量, 提升系统的处理能力, 每个节点都是一个可独立运行的单元, 失效时也不会影响应用整体的可用性[3]. 另外, 分布式架构下, 应用系统分散到多个节点运行, 降低了对单节点的处理能力要求, 给使用 X86 服务器替代高性能的主机和小型机服务器创造了条件, 大大降低了基础设施的投入成本, 提升了安全和自主可控的水平.

2 困难和挑战

长期以来, 银行 IT 系统一直是安全稳定、一致性好的典范. 但随着互联网+和大数据渗透到生活的各个领域, 客户的交易行为发生了改变, 核心业务系统的规模急剧扩大, 运行风险增加, 处理能力的瓶颈也逐渐凸显[2] . 将分布式架构应用到银行核心业务系统中, 所面临的困难和挑战主要体现在以下几个方面.

2.1 并发处理能力

互联网时代下的业务模式已经远远超出了传统业

① 收稿时间:2016-10-08;收到修改稿时间:2016-12-05 [doi:10.15888/j.cnki.csa.005852]

务增长的概念, 秒杀、双十一、红包等业务场景, 对系统的处理能力提出了更高的要求. 采用分布式架构的以银行核心业务系统一般采用两层结构, 即应用服务器层和数据库服务器层, 按照 SOA 的设计原则, 部署在应用服务器层业务逻辑都已经实现了服务化和无状态的处理, 具备了横向扩展的能力. 目前处理能力的瓶颈主要集中在数据库服务器上, 这是分布式架构在银行核心业务系统应用的最大挑战.

2.2 可用性水平

安全稳定运行一直是银行业务系统最为看重的指标, 不管是外部监管的要求, 还是自身对客户服务的承诺, 都必须保证系统的可用性. 目前, 集中式的数据库已经成为影响系统可用性的最大风险, 在系统维护和应用版本升级时, 无法做到 7*24 小时不间断的对外服务, 影响客户体验, 也无法应对未来全球化业务拓展的要求.

2.3 交易一致性

一致性一直是银行核心业务系统赖以生成的基础, 也是长期以来一直影响分布式架构在银行核心业务应用的最大难题. 分布式架构下讨论一致性问题, 必然要提到 CAP 理论: 一个系统不能同时满足一致性(Consistency), 可用性(Availability)和分区容忍性(Partition Tolerance)这三个需求. 虽然这个理论后来得到了实验证明, 但并不代表银行核心业务系统无法应用分布式架构. 如何在保证处理能力和可用性的前提下, 提升交易的一致性水平, 分布式架构必须给出解决方案.

2.4 运维复杂度

分布式架构下, 云计算和虚拟化技术的应用, 使得系统的服务器数量急剧膨胀, 关联关系也更为复杂, 给运维工作带来的巨大挑战[8,9]. 如何在资源供给、应用部署、集中监控、故障诊断、应急处置等方面提升自动化水平, 实现高效的运维管理, 也是分布式架构推广必须要解决的问题.

3 解决方案

分布式架构的核心在于“分”, 应用层的“分”通过服务化和无状态解决, 数据库的“分”可以通过分库分表和读写分离解决, 这也是分布式技术最为典型的能力要求. 难点在于“既要能分, 也要能合”, “分”是指能够通过分布将应用系统变小, 从而使其具备横向扩展的能力, 降低运行风险. “合”更多强调的是对应用开发的透明化处理, 要支持跨库的访问.

3.1 分库分表策略

常见的分库分表策略包括垂直切分和水平切分. 垂直切分是指从业务分析入手, 将系统按照不同的业务功能, 划分为一些相对独立的子系统或模块, 这些模块内的数据表都以一定的关系形成一个整体, 并与其它模块间相对独立. 水平切分是指按照一定的逻辑, 将数据表按照分区键值进行切分, 让大表变成小表, 实现数据的分布式存储.

从实际业务场景来看, 多数场景下都需要将垂直切分和水平切分联合使用, 先对数据进行垂直切分, 再针对场景下数据访问特征选择性地进行水平切分. 从而将整个数据库切分成一个分布式矩阵.

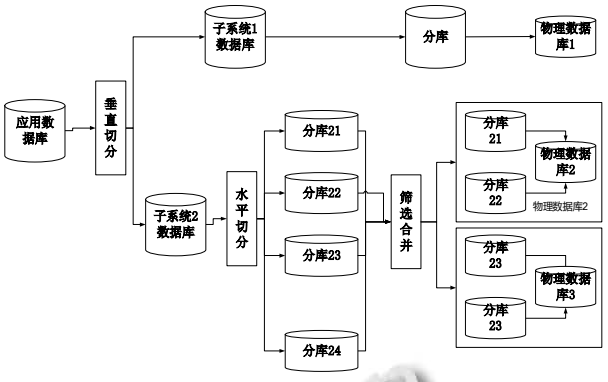

图 1 分库分表概念模型[10]

垂直切分更偏重于系统内部结构的划分, 通过应用功能的隔离, 形成一组可独立提供服务的子系统或模块, 随着业务功能的切分, 数据也就自然实现了分布. 水平切分则更关注应用模块内部的数据访问特征, 需要在尽量保证对应用透明的情况下, 实现数据的分片, 提高横向扩展能力, 以提升处理性能.

3.2 数据分布算法设计

根据 CAP 理论, 在数据库分库分表后, 做到了分区的容忍性和可用性的提升, 这必然带来一致性管理的复杂度. 目前在数据库层还没有既能保证性能, 又能够保证跨库操作强一致性的技术, 必须要从应用设计入手, 降低产生一致性问题的风险. 解决的思路就是让事务尽量发生在单一数据库内, 这样就可以利用数据库自身的事务机制保证业务操作的一致.

可以使用SBF数据分布模型来解决分库分表的问题(即 ShardingKey 分区键-DataBucket 数据桶-TableFamily

表族). 其核心思想是应用数据逻辑态与物理态的隔离, 从请求数据识别出分区键, 然后应用算法确定数据桶的编号, 相同编号的数据桶形成一个表族. 最后将数据桶映射为物理数据库的表, 然后根据数据库的数量, 将所有的表族均匀分布到各物理数据库中.

很多应用在进行分库分表时, 一般都采用简单的取模算法(a=b(mod p)), 其中 p 代表目标要分布到的物理节点, 如果物理的节点增加或减少, 所有计算出的 hash 值将发生变化, 这将引起数据分布位置的改变, 全量数据都需要重新分布, 迁移的代价较大. 分布式系统中广泛应用一致性哈希算法来解决这个问题, 其核心是将要分布的数据经过计算映射到一个具有 0～(232-1)的数据空间中, 形成一个闭合的环形. 然后将物理节点也通过哈希算法映射到这个环上, 然后以顺时针方向将所有的对象存储到离自己最近的物理节点上. 解决了单调性、分散性和负载的问题, 同时引入虚拟节点的概念, 来提升数据分布的平衡性.

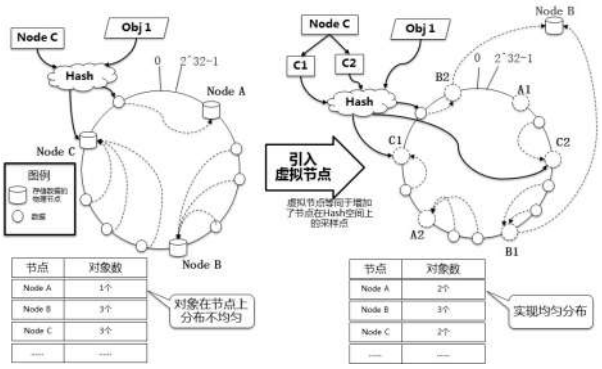

图 2　数据分布算法设计[11]

将该算法应用于核心业务系统的数据库分库分表场景下, 最重要的就是让应用的数据访问模型与算法特征、数据库特征和数据库物理部署相结合, 充分发挥一致性哈希算法的优势. 具体做法如下:

(1) 将应用的分区键作为哈希计算的输入

分区键是从应用维度分析确定的分库分表核心数据要素, 有了分区键, 就可以将一个完整的业务系统, 以水平的方式切分为多个独立的部分, 使其具备横向扩展的能力.

(2) 将数据桶作为虚拟节点

数据桶是一种对应用屏蔽后台数据库的物理部署形态的数据划分方式. 在应用系统的开发中, 银行信息系统最关心的是业务数据的流转和操作, 如果直接将分区键映射到物理数据库, 将使应用逻辑的处理与物理部署耦合, 给数据库的横向扩展带来较大的复杂度. 将数据桶映射为虚拟节点, 适应了一致性哈希算法的要求, 使得数据分布更加均衡、稳定. 数据桶数量的确定应该根据实际业务数据量、跨分区键操作场景、单一数据库容量来综合考虑, 并不一定是越多越好. 一般取值是 2n.

(3) 数据桶作为目标数据表

数据桶是一个逻辑概念, 代表的是业务需要操作的目标数据模型, 应用开发最终一定是通过 SQL 语句来操作, 最终在物理态上, 一定是映射为数据库表, 而不是数据库. 在应用开发的角度, 是不关心具体物理数据库的连接的, 这部分的工作一般都是由数据库访问层或者公共的数据访问组件来完成.

(4) 同一编号的表族部署到同一数据库

以分区键为核心, 应用会形成一个表族, 经过哈希运算后, 会产生一组同一编号的数据桶, 在数据模型上会表现为一组数据库表, 为了最大程度的避免跨库的操作和分布式事务, 这些表都应该部署到同一个数据库中.

(5) 将所有表族按照物理数据库数量均匀分布

物理部署时, 在数据库层一般都会使用同样型号的物理设备, 为了保证负载的均衡, 一般会采取均匀分布的方式, 将表族部署到目标数据库. 这样可以保证多个物理库在数据量上的平衡性. 通过一致性哈希算法也可以基本保证从数据的访问上能够均匀分步到所有的数据桶中, 避免出现数据热点. 如果某个数据桶成为了热点, 通过动态的数据重分布工具也可以快速地将该数据桶相关的表族迁移到负载相对较低的其他物理数据库.

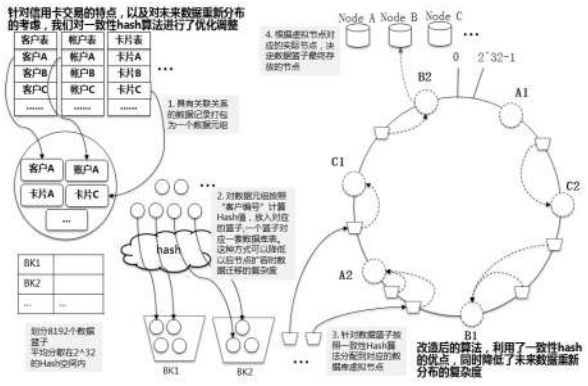

图 3　数据热点重分布[11]

从实际投产后的运行情况来看, 经过上述方案改

造后的交易系统, 各物理数据库的处理压力基本是均衡的状态, 不存在热点特别突出的情况.

3.3 数据访问路由

算法解决了分库分表的最核心的问题, 接下来就是要将算法嵌入到交易流程中, 这是实施应用迁移和改造的必要条件.

从银行核心业务系统的交易处理流程来看, 一般都分为请求接入、调度控制、应用逻辑处理、外部服务调用和数据库访问几个部分. 其中应用逻辑、外部服务调用和数据库访问逻辑是应用开发的主要内容, 其他功能基本由底层开发框架实现. 数据访问路由的难点在于, 需要在不影响主体业务逻辑的前提下, 将数据分布算法嵌入到交易流程中, 使应用具备分库分表的能力.

首先, 需要在请求接入时确定应用的分区键, 输入是服务的请求数据, 然后根据应用场景的不同嵌入分区键的识别逻辑, 实现分区键的转换功能. 转换后的分区键和计算的桶编号需要写入本交易的内存区, 用于后续的分库分表处理.

其次, 就是要改造数据库访问组件. 应该统一应用访问数据库的接口, 这是能够实现应用透明化处理的前提. 接口统一后, 后续就可以实现数据库访问层, 嵌入数据访问路由逻辑, 核心功能包括 SQL 的解析和重写, 多数据源管理, 单库数据访问接口等, 通过几个模块的协作就可以完成数据访问路由的决策、分发和执行.

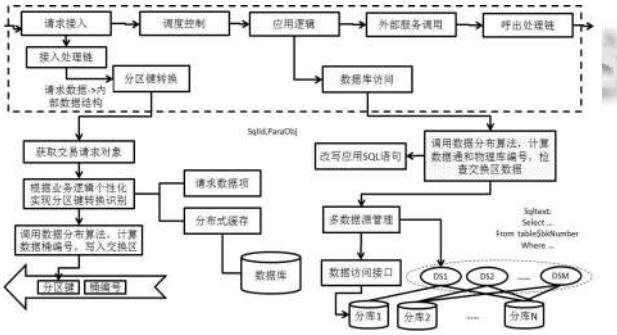

图 4 数据访问路由[11]

3.4 数据重分布

数据重分布是分库分表后的基本能力要求. 重分布过程应该以最大程度地的保证应用可用性为前提, 不能因为重分布导致全量数据的重组. 从服务视角来看, 每一次的服务调用, 都是针对单一的数据记录进行操作, 数据表中存储的也是数据记录. 基于关系型数据库进行数据重分布, 最好的方式并不是通过记录, 而是通过表. 将重分布的颗粒度定义到表级别, 可以利用数据库的导入导出工具, 大大提高数据重分布的效率, 降低对系统可用性的影响. 实际操作中, 可以将数据桶映射为物理数据库的表, 作为数据迁移和重分布的基本单位.

3.5 数据聚合

跨库的查询是分库分表模式下最为典型的应用场景, 比如柜员代客户进行交易时, 经常需要使用操作员编号进行各类的查询, 而一般应用的分区键很少会选择操作员编号, 这样就必然会发生跨库的数据库访问. 解决的思路就是将查询下发到所有的数据库执行, 然后再对结果集进行合并和筛选. 方案主要包括以下几部分:

(1) 并行执行调度

要保证跨库查询的效率, 必须要采用多线程的方式并发处理, 需要有一个集中的调度模块, 来协调其他模块的功能. 该模块需要集成到数据库访问组件中, 对上层应用屏蔽后端复杂的实现细节, 保持单一数据库下的应用开发模式.

(2) 多分区键识别和重组

该模块需要根据应用提交的请求参数对象完成分区键的识别和重组功能, 确定多分区键与目标物理数据库的映射关系. 最为重要的就是要根据数据分布算法, 确定本次查询涉及的目标数据库, 尽量减少跨库的访问范围, 提高并行查询的处理效率, 减少资源占用.

(3) 线程池管理

由于创建线程的成本较高, 引入线程池管理模块, 可以最大程度地提高线程利用率, 避免反复创建线程引起的性能消耗, 提升系统处理效率.

(4) 多数据源管理和数据访问接口

由于采用分布式架构, 应用服务需要面临跨多个物理数据源访问数据, 需要平台层支持将访问多数据源数据以接口形式便利的提供给应用使用.

(5) SQL 解析和执行计划优化

跨库场景下的 SQL 执行, 语句将被下发到多个目标数据库执行, 涉及查询条件和数据表名的重写. 另外, 为了提高执行效率, 也需要对语句的执行计划进行优化.

(6) 结果集合并筛选

为了保证返回应用结果的正确性, 要求对各分库执行返回的结果, 进行再次的合并和筛选.

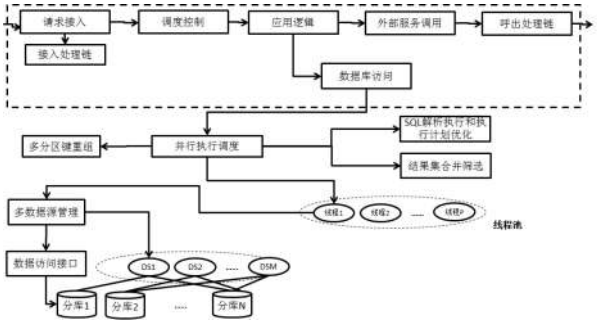

图 5　跨库数据查询[11]

3.6 数据共享和访问性能优化

银行核心业务基本都是提供核心的产品级功能, 比如存款、贷款、贷记卡, 强调参数化的设计, 不管是业务还是技术都有很多的系统参数. 类似汇率、定价这样的公共数据, 也面临在分布式架构下如何实现数据共享的问题. 这些数据基本都是读多写少的特征, 这正是分布式缓存技术适用的场景, 其特性包括:

(1) 高性能: 分布式缓存以服务器内存作为数据存储的介质, 提供以 KEY/VALUE 形式的存储, 减少在高并发数据访问时产生磁盘 IO 瓶颈, 提供高性能的数据读写服务.

(2) 动态扩展性: 分布式缓存技术支持弹性扩展, 可以做到在不影响业务逻辑的情况下, 实现节点的扩容, 提供可预测的性能与扩展性, 应对数据变化带来的负载.

(3) 高可用性: 支持数据和服务两个方面的可用性, 缓存基于冗余机制保证数据的高可用, 各个节点之间以对等的地位存在, 无单点故障, 支持故障的自动发现, 透明的实施故障隔离, 保证服务的连续性.

(4) 易用性: 提供统一的数据与管理视图, 对外提供简单的读写接口, 屏蔽后端数据的分布式拓扑结构. 通过管理视图, 可以在一点集中管理和监控缓存的运行情况.

但缓存也有一个问题: 缓存的数据都存储在内存中, 如果出现服务器宕机和进程异常退出的情况, 极有可能造成数据丢失. 在应用时应该遵循以下几条原则:

① 应用缓存的交易必须具有大并发量的特性; 放在缓存的数据应当是非源数据的副本, 满足只读且使用频繁的条件, 比如参数数据、全局索引信息等.

② 由于银行业务对可用性的要求, 在使用缓存时要从设计态做到应用的高可用, 一旦缓存出现问题, 应用要能够做到无缝切换.

③ 应用缓存技术时, 应该注意将设计点集中在数据访问模块, 避免对整体业务逻辑的影响.

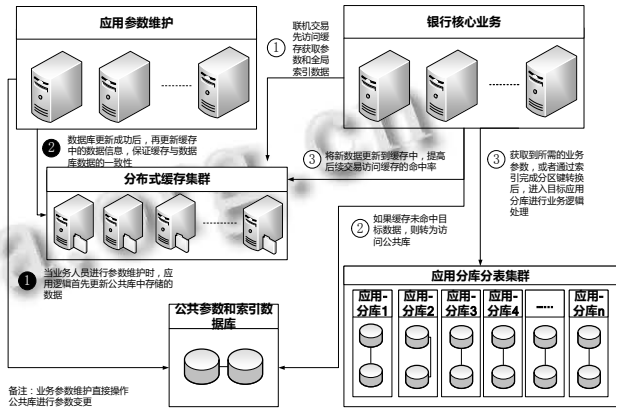

图 6　基于分布式缓存的数据共享访问[11]

3.7 读写分离

读写分离技术是分布式架构下提升系统处理能力的主流技术. 一般由一个主数据库和多个从数据库构成. 主库负责数据更新和实时数据查询, 从库负责历史和准实时的数据查询. 分布式架构设计时, 可从下面几个层级实现读写分离机制:

(1) 组件级: 与垂直切分的原理相同, 即从服务的维度将历史查询和准实时查询从核心应用的主库中隔离出来, 在企业级服务总线(简称 ESB), 实现组件和交易级的服务路由选择, 服务提供方可以针对读库和写库处理分别发布不同的交易码, 从而在组件级实现读写分离要求.

(2) 联机服务级: 当交易请求通过 ESB 到核心业务系统后, 在请求接入层按交易码进行数据访问路由选择, 在应用内部实现交易服务数据访问的读写分离.

(3) 数据访问层级: 核心业务系统最终还是要通过访问数据库完成业务逻辑处理, 在数据访问层中实现了对于 SQL 语句的解析和识别, 如果是可以访问读库的查询功能, 则可通过多数据源管理模块路由到读库访问, 降低主库的处理压力.

3.8 可用性保证机制

应用分库分表后, 在大幅提升系统处理性能的同时, 规避了大型单一集中式数据库的运行风险. 关于

可用性保证机制, 可以从以下几个方面设计:

(1) 故障隔离机制

分库分表后, 包含多个物理数据库, 当某一个分库发生故障时, 应用必须具备故障隔离的能力, 避免影响应用整体的可用性水平. 通过“SBF”数据分布模型, 可以准确地识别数据桶编号与物理数据库的对应关系, 快速地通过改变物理数据库可用性的标识, 实现故障隔离.

(2) 集群架构

目前主流的关系型数据库都有集群机制, 在集中式架构下经过了长期的生产运行检验, 是数据库可用性的关键保障. 在核心业务系统应用分库分表后, 每个分库仍然应该保持集群架构, 进一步提升分库的可用性水平.

(3) 数据库的灾备

集群架构在某些情况下也可能会出现失效, 引起分库的访问异常. 针对非常关键的核心业务数据库, 可以进一步通过关系型数据库的灾备技术, 部署灾备数据库, 保证极端情况下数据库级的可用性.

3.9 高效运维

分布式架构下的应用具备了弹性能力, 不再依赖单个节点或者基础软硬件的可靠性, 而是通过架构来保证系统整体的可用性. 但随着云计算和虚拟化技术的应用, 应用内部的结构和服务器的规模越来越大, 在投产部署、应用监控、故障诊断、应急处置等方面, 大大增加了运维的复杂度[7]. 分布式架构下的高效运维体系, 不仅涉及网络、存储、计算等基础资源, 还涉及云管理、监控、运维流程、配置管理等 IT 服务管理. 其中云管理平台是核心环节, 云管理平台基于开源项目 OpenStack 的架构设计, 根据银行数据中心运维的特点进行适当扩展, 主要增加了和运维有关的日常检查、软件分发、配置管理和服务启停等功能模块. 在技术架构方面, 主要由用户门户、服务管理、流程引擎、消息总线和资源适配五部分组成. 其中, 用户门户实现界面接入; 服务管理实现资源调度和资源管理, 以及云服务相关的镜像、版本、容量和计量等管理、运维相关的自动化操作等内容; 流程引擎负责服务流程的编排和调度; 消息总线负责平台内各组件的消息传送; 资源适配负责对网络资源、存储资源和计算资源等物理资源的具体操作. 投产以来显著提高了运维的自动化程度. 同时, 集中监控系统的建设, 实现了从基础设施到应用的全方位监控, 为分布式架构下的快速故障定位和应急处置提供了有力的工具.

4 结论

近年来, 以云计算、大数据、社交网络等为代表的新一代互联网技术的迅速崛起, 并不断向金融领域渗透, 传统金融 IT 领域迎来了新的变化动力. 无论是从外部面临的竞争和监管要求, 还是从自身 IT 发展的形势来看, 分布式架构都是未来发展的趋势, 关键是在自主可控的前提下, 找到适合自己的发展道路. 分布式架构建设的目标并不是要替代传统的集中式架构. 目前来看, “集中式+分布式”的融合架构, 仍然是最先进的也是最适合银行业务特点的架构方案.

分布式架构研究和应用的目标是通过对融合架构的持续优化, 实现云计算在银行业务的落地, 特别是在核心业务的应用. 通过近年来的工作, 分布式架构已取得了一定成绩, 已经成功地在一些大型的核心业务系统上进行了分布式架构的尝试, 也已经顺利投产. 投产系统客户数超过 8000 万, 账户数 1.2 亿, 经过分库分表处理后, 数据分布到 1024 套表中, 部署在 6 套物理数据库中, 每套表约存储 12 万左右的账户记录和相关明细数据. 联机日均交易量约为 5000 万笔, 峰值 TPS 约 1200, 交易平均处理时间约 50ms. 批处理共 1731 个作业, 整体批处理时间约 2.5 小时. 系统整体非功能指标要明显优于老系统, 完全符合设计预期.

实践证明, 商业银行通过自主设计、自主研发的方式实现分布式技术的应用是可行的, 实施效果比外购商业软件具有明显优势. 银行引入分布式技术, 应对互联网行业竞争, 应发挥在网络资源、经营对象、信息安全及线下服务的优势, 坚持以自身为主做“银行+互联网”、以开放的态度谋求合作性竞争、以客户体验至上为宗旨设计产品提供服务、以积极稳妥的态度推进分布式架构的建设, 不断提升银行 IT 的管理和技术水平.

参考文献

1 金磐石.“产、用”战略联动推进信息化自主可控进程.金融电子化,2014,(8):28–29.
2 金磐石.科学构建灾备保障体系.金融电子化,2013,(1):23.
3 彭渊.大规模分布式系统架构设计与实战.北京:机械工业出版社,2014.

4 金磐石.建设银行云计算数据中心及运维体系建设实践探讨.中国金融电脑,2014,(7):18–22.
5 张校逸,成华.分布式数据库:金融行业关系型数据库新选择.金融电子化,2015,(1):81–82.
6 黎刚.分布式云计算技术在金融领域的应用研究.财经界(学术版),2012,(9):14.
7 张建文,汪鑫.云计算技术在银行中的应用探讨.华南金融电脑,2009,17(6):17–19.
8 孟繁锦.构件金融分布式多活数据中心.中国金融电脑,2014,(8):87.
9 刘启滨.NOSQL 在金融行业的应用分析.金融科技时代,2013,(10):63–64.
10 数据库分库分表(sharding)系列 http://blog.csdn.net/bluishglc/article/details/7696085.
11 内部资料-建行新一代分布式架构详细设计.