



北京金融科技产业联盟
BEIJING FINTECH INDUSTRY ALLIANCE

分布式数据库单元业务应用 研究报告

北京金融科技产业联盟
2022 年 12 月

版权声明

本报告版权属于北京金融科技产业联盟，并受法律保护。转载、编摘或利用其他方式使用本白皮书文字或观点的，应注明来源。违反上述声明者，将被追究相关法律责任。



编制委员会

主任：

潘润红

编委会成员：

聂丽琴 王志刚 李晓栋 邢 磊 胡 捷

编写组成员：

王莉莉	杜 蓉	李萧萧	王鹏冲	李中原	夏文勇	王 辉
李晓欢	林 海	高孝鑫	张 楠	卢道和	胡盼盼	王 枫
王 榆	杨详合	江智睿	王睿操	苏 强	张 晓	陆天炜
戴 扶	申立军	周日明	明玉琢	苏德财	黄元霞	张积斌
黄小慧						

主审：

黄本涛 彭卫华

统稿：

张 蕤

参编单位：

北京金融科技产业联盟秘书处

中国光大银行股份有限公司

中国工商银行股份有限公司

中国平安银行股份有限公司

中国华夏银行股份有限公司

中国建设银行股份有限公司

北京万里开源有限公司

中兴通讯股份有限公司

深圳前海微众银行股份有限公司

北京奥星贝斯科技有限公司

腾讯云计算（北京）有限责任公司

成都虚谷伟业科技有限公司

北京百度网讯科技有限公司

广州巨杉数据库软件有限公司

上海热璞网络有限公司



摘要

在银行业客户体验和数字科技创新双轮驱动推进下，微服务、单元化架构、分布式数据库技术的融合已经成为传统银行基础架构演进的一个重要技术趋势。作为一个新兴的技术架构，微服务结合单元化架构与分布式数据库技术的融合也伴随着众多的难点与挑战，包括但不限于高可靠与容灾、单元拆分以及整体运维体系复杂度的提升。

本报告将整理金融机构分布式数据库在单元化场景部署实施需求以及特性需求，并从单元化拆分、单元与分布式数据库部署对应、单元扩容、高可靠、灰度发布、数据同步、以及运维解决方案等多方面阐述分布式数据库在单元化业务场景下的部署思路，最后提供金融行业典型试点案例，为金融机构在单元化业务应用场景中使用分布式数据库提供参考。

目 录

一、研究背景	1
(一) 单元化概念及架构	1
(二) 单元类型	3
(三) 分布式数据库与单元化	5
二、场景分析	8
(一) 单元化场景分析	8
(二) 分布式数据库分析	9
(三) 部署难点与要求	12
三、应用方案	16
(一) 单元业务拆分	16
(二) 业务拆分数据库设计	26
(三) 数据库部署方案	31
(四) 数据库运维要求	47
四、典型案例	55
(一) 建设银行试点案例	55
(二) 平安银行试点案例	62
(三) 华夏银行试点案例	71
(四) 微众银行试点案例	76
(五) 支付宝试点案例	86

一、研究背景

(一) 单元化概念及架构

1、单元化概念

单元是指一个能完成特定业务操作的自闭环集合，在这个集合中包含了特定业务所需的关键服务，以及分配给这个单元的数据。一个单元是可以独立运行特定业务的最小集合；单元化架构就是把单元作为部署的基本单位，在全栈所有机房中部署若干单元，每个机房里的单元数目不定，任意一个单元都部署了系统所需的特定应用，数据则是全量数据按照某种维度划分后的一部分。

单元是一个缩小版整站，部署了特定业务应用，但他不是全量的数据，因为一个单元只能操作部分数据。以银行账户核心的存贷款服务为例，单元化架构之后，一个单元可以只存储个人存款服务这个特定业务应用的部分客户号相关的数据，但这些客户号所有或绝大部分的个人存款服务都可以在这个单元中完成。

2、单元化架构

单元化架构是从并行计算领域发展而来。在分布式服务设计领域，一个单元（Cell）就是满足某个分区所有业务操作的自包含集合。而一个分区（Shard），则是整体数据集的一个子集，如果你用账号来划分用户，那同样尾号的那部分用户就可以认为是一个分区。单元化就是将一个服务设计改造让其符合单元特征

的过程。

能够单元化的系统，很容易在多机房中扩展，且不受机房建设上限限制，因为可以轻易地把几个单元部署在一个机房，而把另外几个部署在其他机房。如图 1 所示，通过在业务入口处设置一个流量调配器，可以调整业务流量在单元之间的比例。

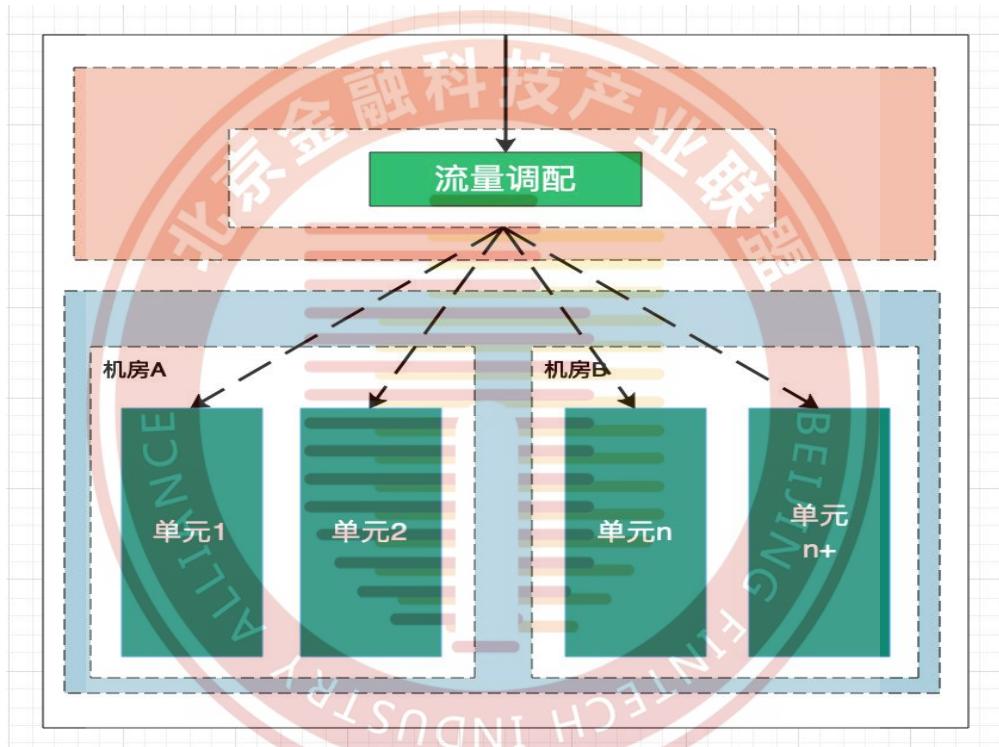


图 1 业务单元化示意图

单元化架构要求系统须具备数据分区能力，数据分区承载了各个单元的业务流量比例。数据分区即是把全局数据按照某一个维度水平划分开来，每个分区的数据内容互不重叠，每个数据节点有自己应用系统、数据库。

从全行系统的角度看，单元化按照技术或者业务等一定方式

将系统拆分到不同的机房里面去，使业务链路尽可能在单机房内完成，其效果是既降低了链路耗时，又实现了多机房多活且为多机房扩展提供基础。

从技术拆分的角度来看，单元可以分成可拆分和不可拆分两类，如账务、存款、贷款、资产交换等绝大多数银行业务是按照人维度可以进行拆分的，如客户、限额等存在多个维度或者非人的维度，因此不可拆分；客户登录可能存在身份证号、银行卡号、手机号、邮箱等多个维度。

（二）单元类型

目前业内对于单元的命名方式有多种，例如邮储银行分区单元（DUS）、民生银行分区单元（DUS/U Zone）、网商银行分区单元（G/R/C Zone）、微众银行标准化部署节点（DCN）、光大银行分区单元（DUS）及其他企业分区单元（SET）。不同的金融机构实际的 DUS 划分方式可能会有些不同，但基本思想大体一致，本文为了方便论述，统一使用分区单元（DUS）来表述。

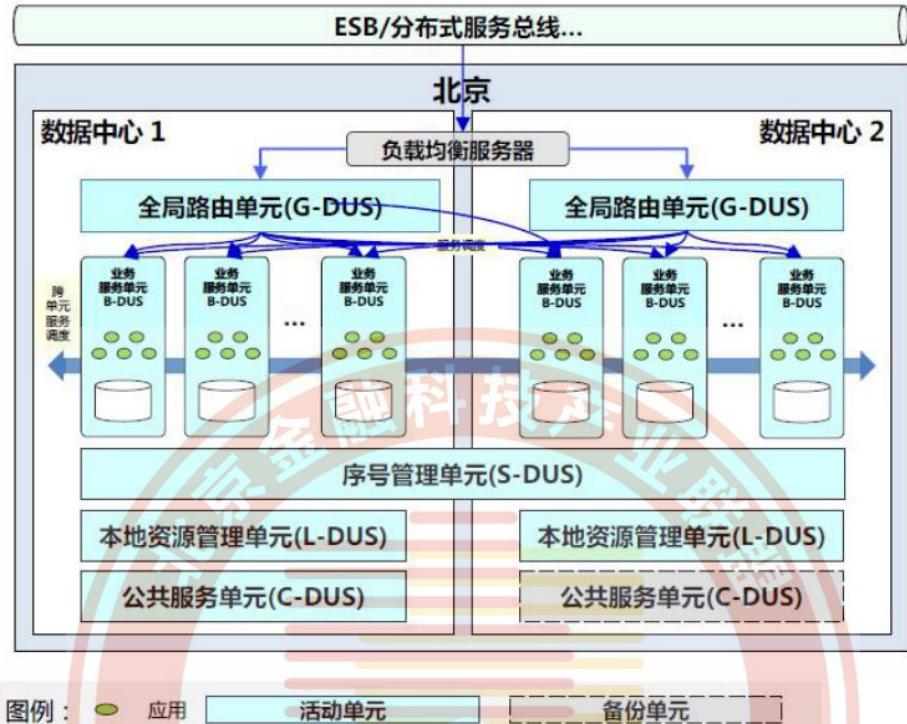


图 2 常见 DUS 划分示意图

如上图 2 所示，常见的 DUS 划分主要如下：

1. 全局路由单元 (G-DUS)

包含网关服务，处理外联系统的服务请求，提供服务路由，将外部请求转发到内部的业务单元的应用服务实例。典型的应用服务为联机网关服务、文件网关服务、外呼等。

2. 业务服务单元 (B-DUS)

包含多个客户的业务数据以及为此类客户提供服务支持的应用实例，负责完成客户的业务处理。典型的应用服务为联机服务、批量主控服务、批量处理服务、批量转联机服务。

3. 序号管理单元 (S-DUS)

为全部服务处理单元提供系统服务支持,如账号等序列号生产服务,该单元的特点是在核心系统内以单点方式提供服务。典型的应用服务为全局序列号管理服务。

4. 公共服务单元 (C-DUS)

为全部业务处理单元提供支撑的应用服务,以及与自动化运维管理平台相对接的相关数据,包括批量调度服务、日志聚合信息、监控数据等。典型的应用服务为分布式批量调度服务。

5. 本地资源管理单元 (L-DUS)

包括针对单个数据中心内的业务处理单元所提供的公共数据以及该数据的管理服务,包括映射数据、非客户维度的参数(业务、技术参数)以及此类数据的管理服务。典型的应用服务为参数管理服务、服务注册管理服务、映射关系管理服务、配置信息管理服务。

本课题更多关注单元化与分布式数据库的结合,所以后续对于单元拆分方式将聚焦在业务服务单元 (B-DUS) 的拆分方案。

(三) 分布式数据库与单元化

分布式数据库系统通常使用较小的计算机系统,每台计算机可单独放在一个地方,每台计算机中都可能有 DBMS 的一份完整拷贝副本,或者部分拷贝副本,并具有自己局部的数据库,位于不同地点的许多计算机通过网络互相连接,共同组成一个完整

的、逻辑上集中、物理上分布的大型数据库。

典型通用的分布式数据库架构如图 3 所示，分为计算层和存储层。计算层包含多个计算节点，存储层包含众多数据节点。计算节点的作用不同分布式数据库略有些差异，本文不展开论述；而作为各个分布式数据库的共性点，数据节点负责存储业务单元数据，数据节点将以多副本数据冗余的方式提供业务数据的高可靠，用于存储一份数据多个副本的数据节点我们称为节点组或数据节点组，本文的后续部分将会多次提及节点组或数据节点组。

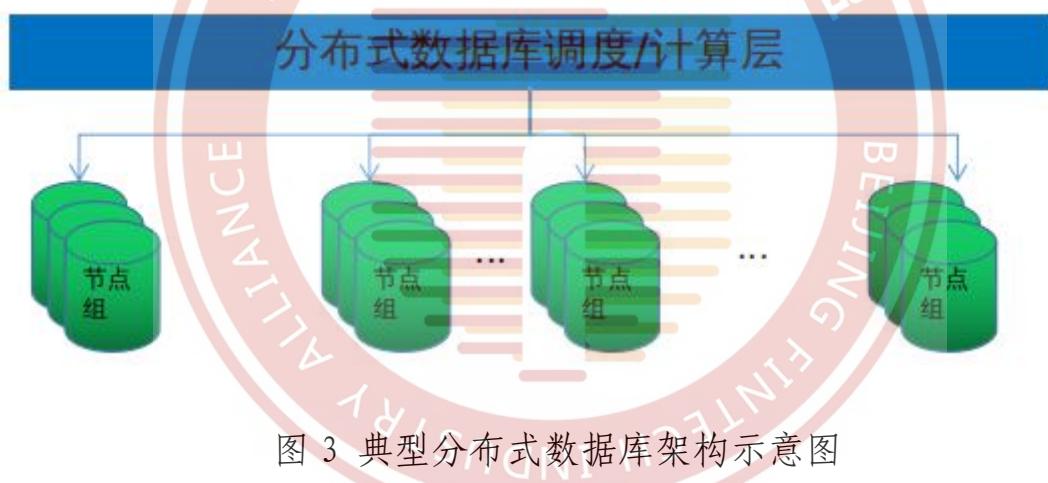


图 3 典型分布式数据库架构示意图

分布式数据库在单元业务下部署有助于保障线上系统日常业务的高效能、高安全、高迭代的平稳运行，同时有助于金融机构、金融企业提升信息化综合能力，此外还有助于提升国产数据库领域的支撑能力。对于金融机构、金融企业完善技术平台、逐步实现基础软件自主可控有重要的现实意义。

单元化架构与分布式数据库技术的融合作为一个新兴的技

术架构，也伴随着众多的难点与挑战，包括但不限于链路延迟、高可靠与容灾、单元拆分以及整体运维体系复杂度的提升，而这些也是本课题需要进一步研究和讨论的。

在银行业客户体验和数字科技创新双轮驱动推进下，传统银行基础架构加速向微服务与单元化转型，向支撑更加开放与灵活的业务架构演进。同时随着国家基础软件自主可控浪潮的推进，分布式数据库技术也越来越多地应用在了银行业各类重要/核心系统中。微服务、单元化架构与分布式数据库技术的融合成为一个重要的技术趋势。

为此，北京金融科技产业联盟在分布式数据库专委会内开展了单元业务下应用场景的调研，并组织编写了《分布式数据库单元业务应用场景研究报告》，为各金融机构与分布式数据库厂商提供参考。

二、场景分析

(一) 单元化场景分析

1. 应用需求简述

对本课题相关参与单位反馈的调研结果进行分析研究，发现单元化架构与分布式数据库技术融合的技术路线已作为主要 IT 基础架构在部分金融机构中进行试点或落地；同时有大量金融机构正在调研中，并且将单元化和分布式数据库的结合作为重要的技术选型架构方向之一；整理金融机构分布式数据库在单元化业务应用场景部署实施需求及特性需求，对于各金融机构与分布式数据库厂商均具有参考价值。

本报告分析的分布式数据库在单元业务中的应用场景需求汇总如下表 1 所示：

表 1 分布式数据库在单元业务场景中应用技术需求汇总

编号	需求
1	典型单元化架构业务拆分方式
2	单元与分布式数据库分片的部署对应关系
3	单元调整过程中业务服务单元与分布式数据库配合的典型实施方案，包括横向与纵向扩容、单元拆分与合并。
4	进行单位化拆分后，分布式数据库的本地高可靠、同城双活或多活、异地灾备等部署的典型方案，总结分布式数据库多副本的高可靠性技术。
5	单元化架构下业务的灰度发布过程与分布式数据库的灰度发布过程
6	单元化架构下分布式数据库对外提供数据同步汇总的典型技术方案
7	研调整理单元化架构下分布式数据库应用的典型运维解决方案，包含该场景下全链路性能检测、备份恢复等技术方案。

2. 单元化场景应用痛点

本课题在实际调研过程中，针对国内多家银行、金融机构的应用情况总结得到在生产应用中需要采用单元化方式解决的痛点有：

跨异地跨机房多活容灾。类似两地三中心、三地五中心部署架构，通过架构单元化改造实现业务全局的异地多活。

业务地域拆分与负载均衡。通过单元化架构、将业务按照地域进行单元拆分、实现有效的负载拆分、多地多活、业务高可靠与负载均衡。

隔离故障提升系统整体稳定性。单元化本身就有一定的隔离性，这样在单一业务或系统出现故障时不会影响其他的业务、系统或数据，最小范围的系统风险。同时多个或者所有单元同时出现故障的可能性大大降低，另外也减少了故障的排查、恢复时间。

通过灵活的横向扩展能力满足业务发展要求。针对现有业务，原有系统的在性能、扩展性、可用性、安全性等方面的问题，通过单元化的重构改造进行改进，以实现高性能、高可用、高安全、高可扩展的目标。

（二）分布式数据库分析

1. 应用场景

分布式数据库是为单元化业务改造提供重要支撑的核心技

术和产品，本文针对“分布式单元化场景在银行内部是否需要”、“分布式单元化场景在银行内部是否有场景”、“分布式单元化场景是否是银行关心或迫切需要改进的问题”等进行了调研。经过与多家银行、金融机构沟通，目前银行、金融机构采用或有意向在单元化业务中下采用分布式数据库的比较有代表性的应用场景包括：联机交易、批量处理、前置系统和客户服务系统这四个场景：

联机交易是系统对外直接提供的交易，该类交易具有以下特征：

事务性—分布式强一致读写；

实时性—交易有生命周期，并有超时机制等，调用方需实时等待被调方的反馈，成功或失败皆有反馈；

并发性—同一类甚至同一个交易可同时被多个线程调用，相互间有锁处理机制。可将同一类或者同一个交易类型进行单元化处理，将其整个流程封装成一个闭环单元，供多个线程调用，保证单元之间相互隔离。

批量处理时该交易相关的参数、系统状态已经锁定，系统需要进行的操作具有统一性，使用相同的规则处理大量数据；批量交易具有串行性，并不是说批量交易中不能有并发，而是有固定的步骤逐步操作，每一步都有对某些条件的依赖。可将规则封装成单元化，每个规则相对独立，同时在做批量交易时可按照一定

的步骤，逐个执行单元，类似面向对象或面向组件的思路。

前置系统是处理银行和银联之前的交换业务的系统，主要负责行内系统（如行内渠道前置、行内交换平台、行内 ESB 系统）、CUPS（银联交换系统）间的报文格式转换，同时对交易异常做处理以保证联机交易的完整性和数据一致性。将前置系统单元化，形成统一的报文格式转换，可保证整个联机交易的事务完整性和数据一致性。

客户服务系统：将银行内流程进行梳理和统一，并将其单元化，形成独立的个体，将具体的业务办理系统（自动或人工）与客户服务系统形成接口对接，便于以后的更新、升级和迁移等操作。

2. 架构分析

通过调研分析，单元化业务场景中分布式数据库的典型部署场景见图 4。

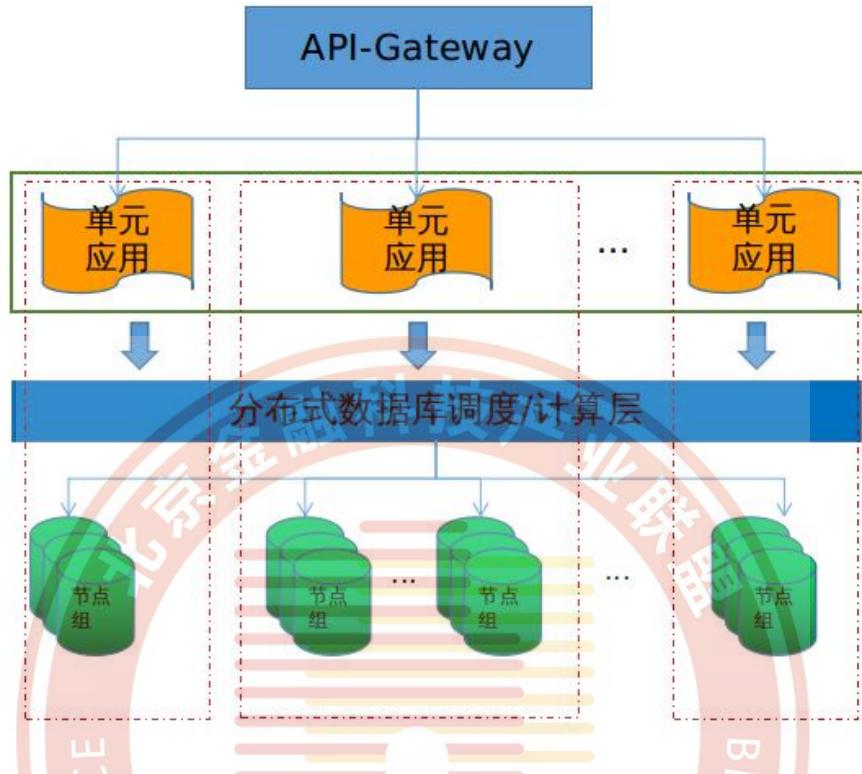


图 4 单元化业务场景中分布式数据库的典型部署场景示意图

单元化业务的单元内数据存储在分布式数据库集群的 1 组存储层节点中，不同的单元存储在不同的存储层节点组中，但他们都属于同一个分布式数据库集群，共享调度/计算层。如图 4 所示。每个单元存储所用的 1 组存储层节点可以是分布式数据库的 1 个带数据高可靠的数据分片；同时针对较大的业务单元（例如按地域划分单元后的热点地域），1 组存储节点也可以是多个带数据高可靠的数据分片。单元与单元间通过使用物理上隔离的存储层节点，实现单元间数据的隔离。

（三）部署难点与要求

1. 部署难点

基于本次调研表的反馈信息整理，单元化业务场景中与分布式数据库部署结合时有如下部署实施难点或需求：

单元化拆分与分布式数据库分片技术的融合。单位化架构需要业务方按照业务单位化进行拆分。单元化架构需与分布式数据库的分片技术相结合，进行单位化拆分处理。单元的拆分需要做好事前预估测试与后期监测工作。目前调研反馈主要的单元拆分方式包括：地域（含分行）、客户信息（含编号、账号等）等。单元与分布式数据库分片的部署对应关系也是下一步的研究整理内容。例如一个单元对应一个分片或一个单元对应多个分片，也有分库分表分集群的部署方案。

单元化架构的单元调整与分布式数据库技术的融合。随着业务规模的扩展，初始划分的单元无法继续长期支撑，就需要进行单元扩容，增加新的单元。扩展单元的过程除了单元业务的扩展外，也涉及分布式数据库的扩展。存在业务复杂度增加导致的单元内负载增加的情形。这种场景下常见的方法是进行单元纵向扩容，例如增加机器配置；同时如果主要是数据库侧负载的增加，也可能存在分布式数据库层扩容的需求，包括横向扩容和纵向扩容。

单元化架构高可靠与容灾与分布式数据库技术的融合。单元化架构增加了由于单个单元出现故障对整体可用性影响的概率，

但是由于多个单元同时不可用的概率降低，因此系统整体的可用指标大大提高。以分布式数据库为例，数据库每个分片的数据副本数与数据副本的部署方式，以及不同部署方式与业务重要度的对应关系。每个单元内分布式数据库本身的数据高可靠技术是单元化高可靠的基础，原则上，数据库 RPO 必须为 0，RTO 时间需可控。

单元化架构灰度发布过程与分布式数据库技术的融合。灰度发布是在线业务持续运行的基本要求之一。

跨单元数据同步汇总与分布式数据库技术融合。对于单元化架构下，对跨单元的数据进行同步汇总，以支持将部分批量或分析处理的业务需求可以在线下其他的技术架构中进行支撑。这个过程涉及从单元化下数据存储的分布式数据库的数据抽取、同步、验证等技术。

2. 能力要求

本次调研“单元化业务场景下是否有对分布式数据库操作需求”，从单元内、跨单元以及数据同步汇三个维度进行了收集。其中数据汇总部分已经在“单元化场景与分布式数据库结合的部署实施需求”章节中进行了覆盖，而跨单元操作的需求普遍较少，所以重点关注单元内的数据库特性需求：

单元化后单元内数据存取性能与监测。单元化改造，特别是基于微服务改造后，服务模块间调度链路增加，SQL 语句交互次

数也增加。

可靠性与容灾。参见“单元化场景与分布式数据库结合的部署实施需求”章节中相关内容。

扩展性。参见“单元化场景与分布式数据库结合的部署实施需求”章节中相关内容。

易用易维护。单元化架构改造后，从运维角度，由于机器与节点数的增加，整体运维复杂度也大幅增强。如何确保多个单元之间的运维一致性对运维能力提出更高的要求。



三、应用方案

以下方案描述中使用的术语及定义参考表 2。

表 2 术语定义

关键术语	定义解释
单元	指一个能完成特定业务操作的自闭环集合，在这个集合中包含了特定业务所需的关键服务，以及分配给这个单元的数据。
业务服务单元 (B-DUS)	包含多个客户的业务数据以及为此类客户提供服务支持的应用实例，负责完成客户的业务处理。
节点组 (或数据节点组)	用于存储一份数据多个副本的数据节点我们称为节点组。
小单元	一个单元部署在分布式数据库的一个数据节点组上。
大单元	一个单元部署在一整套分布式数据库集群上。
CMDB	配置管理数据库。

(一) 单元业务拆分

1. 拆分策略

业务服务单元的拆分策略主要分为两大类：垂直拆分与水平拆分。

1) **垂直拆分**。垂直拆分指业务单元按照产品领域维度进行切分，形成不同的产品单元。以核心系统为例，一个可能的垂直拆分方式为个人存款、个人贷款、对公存款、对公贷款、现金以及内部账。

2) **水平拆分**。水平拆分指业务单元按照地区、客户号（或

账户号)进行切分,每个单元只包含一个产品/业务的部分数据。常见的拆分策略包括:按照地区,例如按照分行进行拆分,通常拆分的单元比较大;按照客户号进行切分,包括 range 切分和 hash 切分。Range 切分就是每个单元固定一个客户号范围;Hash 切分就是将业务数据基于用户号进行 hash 散列,一个单元存储 hash 结果的一个或多个分片;按照地区与客户号切分的混合切分,例如先按照地区分,然后在地区内再按照客户号切分;除了上述常见的水平拆分方式外,还有一种基于时间的 range 拆分方法在相关机构的实际业务系统中落地的实践,即:按照时间进行切分,例如按订单生成的时间,每个月做一次且切分。

2. 典型方案

基于上一节“业务服务单元拆分策略”的内容,这里集中介绍如下 6 类典型垂直拆分产品后的业务单元拆分方案:

1) 水平按照客户号的 range 做拆分。可选的将不同的产品线拆分到不同的单元,每个单元会包含 1 个产品或多个产品的一个固定 range 范围的客户号,这是小单元设计,如果未进行产品线拆分,则每个单元包含所有的产品;单元客户容量上限固定,容量超出时新客户号进新的单元;一个客户号原则上固定属于某一个单元,不会出现客户号在单元间的挪动。图 5 是一个每单元 200 个客户号的拆分示例。

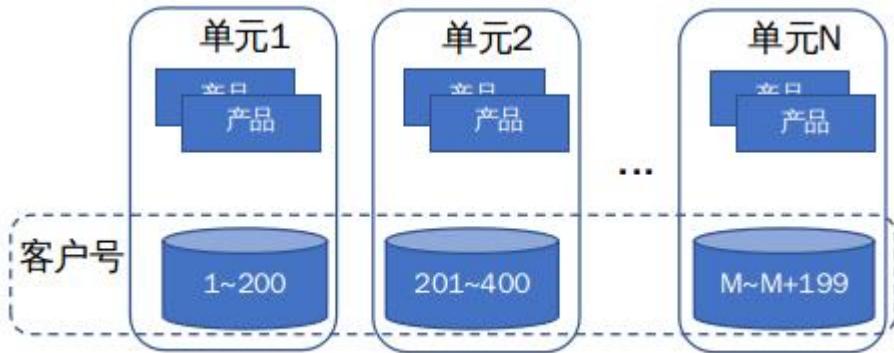


图 5 客户号 range 拆分示例图

2) 水平按照客户号的 hash 拆分，每个单元包含多个 shard 分片。可选的将不同的产品拆分到不同的单元，客户数据按照客户号进行散列，存储到一个给定数量的单元集合中，每个单元存储多个 hash 散列后的 shard 分片，是小单元设计，如果未进行产品线拆分，则每个单元包含所有的产品；单元客户容量过多时将新增单元并挪动已有单元的 shard 分片到新的单元，所以 shard 是扩容的最小粒度，shard 的数量一定程度上决定了架构的扩容上限，扩容时将出现客户号在单元间的挪动，如图 6 所示。

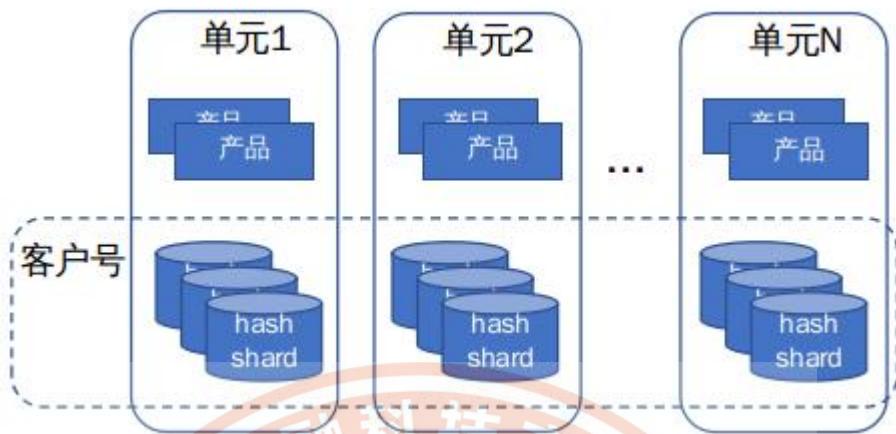


图 6 客户号 hash 拆分示例图

3) 先水平拆分区域，再按照客户号的 hash 拆分，每个单元包含多个 shard 分片。可选的将不同的产品拆分到不同的单元，数据先按照地域进行拆分，然后对于一个地域的数据再通过客户号 hash 散列存储到一个给定数量的单元集合中，每个单元存储多个 hash 散列后的 shard 分片，是小单元设计，如果未进行产品线拆分，则每个单元包含所有的产品。单元的管理与方案 2 “先垂直拆产品、然后水平按照客户号的 hash 拆分，每个单元包含多个 shard 分片”相同，如图 7 所示。

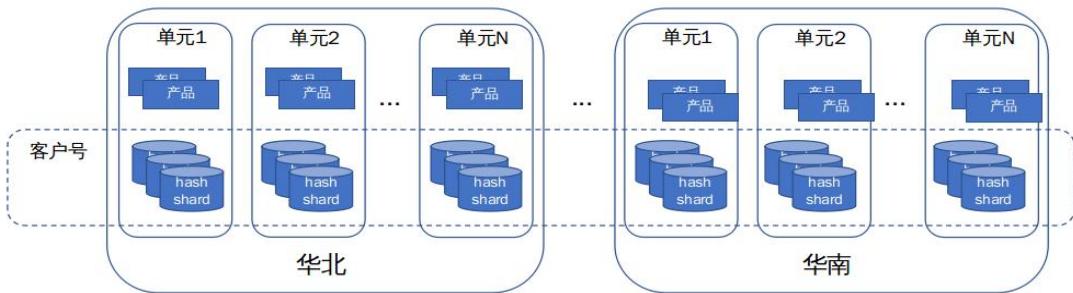


图 7 水平拆分区域后客户号 hash 拆分示例图

4) 先垂直拆产品然后水平拆分区域。不同的产品线使用不同的单元，产品线的数据先按照地域进行拆分；由于一个区域的数据通常会较大，所以是大单元设计；同时单元与单元的容量可能存在较大差异，单元的扩容也更多基于单元内的扩容手段，新增单元操作较重，如图 8 所示。



图 8 垂直拆产品后水平拆分区域示例图

5) 直接按区域拆分，大单元设计。类似方案 4 先垂直拆产品然后水平拆分区域，不同的是多个产品会对单元进行共用，同样是大单元设计，若图 9 所示。

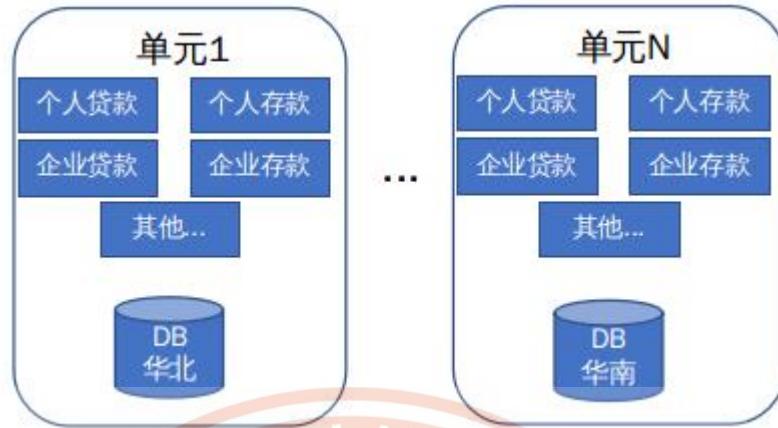


图 9 水平拆分区域示例图

6) 按时间进行切分，循环使用现有单元。针对类似于订单等特定业务模块，数据按时间进行切分，粒度通常到月或季度；订单类数据是一个持续增长值，有明显的时效性，因此基于时间进行拆分后，所有单元都会呈现一个有规律的替换的情况，例如需要定期新建未来单元，并定期归档历史单元数据，这要求系统有极好的单元调度能力，如图 10 所示。

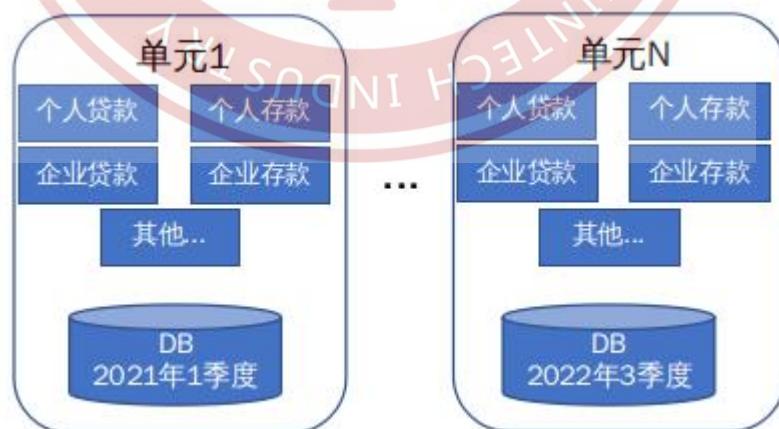


图 10 时间进行拆分示例图

对于不含拆分字段的数据，无法按客户维度进行切分，如定价、机构信息等，这些数据可以考虑的存储方案主要有 2 种：以副本形式保留在每个业务单元中以提高访问性能。通常要求这类数据变更频率低且数据量不大。另一种是单独部署一个单元，称为全局数据单元。

3. 方案对比

针对上文提到的 6 个典型业务单元拆分方案，本小节将从扩展性、故障可用性、降低跨单元操作比重、降低运维复杂度、降低建设成本等几个维度进行对比，如表 3 所示。

对于扩展性，小单元设计更便于架构未来新单元的扩展。方案 1、6 的扩展性最好，理论无上限，新单元扩容方式也轻量；方案 2 和 3 存在理论扩容上限，扩容过程也涉及用户数据的挪动；方案 4 和 5 扩容代价大，通常只是有限数量的单元。

对于故障可用性，方案 1 基于客户号 range 的切分方案隔离性是最好的，可用性影响最小，一个客户号原则上固定属于某一个单元，不会出现客户号在单元间的挪动，某个单元故障也不会影响其他单元。方案 2 和方案 3 基于客户号 hash 进行切分，存在单元挪动的可能，同时单元规模相比方案 1 也通常会更大一些，但从故障隔离的角度，方案 2/3 与方案 1 的效果基本相当。方案 4 和 5 是大单元设计，单元故障的隔离性意义不大，方案 4

存在产品拆分，相对更好。方案 6 虽然以时间段划分，隔离性好，但如果业务中是否存在历史数据聚合等业务场景，就需要引入其他方案用来保障汇集场景。

对于降低跨单元操作比重，大单元设计优势更大。方案 4、5、6 的扩单元操作，特别是跨单元分布式事务操作，比例小于方案 1、方案 2 和方案 3，同时方案 5 产品更像集中式部署架构，所以扩单元操作更少。

对于降低运维复杂度，显然架构集中度越高，拆分的单元越少，运维复杂度也越低；所以方案 5 和方案 4 复杂度相比方案 1、方案 2、方案 3 和方案 6 要低。对于方案 2、方案 3，扩容过程是否挪动数据对运维的复杂度会有明显影响；如果方案 2/3/6 的单元扩增过程主要是通过调整数据 shard 的主副本的分布实现，扩容过程不涉及数据挪动，同时因为扩容过程不涉及新的 shard 产生，对于生产的监控、备份、数据同步链路影响都很小，那么运维复杂度较低。后续对比中，我们将使用“方案 2/3（扩容不挪数据）”来表示上述情况。所以方案 1/6 的运维复杂度与“方案 2/3（扩容不挪数据）”相当，且优于“方案 2/3（扩容挪数据）”。同时对于方案 1 与方案 2/3/6 的运维复杂度对比，目前业内还有如下有代表性的观点，供大家参考：

网商银行认为方案 1 尽管不需要数据搬迁，但由于是新增单元中包含了新增数据库，则需要新增数据库实例、数据库监控、

同城备份、超远备份，以及数据库下游日志解析链路的同步等，甚至整个离线数仓等大数据体系跟随变动。显然方案 1 的复杂度远大于方案 2、3、6。若使用了支持一致性协议的原生分布式数据库，则数据搬迁的复杂度可忽略。举个例子，贷款扩容出新的单元，用户编号范围 200 万~300 万，则贷款的所有数据仓库的基表都需要增加新的同步任务，因为贷款的数仓基表是贷款的全量数据。

微众银行认为方案 1 在新增单元的过程中，确实需要进行新增数据库、新增上下游同步等操作，但是在方案 1 中，由于每个单元的配置完全对等，所以针对单元的扩容操作，是可以实现模板化和全自动化的，所以带来的运维复杂度是可控的。另外方案 1 在运维安全性上还有三个优势：

新增单元的过程中，对存量单元的用户以及服务完全不涉及，不会产生因为存量数据挪动而带来的可用性的风险；

对于新增单元的用户主量是可灰度的。可以通过权重精确控制，先放量极小数量的用户到新增单元，验证正常之后，再逐步放量更多的用户到新增单元。

可以较方便的设置一个专用的预发布单元。这个单元可以放置固定的用户（比如单位内部员工用户），专门用于版本的预发布和版本验证。有效降低版本更新带来的可用性风险。

对于建设成本，单元化拆分的越少，成本越低，单元化的拆

分通常会带来整体部署规模的增加。建设成本主要包括了方案搭建所需的服务器数量，以及方案运行所需的技术难度。以小单元的方案 1、方案 2、方案 3 为例，涉及更多的跨单元操作就需要更多依赖单元间分布式组件，例如分布式事务组件、分布式全局路由组件等，整体建设难度会高于方案 4 和 5。而从方案所需服务器的数量来说，由于方案 5 的集中化程度最高，服务器数量也相对更小；方案 1 的单元化拆分的力度更小，所需的服务器数量也相对更多。方案 6 仅适用于特定场景，所以需要单独评估，但通常会高于常规方案。

表 3 五种方案对比图

方案	扩展性	故障可用性	降低跨单元 操作比重	降低运维复 杂度	降低建设 成本
方案 1	优	优	差	中	差
方案 2/3 (扩容挪数据)	良	优	差	差	中
方案 2/3 (扩容不挪数据)	优	优	差	中	中
方案 4	差	中	良	良	良
方案 5	差	差	优	优	优
方案 6	优	中	优	中	差

表 4 网商银行针对上述对比表格内容的不同观点补充

方案	扩展性	故障可用性	降低跨单元操作比重	降低运维复杂度	降低建设成本
方案 1	优	优	差	差	差
方案 2	优	优	差	差	中
方案 3	良	良	差	差	中
方案 4	差	中	良	良	良
方案 5	差	差	优	优	优
方案 6	优	中	优	中	差

表 5 表格为微众银行针对上述对比表格内容的不同观点补充

方案	扩展性	故障可用性	降低跨单元操作比重	降低运维复杂度	降低建设成本
方案 1	优	优	差	中	差
方案 2	优	优	差	差	中
方案 3	良	良	差	差	中
方案 4	差	中	良	良	良
方案 5	差	差	优	优	优
方案 6	优	中	优	中	差

考虑到前 5 种方案更加常见，后续本文的讨论将主要集中在前 5 种方案相关的部署实施方案。

(二) 业务拆分数据库设计

本小节将讨论前文 5 类单元拆分方案应用时对应的分布式

数据库设计。

1. 水平按 Range 拆分

第一类是水平按照客户号的 range 做拆分。方案 1 中 1 个业务单元数据将存储到分布式数据库的一个节点组中；不同的业务单元使用不同的节点组；多个业务单元共享一个分布式数据库集群。

以银行账户核心的存贷款服务的个人存款为例，假设目前该业务有 5000 万的客户号，我们以每 500 万个用户为一个单元，所以需要 10 个单元；我们搭建一个含 10 个数据节点组的分布式数据库集群，每个数据节点组支撑 1 个单元；即第 1 到 500 万的客户号存放在第一个数据节点组，第 500 万 +1 到 1000 万的客户号存放在第二个数据节点组，以此类推。当该业务增长客户号增加时，分布式数据库集群新增扩容一个数据节点用于存储新增的客户号。

2. 水平按 Hash 拆分

第二类是水平按照客户号的 hash 拆分，每个单元包含多个 shard 分片。方案 2 中 1 个业务单元数据将存储到分布式数据库的一个节点组中；与方案 1 不同点在于，方案 2 的业务数据按照 hash 拆分，拆分成众多的 shard 分片，而 1 个节点组将存储多个 shard 分片。多个业务单元共享一个分布式数据库集群。

以银行账户核心的存贷款服务的个人存款为例，假设目前该业务有 5000 万的客户号，我们预期 1 个单元存储能存储 500 ~ 1000 万左右的用户，所以初步规划 10 个单元，为业务增长预留一倍的空间；我们搭建一个含 10 个数据节点组的分布式数据库集群，而后将 5000 万的客户号通过 hash 算法散列到 10 个数据节点组中，每个数据节点组存储大约 500 万的客户号；当该业务增长客户号增加，每个单元存储的客户号接近 800 万左右时，分布式数据库集群新增扩容一批的数据节点组（例如扩容一倍），而后在业务低峰时将部分客户数据重分布到新增数据节点组。每个数据节点组扩容后存储的数据量基本相当。部署示意图类似图 11。

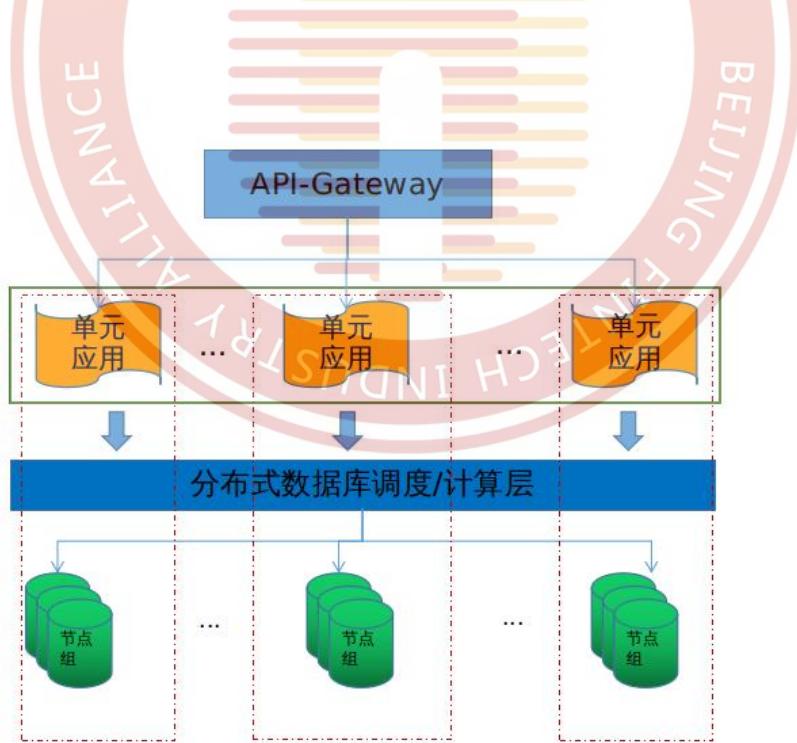


图 11 业务单元存储于数据节点组的部署架构示意图

3. 水平拆分再按 Hash 拆分

第三类是先水平拆分区域，再按照客户号的 hash 拆分，每个单元包含多个 shard 分片。从部署角度，方案 3 与方案 2 等同。1 个单元存储于一个节点组中，一个单元存储多个 shard 分片。多个业务单元共享一个分布式数据库集群。

以银行账户核心的存贷款服务的个人存款为例，假设目前该业务有 5000 万的客户号，先按区域进行拆分，例如华北地区 1000 万用户，而后进行 hash 拆分。整体方案与方案 2 部署类似，主要区别是不同区域所使用的数据节点组是不相交的。

4. 垂直拆分后水平拆分区域

第四类是先垂直拆产品、然后水平拆分区域。如图 12 所示，每个区域单元包含一个产品一个区域的所有数据，对应到一个单独的分布式数据库集群，在集群内可以基于客户号做 hash。即一个单元对应一个分布式数据库集群。

以银行账户核心为例，先按照区域进行划分，例如华北区，然后按业务拆分为存款服务单元、贷款服务单元、现金凭证服务单元和公共账务服务单元等。每个单元部署到一个独立的分布式数据库集群中。

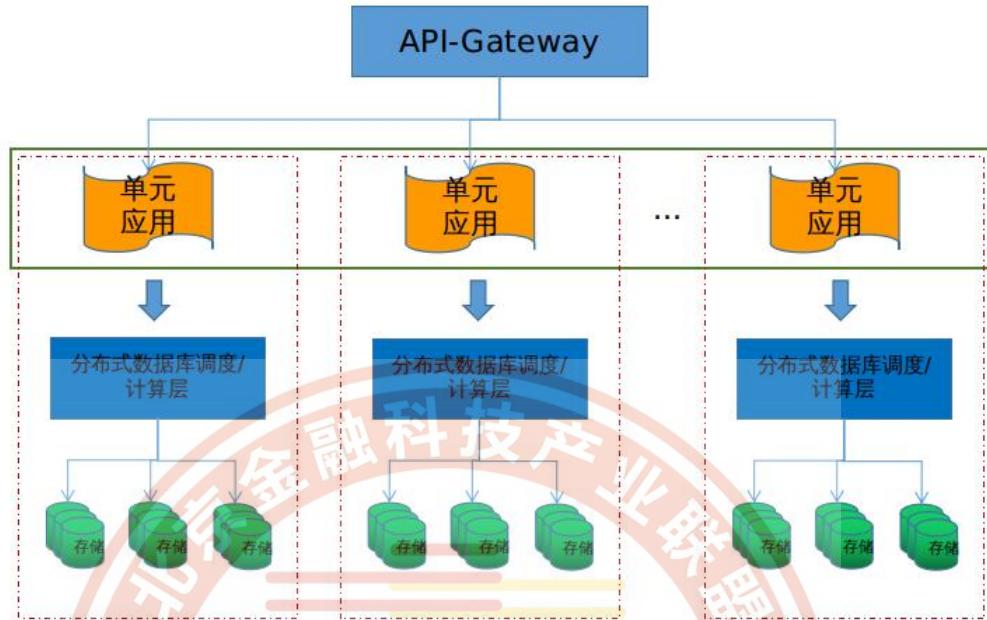


图 12 业务单元独占分布式数据库集群示意图

5. 按区域拆分

第五类是按区域拆分，大单元设计。类似方案 4，每个区域的所有相关产品的所有数据放入一个单元，对应到一个单独的分布式数据库集群，在集群内可以基于客户号做 hash。一个单元对应一个分布式数据库集群。

以银行账户核心为例，按区域进行划分，例如华北区，这个区内是一个完整的账户核心业务，包含存款服务、贷款服务、现金凭证和对公账户服务等，这个区域作为一个大单元部署到一个独立的分布式数据库集群中。

最后针对全局单元，如果数据量不大，可以考虑数据到分布式数据库集群的一个数据节点组；如果数据量较大，可以考虑部

署到一个独立的分布式数据库集群中。

(三) 数据库部署

1. 单元扩容部署方案

单元内的扩容包括了应用扩容和数据库扩容。对于应用节点扩容，单元化架构下，应用服务应该采用无状态设计，这样可以通过应用镜像，快速增加新的应用容器或虚拟节点。对于单元内数据库扩容则分为 2 种情况：小单元场景和大单元场景。

小单元场景，一个单元部署在分布式数据库的一个数据节点组上，单元内的扩容主要通过数据节点垂直扩展实现，即通过数据库节点在线滚动升级的方式逐步将一个数据节点组相关的多个数据库的物理资源进行升配，例如提高 CPU、内存、网络、磁盘 I/O 等。示例如图 13 所示：

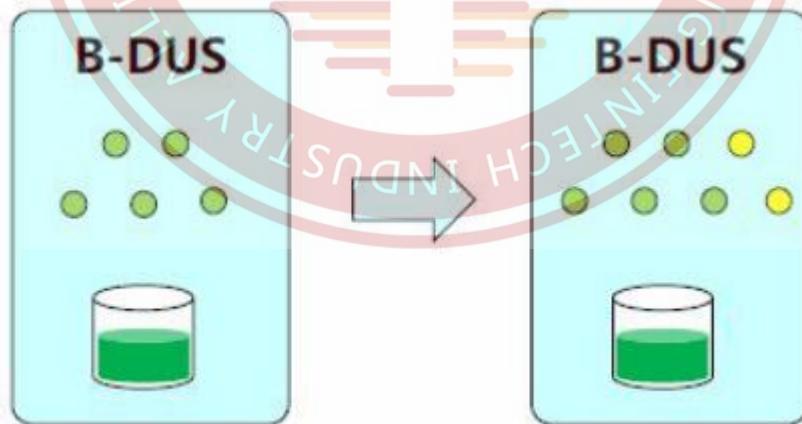


图 13 小单元场景单元内扩容示意图

大单元场景，一个单元部署在一整套分布式数据库集群上，

所以单元内的扩容主要依赖分布式数据库集群的横向扩容实现，通过使用更多的机器，实现分布式数据库在容量与性能吞吐上的提升。分布式数据库能够检测到新的服务器，并将其他服务器上的数据迁移到扩容服务器，尽可能的均衡各个节点的负载，实现整体的平衡，当然这个过程势必需要进行数据迁移，集群会基于负载均衡原则将部分数据迁移到扩容服务器上，这个数据迁移的过程对业务是透明的。示例如图 14 所示：



图 14 大单元场景下单元内分布式数据库扩容示意图

全局单元的数据库扩容主要依赖上述单元内扩容策略，即提升单元硬件升配或依托分布式数据库自身的扩容技术实现，无法依赖新增单元的方式进行扩容。

当单元的存储达到瓶颈，无法通过升级物理节点进行提升已经达到分布式数据库集群有效扩容边界时，就需要进行新增单元扩容，示意如图 15 所示：

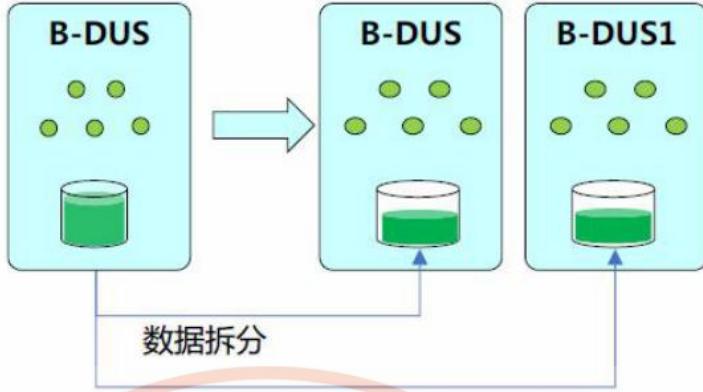


图 15 新增扩容单元示意图

新增单元的扩容方式与业务服务单元的拆分策略紧密相关，本小节将分别讨论 5 种业务单元拆分策略对应的新增单元扩容方法。

1) 先垂直拆产品然后水平按照客户号的 range 做拆分。对于方案 1 而言，单元客户容量上限固定，容量超出时新客户号进新的单元；一个客户号原则上固定属于某一个单元，不会出现客户号在单元间的挪动，所以扩容的基本过程为：当现有的单元快满时，在分布式数据库集群上新增一组数据节点组。当单元客户数达到固定上限，新增的客户号将被路由分配到新的业务服务单元。

2) 先垂直拆产品然后水平按照客户号的 hash 拆分，每个单元包含多个 shard 分片。对于方案 2 而言，每个单元存储的是多个 hash 散列后的 shard 分片。单元客户容量过多时将新增单元并挪动已有单元的 shard 分片到新的单元；shard 是扩容的最小

粒度，shard 的数量一定程度上决定了架构的扩容上限，扩容时将出现客户号在单元间的挪动。所以扩容的基本过程为：当现有的单元快满时，在分布式数据库集群上新增一批数据节点构建多个数据节点组。例如一次性扩容 1 半的数据节点或扩容 1 倍的数据节点。扩容节点数越多，本次扩容所需挪动的数据总量就越大。计算现有单元内需要挪动的 shard 分片，挪动 shard 分片到新增数据节点组，形成一批扩容任务计划。逐步执行扩容任务计划，将业务数据 shard 分片均匀挪动到新增的数据节点组，实现全局数据分布的均衡。这个业务数据挪动的实现可能会涉及相关 shard 分片的锁定，或基于数据库增量变更日志（例如 binlog）来应用数据挪动过程产生的数据变更以确保数据挪动过程的在线。业务数据挪动到新单元后，调整路由映射。为了实现便捷路由管理，一个可行的实现方式为“hash 函数+路由表”的实现，即通过 hash 将业务数据固定散列为一定数量的 shard 分片，而后通过额外的一个路由映射表来实现一个 shard 分片与后端业务服务单元之间的映射关系。数据挪动完成后，只需调整路由映射表中对应 shard 分片的映射关系即可。

3) 先垂直拆产品然后水平拆分区域，再按照客户号的 hash 拆分，每个单元包含多个 shard 分片。方案 3 的扩容过程同方案 2。

4) 先垂直拆产品然后水平拆分区域。对于方案 4 而言，新

增单元需要搭建一个新的分布式数据库集群，所以每次扩容都会涉及大批量数据的重分布。扩容过程整体实施代价较大，与相关分布式数据库的实现也有较大的关联，一个可能的方案是依托数据同步工具，进行单元业务数据全量和增量的搬迁。

5) 直接按区域拆分，大单元设计。方案 5 的扩容过程同方案 4。

2. 高可靠部署方案

1) 本地化高可靠部署方案

如图 16 所示，当某个计算节点发生故障，流量直接切换到存活的计算节点，切换过程应用无感，后续通过计算节点自愈恢复后集群自动发现并重新将该计算节点加入集群。

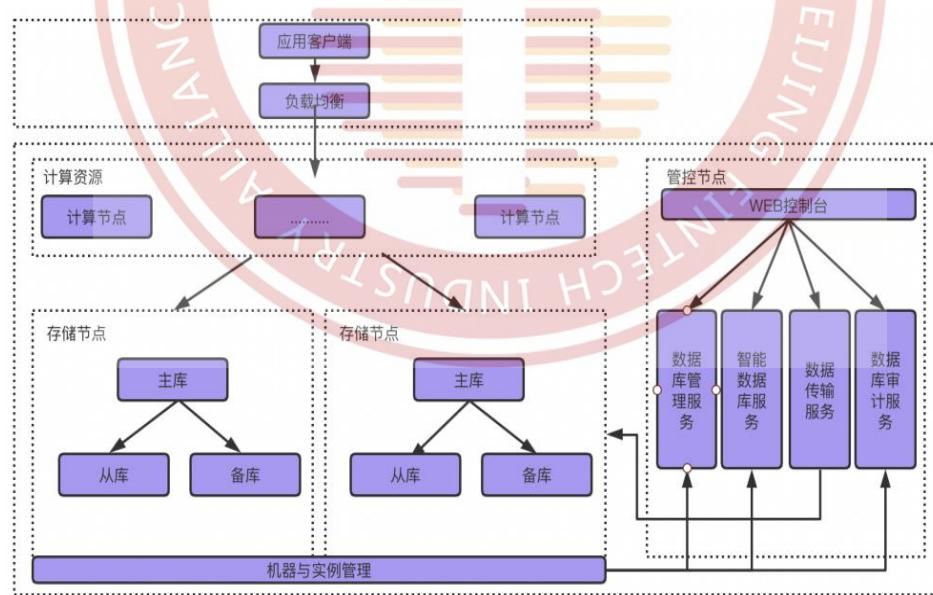


图 16 单元化架构下分布式数据库高可靠部署架构图

单元化架构下分布式数据库数据副本高可靠主要依托分布

式数据库本身的数据高可靠技术实现。目前主流的高可靠技术主要的实现是依托多数派一致性共识算法，例如 paxos 或 raft 算法。

数据节点组(即存储节点)采取多副本方式构成高可用集群，一个节点负责写入，其他节点通过多数派一致性算法保证数据一致性，因为部署在同一机房，集群同步打开强一致性开关。当主库发生故障，系统会自动发现并尝试恢复主库，如果主库无法恢复则发起主从切换，保证数据库高可用。

2) 同城双活部署方案

同城双活是双活技术与同城灾备中心模式结合的一种主流容灾架构。业务系统可以同时通过生产中心和灾备中心进行访问，无需指定特定的访问规则。数据库架构同时兼备异地互备模式的负载均衡和故障自动切换能力，且由于处于同城较近距离，两个数据中心的存储节点可以保持数据强一致。

当其中一个中心发生灾难时，通过接入前端的负载均衡调整，可将全流量输入对等的灾备中心；数据库同时自动进行切换，灾备中心的数据库集群承载全部查询请求。同城双活的部署示意图 17 如下：

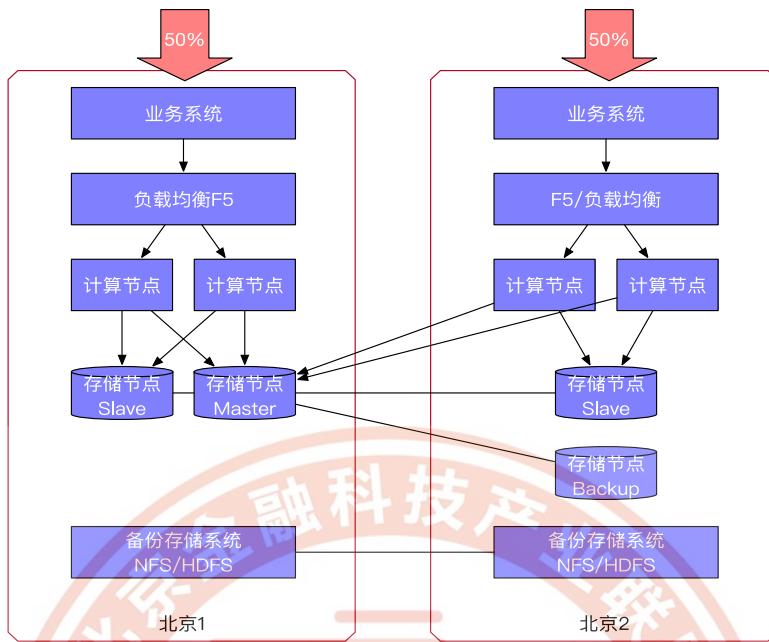


图 17 同城双活部署示意图

基于数据分布式架构可以对应用层提供透明的双活能力。以一个四分片的数据表为例，如图 18 所示，分片数据可以均匀分布在两个中心的数据库数据节点中。

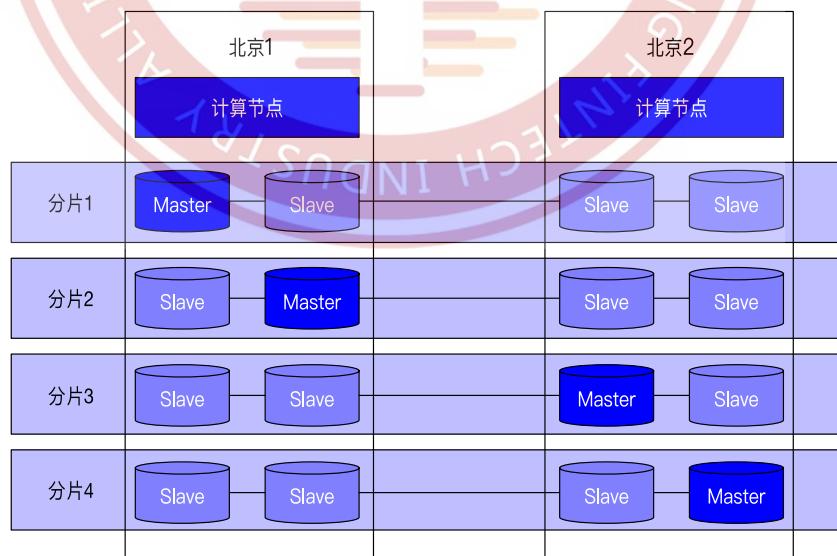


图 18 四分片数据在两个数据库中心分布式示意图

3) 两地三中心部署方案

在同城双活或者同城互备的架构下，再增加一个远距离的容灾中心，可实现两地三中心的容灾架构。如图 19 所示，该架构在同城容灾方案的基础上，获得了对地震、飓风等区域级灾难的抵御能力。由于异地灾备中心距离较远，所以数据同步一般考虑使用异步模式，可基于数据库异步同步功能实现，或者在应用层使用消息队列等组件进行业务数据异步同步，进而实现远距离异地机房的数据最终一致性。

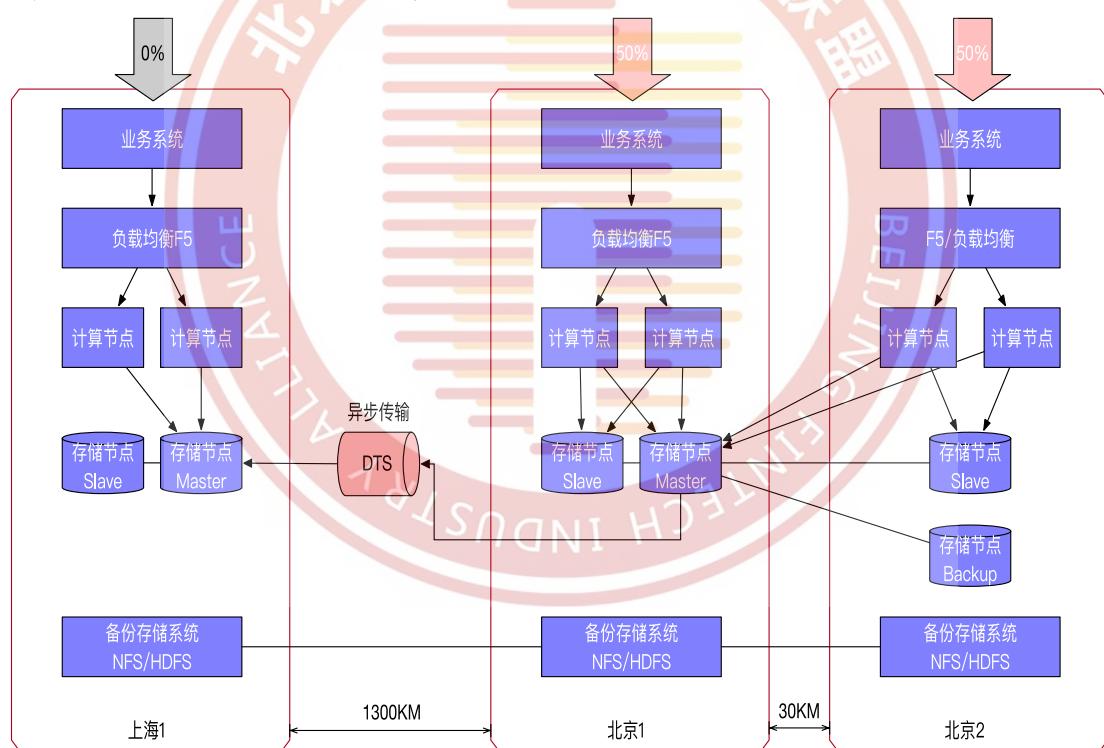


图 19 两地三中心单元化架构示意图

3. 灰度发布部署方案

1) 业务单元灰度发布部署方案

业务服务单元的业务灰度发布部署方案与单元拆分策略紧密相关，本小节将分别讨论 5 种业务单元拆分策略对应的业务灰度发布方案。

先垂直拆产品、然后水平按照客户号的 range 做拆分。对于方案 1 而言，一个单元存储一个特定范围客户号，所以一个简单的灰度发布方案就是构建一个灰度单元，一些受控用于灰度验证的客户号存储于该单元，例如将银行职员的客户号存储到灰度单元中。每次新版本应用发布时将先在该灰度单元进行发布，验证没有问题后再发布到其他业务服务单元。由于方案 1 是小单元设计并且具有优异的故障可用性，灰度单元故障不会影响其他业务服务单元的运行。同时用于灰度验证的都是受控的客户号，整个灰度发布过程更加的可控。

先垂直拆产品、然后水平按照客户号的 hash 拆分，每个单元包含多个 shard 分片。不同于方案 1，方案 2 采用 hash 进行客户号拆分，这样就难以确保一组用于灰度验证的客户号可以固定被散列到一个灰度服务单元中。对任意一个单元进行灰度发布验证，由于方案 2 较好的故障可用性以及方案 2 的小单元设计，灰度单元故障不会影响其他业务服务单元的运行。但由于无法确保灰度发布影响的用户是受控用户，所以灰度发布过程对客户的影响通常也更大。一个可以考虑的折中方案为在 hash 拆分的基础上，对客户号做一个白名单，所有受控用户都放入白名单，并在

进行 hash 前先被路由到一个灰度单元中，用于后续恢复发布验证。

先垂直拆产品、然后水平拆分区域，再按照客户号的 hash 拆分，每个单元包含多个 shard 分片。方案 3 的业务灰度发布方案同方案 2。

先垂直拆产品、然后水平拆分区域。对于方案 4 而言，由于是大单元设计，灰度发布过程的业务模块风险较高，恢复发布所影响的客户范围也难以控制。方案 4 如果采用方案 2 的灰度单元策略，主要的风险点在于灰度单元的数据规模远小于正常业务服务单元，量变可能引起质变，在该灰度单元的验证结果存在与大单元上行为不一致的风险。

直接按区域拆分，大单元设计。相比方案 4，方案 5 由于是多个产品使用同一个单元，业务的灰度发布过程的复杂度也明显增加。

2) 分布式数据库集群灰度发布部署方案

分布式数据库集群的灰度发布将主要由分布式数据库集群自身技术实现，需要确保分布式数据库集群的所有组件升级版本的向下兼容性，并且做到可升可降。

对于计算节点，由于数据库集群需要支持计算节点的动态增减，通过剔除老版本计算节点并加入新版本计算节点的方式，实现计算节点的滚动轮替升级。需要考量的风险点就是不同版本的

计算节点在同一个数据库集群内部运行过程的稳定性，所以计算节点新版本必须向下兼容。

对于存储层的数据节点，通过数据节点组的在线滚动升级，来实现数据节点的新版本升级。基本过程就是先新增一个新版本的数据节点，该数据节点基于节点组主数据节点的副本进行构建，而后剔除一个老版本数据节点，然后重复前面的步骤，直到所有的备数据节点完成新版本升级，最后进行主备切换并对原主节点进行升级。

3) 灰度发布部署过程的管理

灰度发布过程应该循序渐进，控制发布风险影响的范围，参考《新一代银行 IT 架构》，可参考的管理方案如下：首先针对一个子系统的一个实例进行灰度发布，控制影响范围为一台主机，如果出现问题，可以通过停止该主机触发高可靠 failover 机制来快速解决故障。其次针对灰度单元进行灰度发布，将一整个灰度业务服务单元进行发布。发布过程密切关注业务层面与数据库层监控指标是否正常；如果出现异常，进行快速回滚；如果正常需观察一段时间验证发布效果。最后针对普通业务服务单元进行发布。发布过程考虑分批次发布方式，每一批次的发布也是一次灰度发布，都需要预留足够的观察时间。对于低风险变更，灰度发布的批次间隔可以考虑为大于 1 小时；对于中高风险的变更，灰度发布的批次间隔可以考虑为大于 12 小时。

4. 数据汇总聚合部署方案

1) 数据聚合需求

单元化架构需要考虑单元内所有的业务需求在单元内部实现，即单元的业务功能要实现自包含，然而还是有些场景需要对所有单元进行汇总和分析的，其中逻辑较重的需求放到 Hadoop 为主的大数据平台完成，另外把简单或者有一定时效要求的分析需求放到数据聚合库，图 20 为单元化数据聚合典型的逻辑架构。

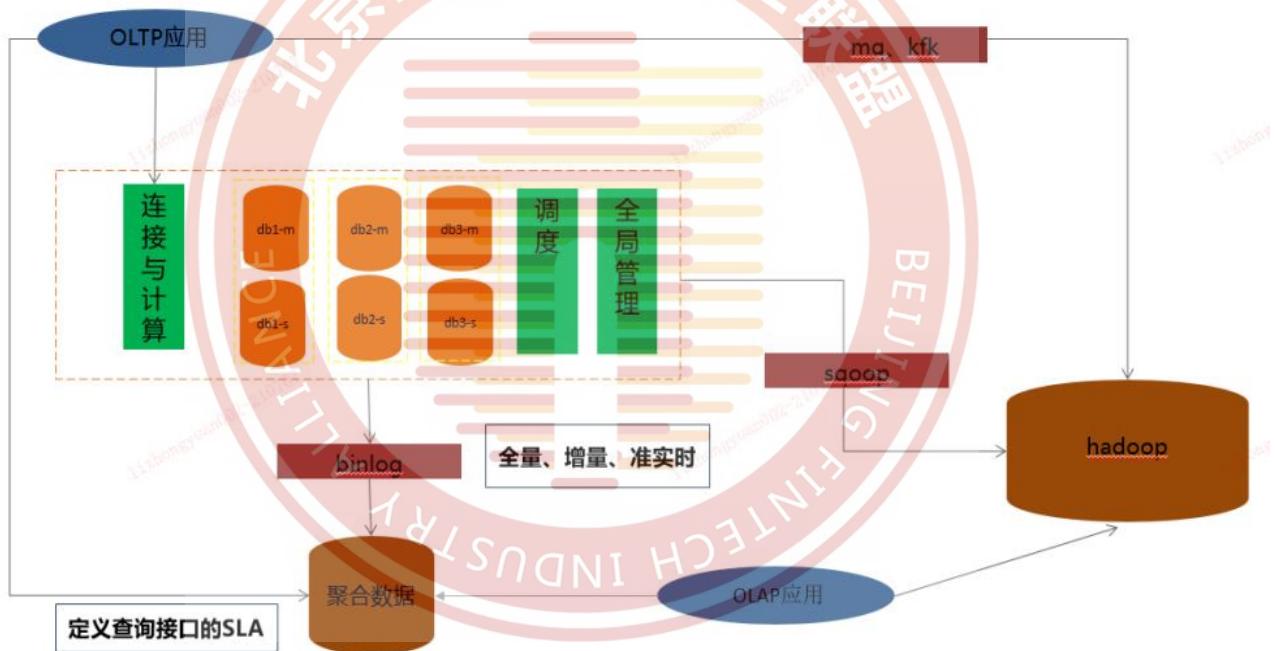


图 20 单元化数据聚合典型的逻辑架构示意图

在部署时，单元化业务数据库采用典型的两地三中心的高可用架构，而数据聚合库由于数据可再生，且不涉及客户业务，因此部署时基本采用单数据中心的模式，以节省资源。同时为了保证在极端情况下数据中心故障不影响相关的功能，所有使用数据

聚合的应用均需要有降级方案，降级方案可以选择降级到业务系统的备库或者大数据平台。图 21 所示为数据聚合库的部署架构。

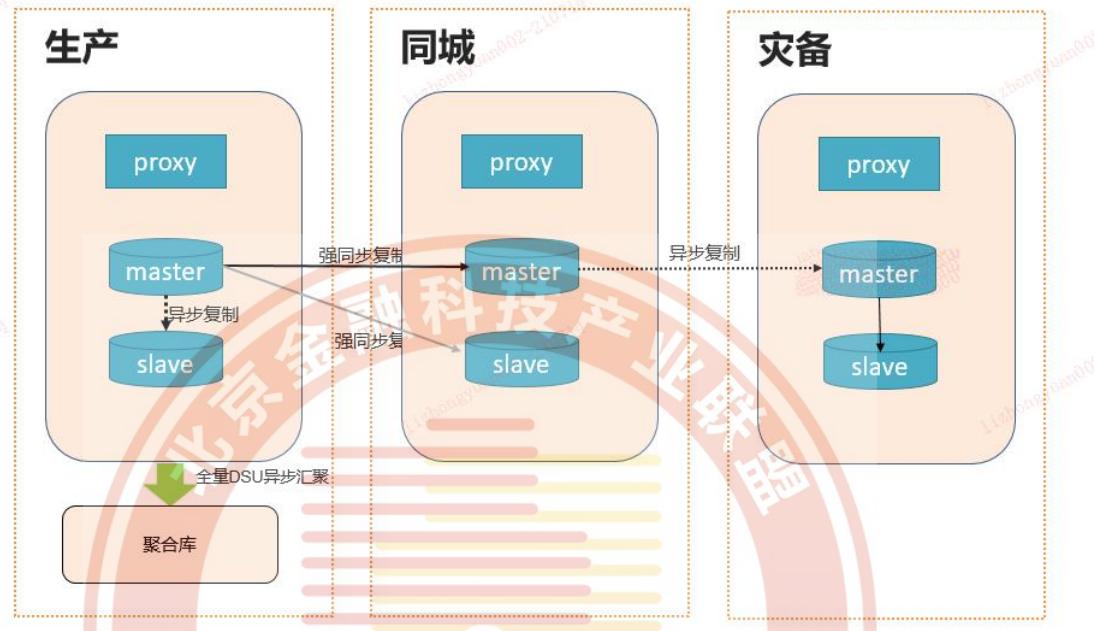


图 21 单元化架构数据聚合部署架构示意图

2) 数据聚合技术方案

数据聚合采用生产者和消费者的模式，生产者负责进行数据库增量变更日志解析（例如 binlog），以消息的方式进行传递，由下游的消费者进行相应的消息读取并写入到下游目标端数据库。

针对不同的业务单元与分布式数据库部署对应方案，数据聚合的技术方案也有差异：针对小单元部署架构，即一个单元为分布式数据库的一个数据节点组，单元的增量变更主要是获取该数据节点组某一副本（例如主副本）的增量变更日志（例如 mysql

内核数据库单数据节点的本地 binlog）。因为单元与单元之间的独立性，不同单元的增量变更日志抽取也相对独立，实现较为简单。针对大单元部署架构，即一个单元对应一个独立的分布式数据库集群，单元的增量变更需要获取整个集群的增量变更日志。可选的实现包括：分布式数据库集群原生提供 CDC 模块，例如一个全局 binlog server。由 CDC 模块负责处理分布式数据库集群数据节点的增量变更日志信息的汇总、排序、归并、去重等，最终统一对外输出一份完整统一的全局增量变更日志。通过外部第 3 方工具抽取分布式数据库集群后端数据节点的增量变更信息，而后按照一定的规则进行排序、归并、去重，需要考虑处理的场景 包括了全局分布表的去重、DDL 语句的合并与同步、分布式事务信息的合并、分布式事务的排序等等。当然如果目标端消费集群只要求数据最终一致性的话，可以不考虑分布式事务信息的合并与排序。如图 22 所示为分布式数据库集群后端数据节点增量变更抽取同步架构示意图。

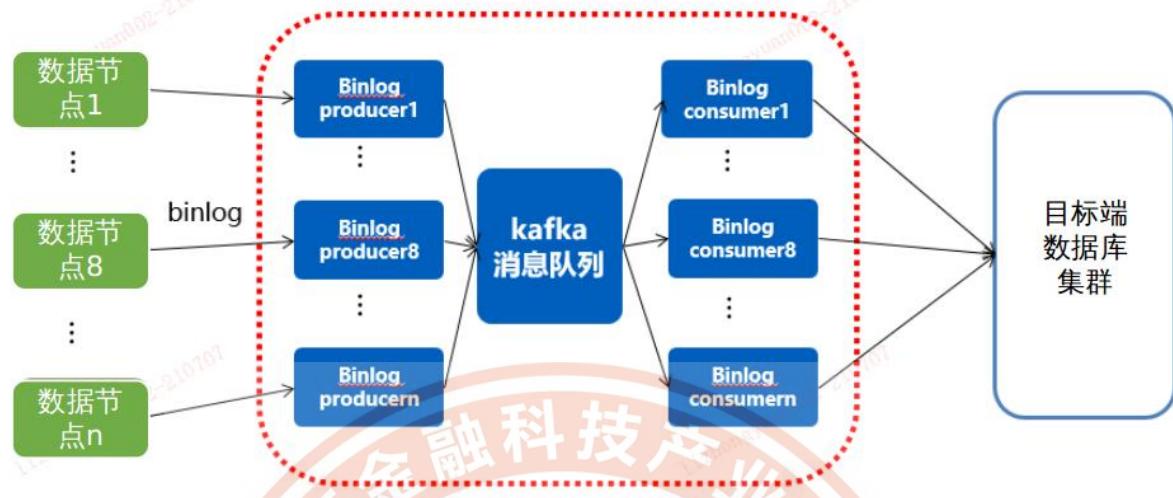


图 22 分布式数据库集群后端数据节点增量变更抽取同步架构示意图

3) 数据聚合挑战

聚合库的容量和性能问题。容量压力主要来自源端通过单元化可以无限增加，而聚合库是否也能达到同样的扩展能力，这就需要一种可扩展的数据库，能够支持足够大的容量。除了容量以外，大数据量查询的性能也往往是一个很大的问题，之所以要进行单元化，往往是因为单个数据库无法支持太大的数据量。这里面有可能需要应用配合进行一定的改造才能达到最优的效果。

数据聚合的时效问题。由于数据量和中间过程涉及多个组件的配合，数据同步时效往往无法做到实时，甚至有时候准实时也是奢望，比如在业务高峰或执行批量时，同步出现异常的时候。因此限于时效的考虑，适合访问聚合数据的业务场景可能会减小一些。

数据聚合的数据核对。动态的数据核对其实是一个非常复杂

的问题，尤其是在大数据量的情况下，动态核对大表甚至可以说是灾难。目前比较可行的数据核对方案主要有全量数据核对、数据切片核对、带时间戳的数据核对等。

数据聚合应急问题。当数据聚合出现异常或者同步时效无法达到目标时效时，应用如何应急是一个需要解决的问题，建议访问聚合数据的应用要有适当的降级方案。

5. 应用备份恢复部署方案

针对不同的单元与分布式数据库部署对应关系，所需采用的备份恢复方案也不相同。

1) 对于小单元部署，即一个单元部署在一个数据节点组，单元数据的备份只需针对该数据节点组进行备份。以 MySQL 内核数据节点分布式数据库为例，可以使用 xtrabackup 物理备份工具对数据节点组的主库或备库进行全量与增量的备份，同时对数据节点组的增量变更日志也同步进行备份（例如 binlog 备份）。除物理备份外，逻辑的关键数据正文备份也是常用的备份策略。

由于单元与单元数据的隔离性，不同单元的数据备份可以完全独立，备份文件的管理也可以独立分别进行。每个单元的数据还原只依赖每个单元自己的备份文件，所以并没有全局一致性的备份需求。对于不含拆分字段的数据以副本形式保留在每个业务单元中的场景，在还原后需要考虑更新为最新数据。还原主要应用的场景包括：灾备场景，用于恢复单元数据。容忍一部分的数

据丢失，丢失数据量大小取决于增量变更日志的备份周期。单元数据节点组扩容或节点替换场景，基于备份恢复迅速初始化一个新的数据节点。历史数据的获取，通过基于时间点还原，读取历史版本数据。其他，例如构建测试环境。

2) 对于大单元部署，即一个单元部署在一个分布式数据库集群，单元数据的备份恢复依赖分布式数据库的备份恢复实现。不同于小单元场景，分布式数据库集群需要提供全局一致性的备份恢复功能，即备份还原的数据不存在部分提交的分布式事务信息、并且还原的集群相当于当前集群的一个历史快照，可以基于集群的全局增量变更日志进行增量变更的同步。备份任务的执行将依托 CMDB 库并且由自动化运维管理平台进行集中式的调度管理。同时建议备份任务走备份私网，避开业务网络，避免备份对业务流量的影响。

(四) 数据库运维要求

1. 基本运维需求

随着单元化架构与分布式数据库技术的引入，银行 IT 系统的运维工作量是传统架构的几倍甚至几十倍的提升。本章节将更多侧重分布式数据库在单元化架构下的运维解决方案，并且本章内容参考了《新一代银行 IT 架构》。单元化架构下对分布式数据库应用运维主要的需求包括但不限于：

高效合理的资源分配。由于单元化架构下，特别是小单元设

计方案，单元的升配、新增与集群节点的扩容是常见操作。如何高效统筹全局、缩短资源环境准备周期、提高资源控制精细度以减少浪费是主要的运维挑战之一。

高可控的灰度发布管理。根据业界经验，在一般系统故障中，有超过 70% 是由发布变更引起的，而该风险在单元化架构下进一步提升。如何降低灰度发布的危险影响范围、将改动的风险份數与缩小，以保障系统整体的可靠性是主要的运维挑战之一。

高效自动化的运维管理。依托分布式架构，从效率、安全、连接等多个维度构建分布式高效执行与分发、自动化调度与执行结果合并、以及便捷系统联动的自动化运维管理平台是单元化架构下分布式数据库应用运维的核心工作，也是主要挑战之一。

高覆盖全链路的监控告警。在单元化与分布式数据库结合的技术架构下，业务全链路所涉及的节点与组件显著增加，全方位的链路监控告警体系的建设是安全生产和链路性能分析的基础，也是主要挑战之一。

2. 运维关键技术

为了应对上文论述的运维挑战，单元化架构下分布式数据库应用的典型运维解决方案需要重点考虑如下运维关键技术的建设：

1) CMDB 配置管理数据库

CMDB 是所有运维工具的基础，存储并集中管理全局物理层、

逻辑层和应用层的配置项及关系信息。同时通过丰富的对外 API 接口，与其他关键技术组件进行融合，支撑运维管理工具以及流程运作中对于配置信息的使用。CMDB 具备如下特性：

配置模型的动态扩展能力。配置项、属性及关系、属性数据类型、唯一性，组合关键字等均可动态定义与使用。

配置查询灵活多样。支持自动以多配置的在线关联查询。

细粒度的权限管理，同时支持权限的在线配置调整。

开放友好的 API 服务，可以方便地进行调度与集成。

版本管理。支持多维度数据变迁历史的查询、支持配置数据版本基线比对、支持版本回退。

闭环管理。系统设计的部署架构需要存储进 CMDB、业务的部署过程的发起需要基于 CMDB 中的部署架构信息、资源的分配需要基于 CMDB 全局资源信息进行、资源的准备情况与部署信息也需要存储进 CMDB、资源的监控需要基于 CMDB 的部署信息、最后系统下线资源回收需要基于 CMDB 并将最终结果记录到 CMDB。

2) 智能资源分配管理系统。智能资源分配管理系统基于 CMDB 数据库中的资源信息，根据预设规则对全行主机资源进行管理和分配，目标是大幅提高生产与测试环境搭建效率、缩短资源分配与准备周期、严格落实高可用规则、同时提高资源利用率。

首先单元化架构下分布式数据库应用的资源分配必须严格遵守高可用部署规则，对于一个数据节点组而言，同城至少要有

2份副本，异地至少有一份副本。对于重要子系统，数据节点组在主机房至少部署2份副本。一个应用子系统的多个实例，分布在2个或2个以上不同的物理机上。在单个数据中心，一个数据节点组分布式在2个或2个以上不同的机柜上，不同的应用域不共享物理机。同时基于CMDB满足应用本身希望满足的特殊规则，例如子系统互斥。

其次支持丰富的IAAS资源供给，可以按需提供物理机、虚拟机和容器的资源分配。基于CMDB的全局资源使用状态信息和应用的部署需求，高效地从全局资源中挑选满足规则的合适的资源进行交付。

再次具备资源的创建能力，即当资源不足时，可以通过外部的资源创建平台实现资源的实时动态创建。例如当虚拟机资源池资源不足时，可以根据母机资源情况，动态创建新的虚拟机资源。

最后是现网扫描巡检能力，自动识别不符合当前规则的应用部署，提供整改建议。

3) 自动化运维平台

自动化运维平台实现了单元化架构下分布式数据库部署维护的全生命周期管理。相比传统架构下的自动运维管理平台，面向单元化与分布式数据库结合架构下的运维管理平台需要面对大量服务器节点场景下的标准化与批量化的运维压力，所以需要着重建设如下能力：

脚本管理与场景化脚本运维能力。由于单元化+分布式架构下，集群节点数显著增加，存在大量需要基于脚本进行统一批量执行的运维操作，例如所有节点的一次配置变更。所以平台针对脚本的使用需要支持脚本的管理与执行、允许脚本在已授权的一台或多台机器上运行、支持针对场景的预定义脚本管理、支持脚本的执行调度管理（例如前后顺序、串行或并行），最后执行性能的扩展性，即针对大规模节点的并发执行效率应该与节点的规模相关度不大。

发布管理。支持对灰度发布的调度管理、例如灰度发布的顺序、间隔、并行力度等；支持差异化的变量配置，可以便捷地定义与使用差异化规则。支持版本管理与回滚，对于一个子系统，自动运维平台应该记录所有主机的应用与数据库节点版本和发布时间线，支持快速高效的回退。

支持分布式架构下的数据库操作结果合并能力。针对小单元架构，运维管理平台需要具备跨单元分布式数据库节点数据的查询合并能力，将各个单元的查询结果合并提供一个完成的查询数据。

支持风险识别与自动备份。特别通过运维管理平台进行涉及数据的高风险操作时，运维管理平台可以自动调用分布式数据库的备份能力，进行数据备份，并按需快速还原。

分布式的定时任务管理。定时任务可以根据管理员定义的固

定时间或间隔周期来针对一个给定范围执行一项任意的任务。任务的执行需要避免多次执行、超时执行和漏执行。针对单元的数据库备份任务就是典型的分布式定时任务。

分布式架构下的文件存储与分发。无论是应用/数据库版本发布还是日常运维，都会涉及文件（例如安装包、脚本）的快速分发（1台或多台主机），同时针对跨中心的场景应用具备就近分发能力，分发过程可以基于最近的存储节点或副本进行分发。

4) 链路监控告警系统

单元化+分布式数据库架构场景下，业务请求调用链路的复杂度显著增加，链路监控告警系统需要覆盖单元化业务到分布式数据库的全链路组件的端到端的监控告警能力。具体如下：

全覆盖监控。提供全链路基础架构、公共平台、中间件、应用、数据库的监控与告警能力，提供联机业务指标和跑批场景指标的监控。

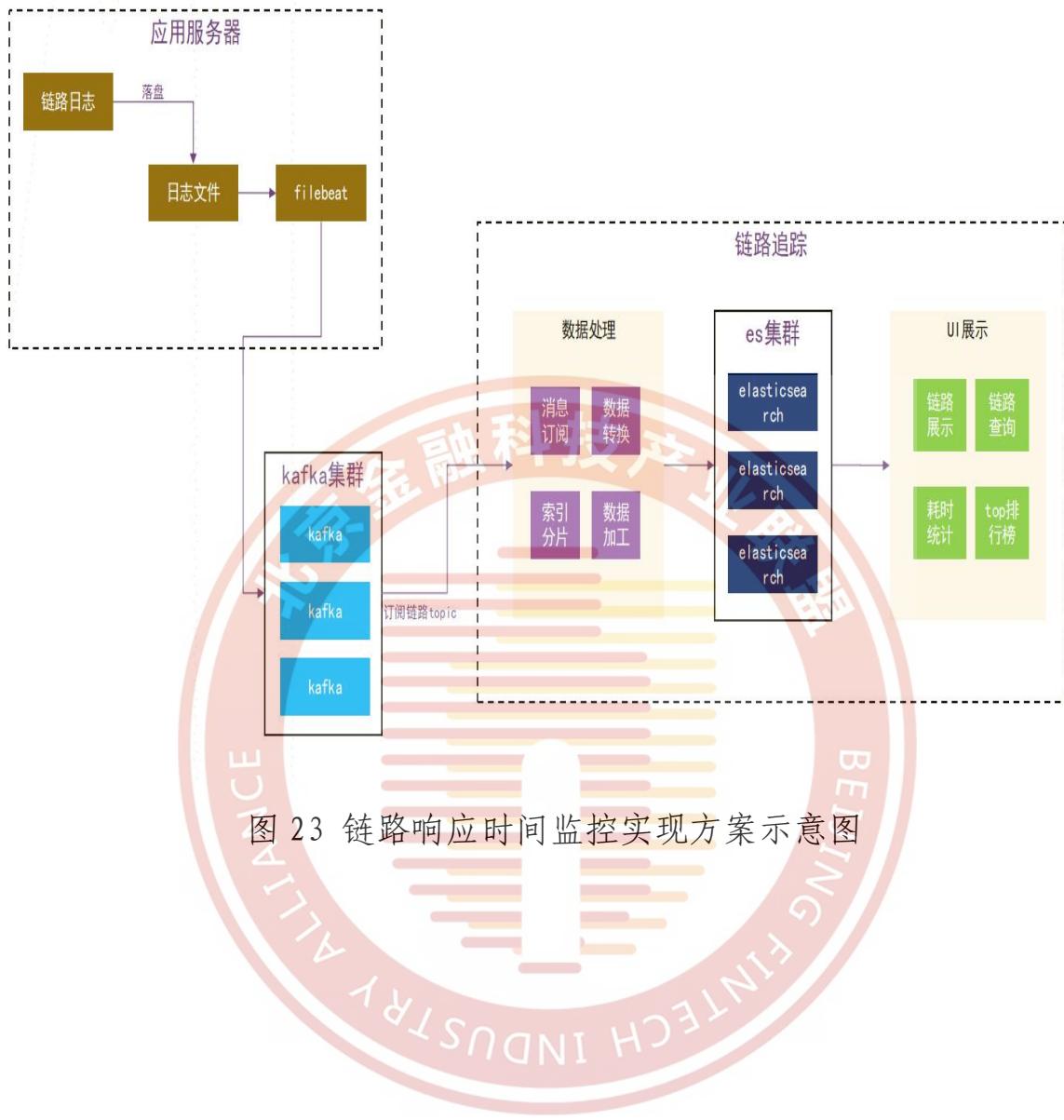
模板化监控配置。提供模板化的监控采集配置，基于策略的阈值配置以及个性化监控告警通知规则，同时可以实现基于CMDB部署信息的监控配置自动添加。

个性化监控视图。建立多维度监控视图、可自定义监控展现面板。建立端到端的交易场景监控与跑批的全局监控视图。

灵活的监控告警上报与存储。提供灵活的监控指标上报和告警信息上报接口，支持指标和告警历史值的查询与导出，并可以

按需设定存储保留策略。

对于单元化+分布式数据库架构下的链路监控，其中一个非常关键的功能就是全链路各环节响应时间的监控采集与分析汇总。即可以从全局层宏观的查看各个环境的响应时间均值情况，也可以针对特定慢交易查看具体慢在哪个环节。如图 23 所示是一个链路响应时间监控的实现方案示例。系统链路追踪通过链路日志收集模块收集微服务调用信息并输出到磁盘的日志文件中，并通过 filebeat 实时传输到 kafka 集群中。通过订阅 kafka 日志 topic 的程序，基于 SpanId（当前这次调用的 ID）、TraceId（一次请求会生成一次全局的请求 ID，跨节点往下传递）进行解析、过滤后传输给 ES 集群，以提供检索、分析功能。最后数据可视化展示。链路信息输出点位于网关、业务接收、业务发送、数据，每一次远程调用请求生成一个 Span。Span 是请求中的基本工作单元，用于记录调用中的详细信息（如时间、调用 ip 地址和端口、接口名称、接口参数、响应参数等），需要通过代码调整在网管层、业务层和数据层进行 span 信息的采集与文件输出，平台根据输出的 span 能拼出一条完整的请求链路。



四、典型案例

(一) 建设银行案例

1. 应用背景

建设银行客户规模和日均交易量达数亿级别，当前主机平台已逐步显现处理性能压力。且随着信息化技术的快速发展，业务规模仍将进一步扩大，传统集中式架构达到扩容瓶颈，无法满足未来业务发展需要；另一面，当前金融业核心系统强依赖于国外软硬件，不满足国家安全战略要求。因此，行内核心系统由主机下移到开放平台势在必行。为实现以上目标，建设银行开展了核心系统下移至国产分布式数据库相关工作。

但目前国内大规模、复杂的核心业务下移至国产分布式数据库的方案还未达到成熟可借鉴的程度，国产软硬件产品也需要时间检验产品成熟度，因此整个工程大胆探索，采用大量开创性技术，通过应用单元化和分布式数据库底座，实现核心系统向分布式平台平稳下移。下面详细介绍。

2. 应用方案

建设银行分布式系统总体技术架构如图 24，包含应用路由、服务集成代理、应用组件集群、配置中心、分布式数据库等主要组件。其中分布式数据库使用 GoldenDB 分布式数据库。如图 24 所示，各组件构成有机整体，实现大型系统灵活路由和高并发处理能力。具体如下：

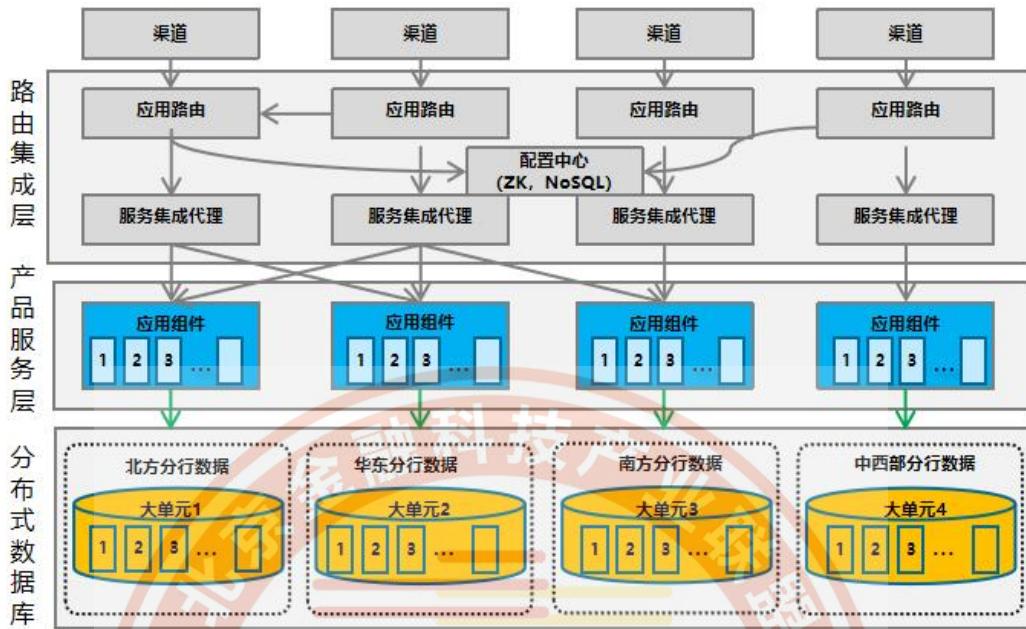


图 24 分布式系统总体技术架构

应用集成层：应用集成层包含应用路由、服务集成代理两大组件。

应用路由：具备横向转发能力，交易请求发往应用集成层，首先会就近进入当地的应用路由。应用路由通过配置中心的 API 接口，确认交易请求归属的 Region，如不属于本地 Region，则将请求转发至交易归属 Region 的应用路由。对应 Region 的应用路接收到请求后会将请求转发给与其同 Region 的服务集成代理。

服务集成代理：服务集成代理具备跨系统事务一致性处理能力，当一笔交易涉及多个服务子系统，如涉及跨 Region 的转账

交易时，服务集成代理会将单笔转账交易拆解成一笔转出交易和一笔转入交易，每笔交易都会同时准备好相应的冲正交易数据。此外，在系统过程投产过程中，一支业务的部分数据还在主机平台，部分数据在开放平台，这种情况也由服务集成代理跨平台协调一致。

产品服务层：产品服务层提供银行业务的具体服务，包含产品服务，是整个信息系统的重要组成，目前运行在三个平台上：主机平台、C 平台和 JAVA 平台。其中主机平台上的产品服务会全部下移到开放平台（C/JAVA）上。开放平台上引入 GoldenDB 分布式数据库，承载原来主机平台上的数据库服务。数据访问代理作为产品服务和数据库之间的适配层，针对 GoldenDB 等多种数据库进行适配和统一访问。

配置中心：负责提供应用路由和服务集成代理所需要的定位信息。配置中心包含客户端和服务端两部分，客户端通过 SDK 的方式嵌入在应用路由和服务集成代理组件内，服务端基于 ZK、NoSQL 缓存等方式开放相应的路由数据查询服务。

GoldenDB 分布式数据库提供大规模数据存储和高并发数据库访问能力。GoldenDB 实现数据多分片、弹性扩容和分布式事务强一致读写，通过数据多副本、快同步、分组管理、备份恢复等技术实现大规模数据库的高可用、高可靠，以及多地多中心容灾能力。

数据切分与路由方案：业务单元划分综合考虑业务负载均衡、数据量负载均衡以及减少跨区操作等因素，通过数据的水平拆分和垂直拆分两种方式，将业务划分为四个 SPU 大逻辑单元，近百个数据分片。

业务请求通过渠道层后，进入应用集成层。首先会就近进入本地应用路由，本地应用路由解析出请求中的业务类型以及 ID 信息，通过访问配置中心的路由数据，确认交易请求归属的业务单元，如不属于本地业务单元，则将请求转发至交易归属业务单元的应用路由。对应业务单元的应用路由接收到请求后会将请求转发给与其同一单元内的服务集成代理。

①	交易码	路由识别策略
	A002811001	卡号
	A002811002	账号
	A002811003	手机号

②	识别策略	路由字段	应用分区号 (五位, 取客户号部分长度)
	卡号	4367450200000001	15001 (客户号A的五位长度)
	账号	2373450200000001	29002 (客户号B的五位长度)
	手机号	13810201982	31003 (客户号C的五位长度)

③	应用分区号	服务处理单元	服务地址
	01000~01999	Region1_SPU_1	AP1_1/AP1_2/AP1_3/AP1_4/...
	02000~02999	Region1_SPU_1	AP1_1/AP1_2/AP1_3/AP1_4/...
	03000~03999	Region2_SPU_1	AP1_1/AP1_2/AP1_3/AP1_4/...

图 25 数据路由规则构成

如图 25 所示具体来说，前端渠道送到平台的请求，根据请求路由的交易码，确定路由的类型，支持卡号、账号和手机号等，通过对对应的号码信息查询到所在应用分区号，再根据应用分区号可以快速确定对应的服务单元地址。应用层的服务单元数量较多，通过众多节点分担联机处理压力，满足高并发处理性能要求。而在数据库层，则使用大单元机制，在一套 GoldenDB 集群下创建四个租户集群，分别部署在四个数据中心。由于数据库层采用大单元设计，同一数据中心内的请求均由同一数据库租户处理，无需应用层协调分布式事务，这显著降低跨服务集成代理跨子系统处理的业务吞吐量，减少资源压力。

采用大单元设计的好处是应用层和数据库层扩容解耦。当数据容量不足时，数据扩容仅发生在数据库层，路由层无需任何配置改变；而应用处理性能不足时，增加对应数据中心的应用服务节点，路由层的应用路由和配置中心仅需要关注业务类型信息即可，如图 26 所示。

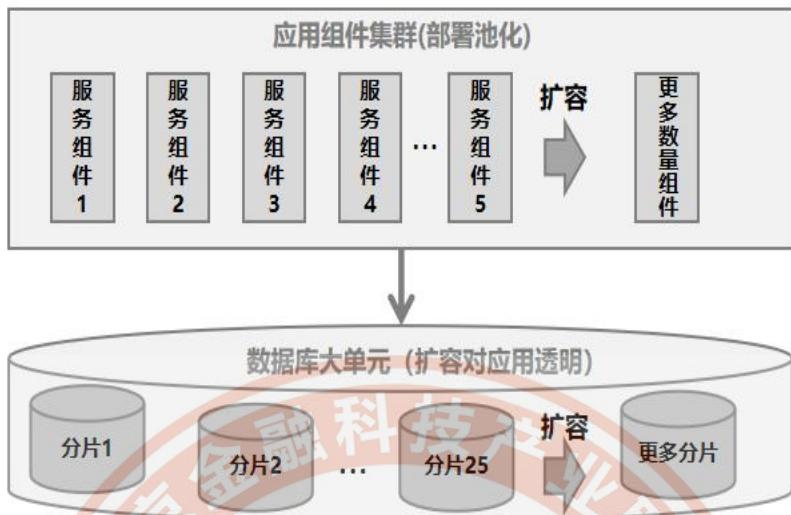


图 26 大单元设计实现应用与数据库解耦

其次是简化路由配置的粒度，不需要基于分行粒度或分片粒度进行路由设计，路由粒度变为四个大单元级，管理更加简单，减少了配置管理的复杂度。

跨数据中心事务由服务集成带来实现。服务集成代理具备处理跨系统子事务的一致性能力，将数据中心请求拆解成一个或多个服务请求。如跨数据中心转账交易时，服务集成代理会将单笔转账交易拆解成一笔转出交易和一笔转入交易，每笔交易都会同时准备好相应的冲正交易数据。执行正常，等待各子系统完成交易操作；执行失败，则会调用该交易的冲正交易，确保转账涉及的各方账户数据恢复到交易之前状态。而大单元内部的跨行操作由 GoldenDB 分布式事务来保障。GoldenDB 通过全局事务节点来管理分布式事务的生命周期，实现数据的实时强一致。

3. 应用成果

对私核心系统里程碑节点：

2021.03 完成国内对私业务的大型分布式核心系统在生产环境部署，实现与主机系统并网运行，交易双发。

2021.07 完成海外核心在生产环境部署，实现与主机系统并网运行，交易双发。

2021.12 青海、宁夏两家分行的对私业务从并网状态切换到开放平台，完成两省投产。

试点业务非功能指标。下移开放平台后，平均交易时长控制在 50ms 左右，接近主机平台处理时延；核心批处理时长缩短 1 个小时；压力峰值大大减轻，在 3 万 TPS 下仍保持低资源占用，达到建设银行对服务质量以及未来业务增长的处理性能要求。

当前已完成两个省份投产，正在逐步将其他省分行从并网状态切换到投产状态，并加大主机平台剩余系统向分布式平台迁移的步伐。

单元化架构提升了系统的横向扩展能力和系统整体可用性。但单元化开发对于应用存在一定的侵入，在新系统设计时需要认真分析单元拆分的逻辑，减少跨单元访问。建设银行采用大单元设计，将核心系统数据拆分为四个大单元，单元内的跨分片事务处理交由分布式数据库处理，大单元之间的事务交由服务集成代理处理，有效减少了在下移改造过程中对原有系统的逻辑入侵，

最大程度地保持了系统的稳定性，实现了应用路由层、产品服务层和分布式数据库层的解耦，便于各层独立扩容和管理。

（二）平安银行案例

1. 应用背景

平安银行信用卡老的核心系统采用的是 IBM 大型机 AS400 架构，由于是国外成熟技术，早期对平安银行信用卡业务的起到很好的推动作用。但是随着业务的发展，系统技术升级困难、软硬件成本过高、系统扩容难度大、业务发展不灵活、新功能实现周期过长等问题愈发凸显。

2018 年平安银行启动信用卡核心系统的重构选型，经过多轮的对比，最终决策以高扩展、高弹性、业务自主可控为主要的架构重构目标，选择分布式单元化架构去 IBM 大型机，数据库使用 TDSQL，预期实现至少支持 10 倍并发业务量，支撑 3.3 倍发卡量，批量处理性能提升 3 倍，并且实现业务系统完全自主可控的目标。

2. 建设方案

1) 单元化业务总体架构

平安银行单元化架构基于量身打造的分布式 PaaS 平台如图 27，应用在进行单元化分布式改造时尽可能只关注业务逻辑的实现，而平台架构、数据库和各种工具交给 PaaS 层。



图 27 平安银行 PaaS 平台

平安银行信用卡业务的单元化 DSU 架构总体如图 28 所示，总体上我们的 DSU (Distributed Service Unit) 架构包括 CS 区 (Common Service)、CM 区 (Central Management) 和 DSU 区，其中 CS 区和 CM 区为非单元化区，DSU 区为单元化区。CS 区是提供公共服务，主要包括配置中心、监控中心、消息中心、注册中心、发布中心等；CM 区是提供公共管理功能，主要进行参数管理、运营管理、经营报表服务。DSU 区是按照客户纬度进行单元化拆分后的业务数据区，单元的基本要求是业务自包含，即客户的业务在单元内完成，同一个客户无须跨单元访问相关数据。

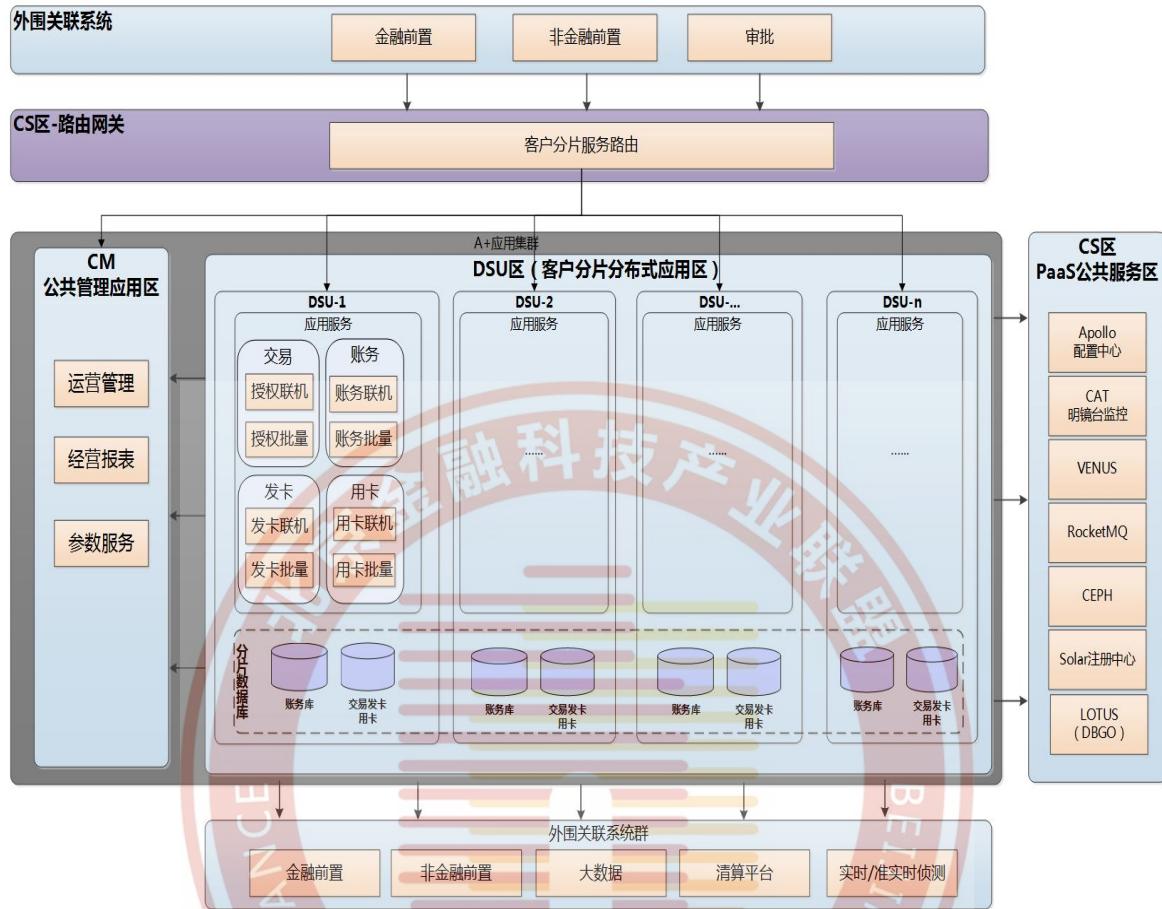


图 28 平安银行信用卡业务的单元化架构图

2) 业务单元的数据切分与路由方案

单元分片的路由逻辑如图 29, 是根据卡号、客户号、银行账号进行映射, 每个客户会根据卡账号的映射得到一个唯一的 PID 号, 根据 PID 进行 hash 得到该客户所在分片。分片信息由 GNS 服务负责维护, 在用户第一次注册时分配, 之后客户每次登录时访问该数据。GNS 把分片信息同步给 DLS 服务, DLS 服务负责进行服务的交付, 收到用户请求后根据 GNS 的分片信息生成路由到该分片的策略。DLS 把请求转发到对应的 DSU 后, 由 DSU 完成处理。

成业务逻辑，并把结果返回给客户端。

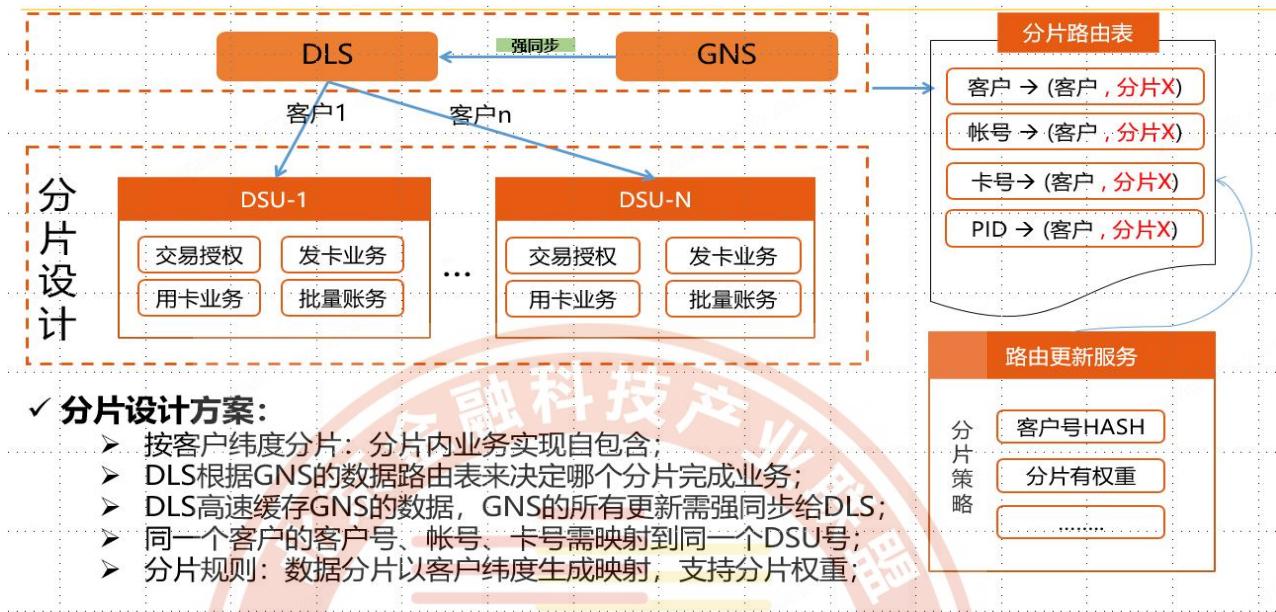


图 29 单元分片的路由逻辑

3) 分布式数据库部署方案

如图 30 所示是平安银行单元化的部署架构，采用了两地三中心的架构，根据所承载用户的不同划分出多个逻辑单元。

IDC-1	IDC-2	IDC-3	
DSU1	DSU1	DSU1	DSU1
DSU2	DSU2	DSU2	DSU2
DSU3	DSU3	DSU3	DSU3
DSU4	DSU4	DSU4	DSU4
DSU5	DSU5	DSU5	DSU5
DSU6	DSU6	DSU6	DSU6
DSU7	DSU7	DSU7	DSU7
...	DSUn
DSU100	DSU100	DSU100	DSU100

图 30 单元化的部署架构

生产数据中心和同城数据中心之间采用数据强同步，如图 31 所示，即任意时刻同城 $RPO=0$ ， RTO 理论在 40s 左右，实际测试最坏的情况不超过 90s。当生产中心出现任何软硬件异常时，业务会自动切换到同城数据中心，期间的业务影响是业务总体的 $1/n$ 短暂不可用，不可用时间小于一分钟。

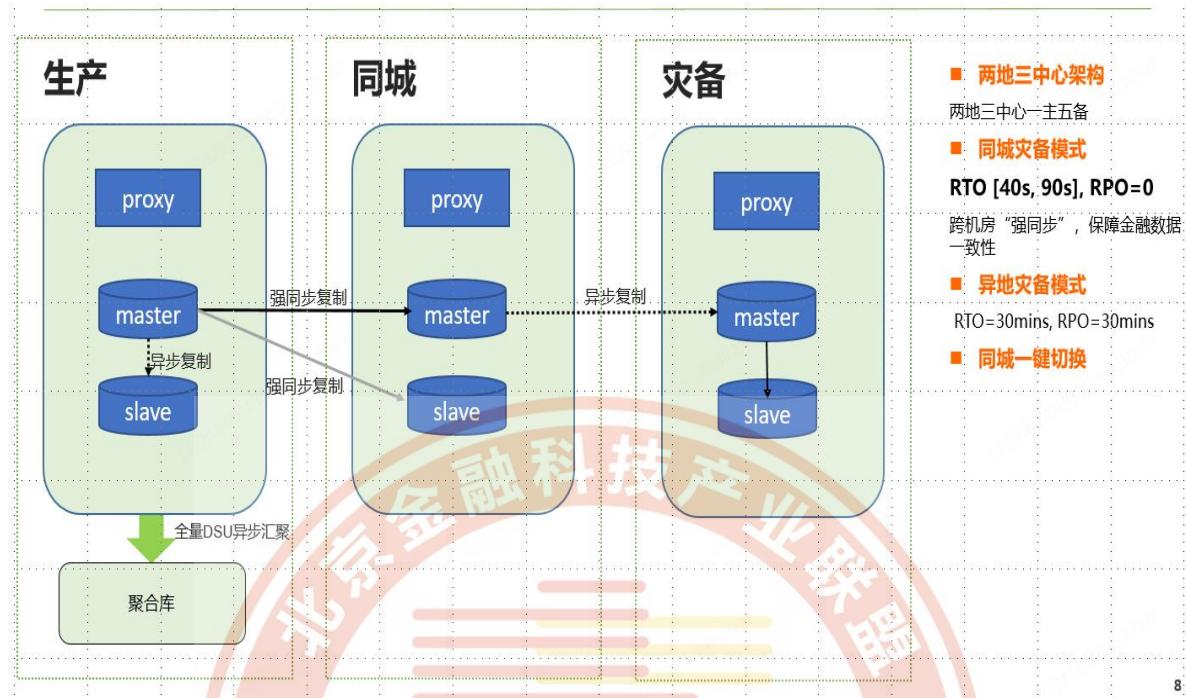


图 31 数据中心架构

4) 试点业务扩容策略

设计扩容策略时同时考虑到纵向和横向的扩容场景，如图 32 所示。

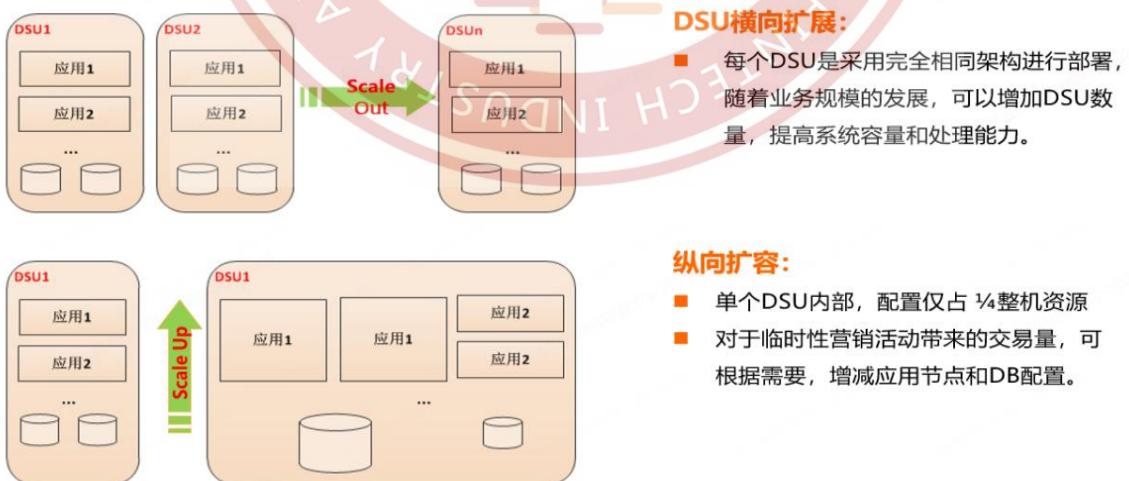
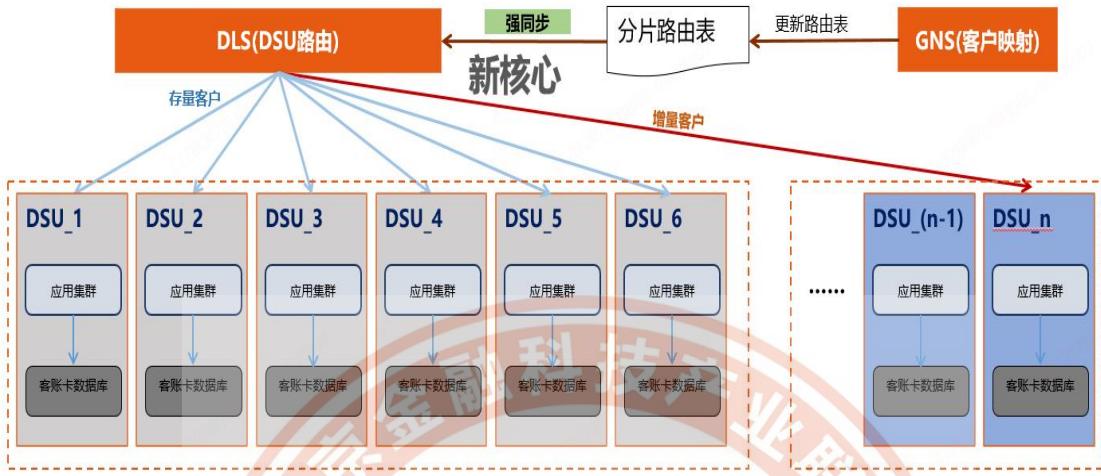


图 32 系统配置扩容

纵向来说，初始上线的系统配置只占服务器资源的 25%，这意味着必要时在不更换硬件的情况下可以进行 4 倍的扩容。而横向则可以支持无限的横向扩容，随着业务增长，可以满足任意量级的业务发展需求。

虽然可以方便的进行纵向扩容，但是需要理解，分布式架构对纵向扩容的需求应该尽可能低，因此设计的纵向扩容方案为“以防为主，以治为辅”的方针，尽量避免数据迁移的发生。同时还需充分考虑到单元的容量余量，给 DSU 容量预留了一定的 buffer。

横向扩容的整体逻辑如图 33 所示：新建一组或多组单元。调整客户的路由规则，比如新的客户路由到新单元的比重调高，让大部分新客户端路由到新的分片。由于路由规则的调整，后续老的 DSU 业务基本不再增长，新 DSU 承担了新业务的流量。



✓ 扩容方式：存量节点满足性能要求

- 数据由DLS通过数据路由表，决定归属哪个分片
- 扩容不涉及存量数据迁移，增量数据访问新增节点
- 分片扩容依据分片的**实际资源使用和容量指标设定扩容阈值（比如40%）**，根据阈值启动扩容
- 不涉及存量数据，可以不停机横向扩展

图 33 横向扩充

5) 数据操作包括数据访问和更新，为了支持单元化的架构，可把数据访问工具和业务一样进行分片化访问，可以一次访问一个分片，也可以一次访问全部分片。虽然系统是分布式的，但是事务却未必要分布式，相反应尽可能规避分布式事务，以保证系统的健壮性。在业务逻辑上，尽可能实现单元内业务自包含。如果一定要进行分布式事务，需考虑在框架层支持，如果框架也不能支持时由应用层实现，应尽可能避免使用数据库层的分布式事务，一方面数据库的分布式事务还不成熟，另外容易形成对特定数据库的依赖并产生单点故障的风险。

6) 试点业务维护方案

由于所有的系统节点完成了去中心化的工作，因此对所有节点都可以使用滚动升级的模式进行维护，而不影响业务的连续性。同时对可能发生意外或者容易出现故障的环节做了完善的应急预案，在紧急情况下可以自动或者人工模式快速的故障恢复。

3. 应用成果

新信用卡核心系统实现了100%源码的自主知识产权。经过压测，新信用卡核心系统并发业务量至少达到原AS400核心系统并发量的20倍，交易耗时降低2.5倍，批量耗时降低3倍，新业务功能的交付周期大大缩短，业务应对市场能力得到提升，系统总体建设、维护成本降低70%左右，系统可用率提升一个档次。

1) 业务时间里程碑节点

2020-10-31，信用卡A+上线切换成功，是分布式方案迈出第一步最重要的里程碑。

2020-11-11，新的信用卡A+上线后第一个双十一和业务促销活动，期间系统运行平稳。

2021-04-30，完成信用卡A+容灾演练，容灾切换过程平滑，系统运行稳定。

2021-07-19，信用卡A+同城切换演练，采用到点一键切换，并在同城运行一天，系统运行平稳。

经过系统促销活动、重要节日、同城、容灾切换等检验，系

统具备自动化同城切换，分钟级容灾切换并真正承担全业务流量的能力。

2) 试点业务非功能指标

系统可用率大大提高，可以达到 5 个 9。使用了仅 1/3 的成本建设了各项指标都远超原系统的集中式架构。具体新老核心对比如图 34 所示。



图 34 新老核心效果对比

3) 总结和展望

单元化的分布式架构有着巨大的优势，主要体现在，第一：虽然单机的可用率很低，但是分布式系统整体的可用性却大大提

升；第二：通过对基础架构、数据库和应用的自主可控，达到了业务层的自主可控和业务快速创新的要求；第三：单元化的分布式架构，搭积木式的即插即用模式，彻底解决了系统发展的瓶颈，无惧任何促销。

当然，单元化架构也有其弊端，主要有以下方面，第一：分布式架构建设和改造的工作量大，需要配套建设的分布式工具也很多；第二：过度单元化可能会带来很多问题，比如上下游业务不均衡带来的小业务被动单元化，以及单元化带来的应用层和数据库层的资源浪费等；第三：单元的失控和故障的蔓延，分布式的故障如果不能做到有效的隔离，很容易形成蔓延，那将会是灾难，因此避免单元的管理失控非常重要。

分布式单元化的架构，在信用卡 A+ 业务上取得成功之后，平安银行也在总结经验，把信用卡的模式推广到其他的核心信息，后续平安银行成功上线了统一支付系统的分布式改造，另外对其他核心系统的改造也在规划和进行中。

（三）华夏银行案例

1. 应用背景

随着华夏银行数字化转型战略的持续推进，银行业务系统需要同时具备高性能、可扩展、高可用、高容错等特性，为支撑各分行特色业务的快速发展，更加开放与灵活的技术架构成为首选。

2. 建设方案

1) 单元化业务总体架构

华夏银行分行中间业务平台采用区域拆分的大单元设计，设计要求统一入口，流量调配，各分行业务进行资源隔离。数据基于各分行进行设计方案，采用区域拆分的方式，分行产品的所有数据放入一个单元，对应到一个单独的分布式数据库集群，如单元1中代发工资、ETC等业务产品对应北京分行DB，组成完整区域大单元，其他分行依次类推。如图35所示：

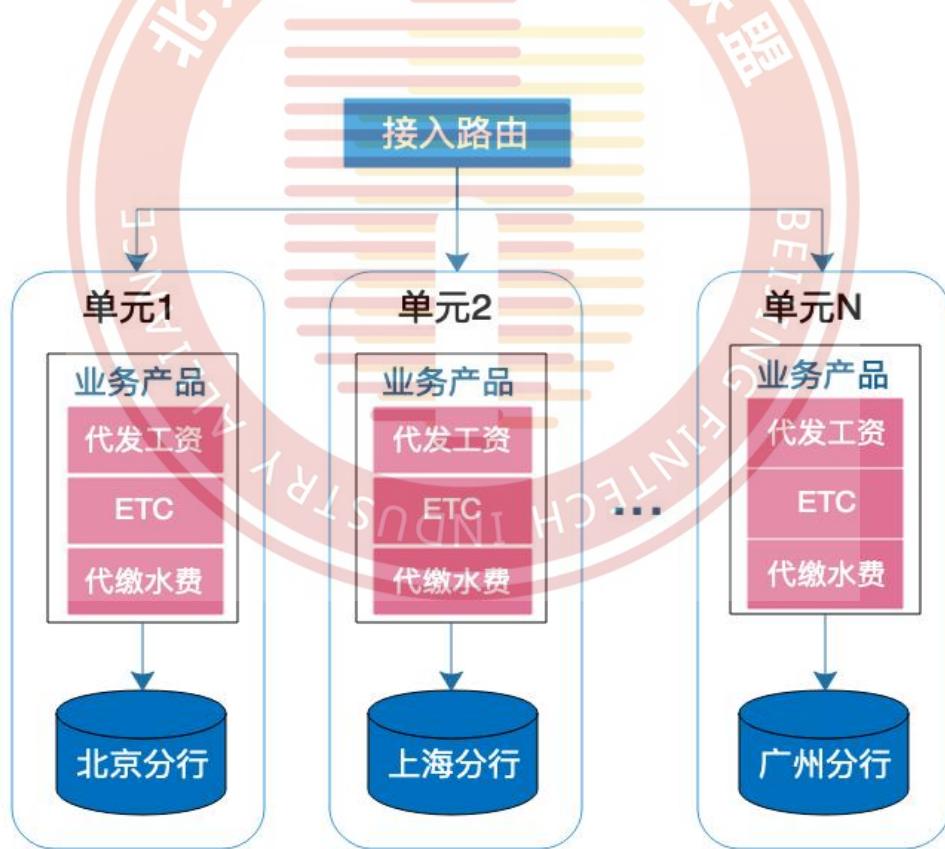


图 35 华夏银行单元业务示意图

2) 分布式数据库部署方案

如图 36 所示是华夏银行单元化的部署架构，根据分行地域的不同划分出多个逻辑单元，采用同城双中心架构。

数据库层采用的是 1 主 3 备的方式，本地机房 1 备，另外中心机房 2 备份，确保一个节点的 ack 返回，主库提交，这样最大的保障数据一致性并兼顾性能问题，最大限度保证数据的安全。

同城双中心部署具备同城双活能力，每个数据中心均可对外提供服务，同时具备城市内部的跨中心容灾能力，任何一个数据中心或主单元故障都可以切换到备中心或备单元提供服务。

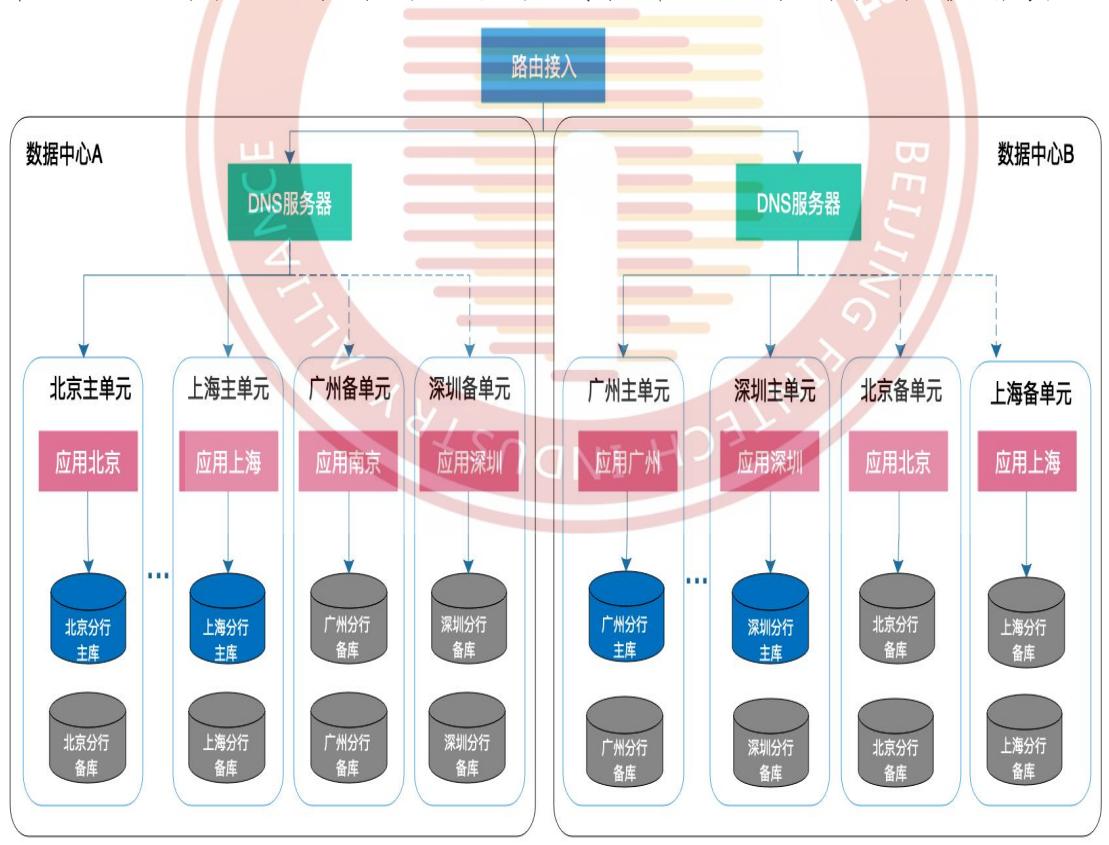


图 36 单元业务部署图

3) 业务高可靠与业务连续性架构

在城市大集群下进行跨中心的一主三从（四副本）模式进行部署，通过配置同城从中心两副本为强同步模式（保证所有数据变更至少会在所有强同步副本中有一个完成持久化），保证所有数据的变更具备跨中心的副本来应对集群的跨中心高可用能力要求，最大限度的保障数据一致性并兼顾性能问题，保证数据的安全（ $RT0<30$ 秒， $RP0=0$ ）。主库故障切换优先切换本地备库提升为主库，同时任何一个主单元故障，备单元都可以快速接管。以北京分行为例如图 37 所示：



图 36 华夏银行北京分行主库故障示意图

4) 业务单元的数据切分与路由方案

分行中间业务数据切分为按照分行地域进行的大单元设计。每个分行单元包含多个业务模块，每个独立分行单元对应一个分布式数据库集群。在接入层通过跨中心高可用的动态域名对应用端提供统一访问入口，通过不同的云平台 IP 进行路由，分发到对应的分行单元中。

5) 业务扩容策略

目前各分行数据库集中存放在独立的大集群数据库中，不同业务场景采用不同的数据拆分策略，对于业务增长慢、容量需求较小的业务并未做数据拆分，采用增加本地资源的策略进行纵向扩容，该方式会因为单机限制存在一定的瓶颈。对于业务增长快、容量需求大的业务做了数据拆分，采用增加多个数据节点组的策略进行横向扩容。

6) 业务跨单元数据操作与分布式事务设计

跨单元操作会产生分布式事务，跨节点的数据一致性需要使用 2PC/3PC/TCC 等实现分布式事务，造成事务信息规模扩大，导致系统性能下降。引入其他中间产品造成运维成本上升，影响系统可靠稳定。因此设计之初，各分行业务单元数据独立，无跨单元与分布式事务设计。

3. 应用成果

华夏银行分行中间业务平台试点已完成了单元化整体改造，

于 2019 年上线成功运行平稳，日均交易量 100 万笔，平均耗时 10ms，目前已承载 40 多家分行中间业务。

通过对业务结构的单元化设计实现了灰度发布、同城双活架构，满足了业务高可用、高性能、高扩展的技术要求。同时采用了国产软硬件，实现了科技创新与安全可控的目标。

（四）微众银行案例

1. 应用背景

微众银行自成立之初，在基础架构选型阶段，就抛弃了的传统的 IOE 架构，转而选用了分布式架构。微众银行设计并建设了基于用户划分的单元化的分布式核心架构。在单元化分布式架构的前提下，不同的业务场景对于数据库存储有着多样化的要求。

1) 由于每个业务单元只需要承载固定数量的用户，所以在业务单元内，数据库的容量和性能需求是可控的，我行在单元内采用单实例架构数据库，可承载容量 3TB 以内的联机数据存储需求（受限于单实例下服务器硬件限制），简化数据库架构。

2) 除了可拆分的单元化业务，还存在批量系统、全局管理类业务系统、多单元数据汇聚、数据归档类等大量非单元化业务，此类业务一般有持续的数据增长需求，对数据库的容量和吞吐都会较高的需求，需要支持水平可扩展的原生分布式数据库支持。

在以上场景需求的前提下，微众银行根据我行的业务场景需求以及对数据库产品的特性要注，经过多轮的调研、评测与验证，

最终确定了以下两类数据库选型方案：

1) 选择腾讯云数据库产品 TDSQL（单实例模式），来承载业务单元内核心交易系统、部分核心批量系统、部分全局类业务系统的数据库需求。

2) 选择平凯星辰（北京）科技有限公司的数据库产品 TiDB，来承载部分核心批量系统、部分全局类业务系统、多单元数据汇聚和归档系统的数据库需求。

2. 建设方案

1) 单元化总结架构

微众银行设计并建设了基于用户划分的单元化的分布式核心架构。每个业务单元都完整包含应用组件、中间件组件、数据库组件等，形成一个完整的自包含单位。

我行采用先垂直拆产品然后水平按照客户号做拆分的单元划分模式。不同的产品线使用不同类型的单元，每个单元只会包含 1 个业务一个固定数量的客户；单元客户容量上限固定，容量超出时新客户号进新的单元；一个客户号原则上固定属于某一个单元，不会出现客户号在单元间的挪动。如果当前所有的业务单元的用户量已达到上限，或者即将达到上限，将发起业务单元的水平扩容。通过业务单元的水平扩容，实现整个系统的水平扩展能力。同时还会设计一个较为特殊的全局管理单元。此单元主要存放全局类业务，此类业务没有用户属性，或者无法按照用户维

度进行业务拆分。基于我行的架构设计与实际业务需求分析，我行最终确定了 TDSQL 与 TiDB 两款数据库产品。整体数据存储架构如图 37 所示：



图 37 微众银行单元化总体架构

在业务单元内采用 TDSQL（单实例模式）作为关键业务交易系统的核心数据库。每个业务单元内有若干个 TDSQL 单实例（又称“TDSQL SET”），单元不同的子系统的库表，可以部署在不同的 TDSQL SET 内，以实现子系统间的数据库资源隔离。

在全局管理单元内，对于数据容量较小（建议在 3TB 以内），且增长比较稳定的全局类业务，数据库采用 TDSQL 单实例模式；对于数据容量较大（3TB 以上），且持续快速增长的全局类业务，数据库采用支持一键水平扩展的 TiDB 数据库集群，以实现更大容量的支持。

部分核心批量类业务（如贷款核心批量），随着单用户的贷款交易记录持续累积，批量系统所需处理的数据量会持续增长，对于数据库的容量和吞吐需求会越来越高。这一类批量系统已不适合使用业务单元内的数据库去承载，可以采用支持一键水平扩展的 TiDB 数据库集群，通过数据库容量和性能的水平扩展，保持批量执行的效率稳定，并实现可持续发展。

多单元数据汇聚、数据归档类业务需要汇集多个单元的历史数据，数据量一般较大，且长期处于持续增长的趋势。这一类业务场景当前大多选用大数据存储组件（如 HBase 数据库）来进行数据存储。但是，大数据存储组件一般存大 SQL 兼容度不高、高可用能力较弱的缺点，对于业务系统，研发难度较大，风险较高，建议采用支持一键水平扩展的 TiDB 数据库集群，以实现容量的水平扩展，保证业务的连续性，同时也可解决 SQL 兼容度和数据库系统高可用的痛点。

2) 分布式数据库部署方案

TDSQL 是腾讯云基于 MySQL/Mariadb 开源社区版本打造的

一款金融级数据库解决方案。在数据库内核层面，TDSQL 针对 MySQL 社区版本和 Mariadb 社区版本的内核，在复制模块做了系统级优化，使得其具备主备副本数据强一致同步的特性，极大提升了数据安全性，同时相对原生的半同步复制机制，TDSQL 强一致复制的性能也有极大提升。

如图 38 所示，TDSQL 集成了 TDSQL Agent、TDSQL SQLEngine、TDSQL Scheduler 等多个模块，实现了读写分离、自动主备强一致性切换、自动故障修复、实时监控、实时备份等一系列功能。

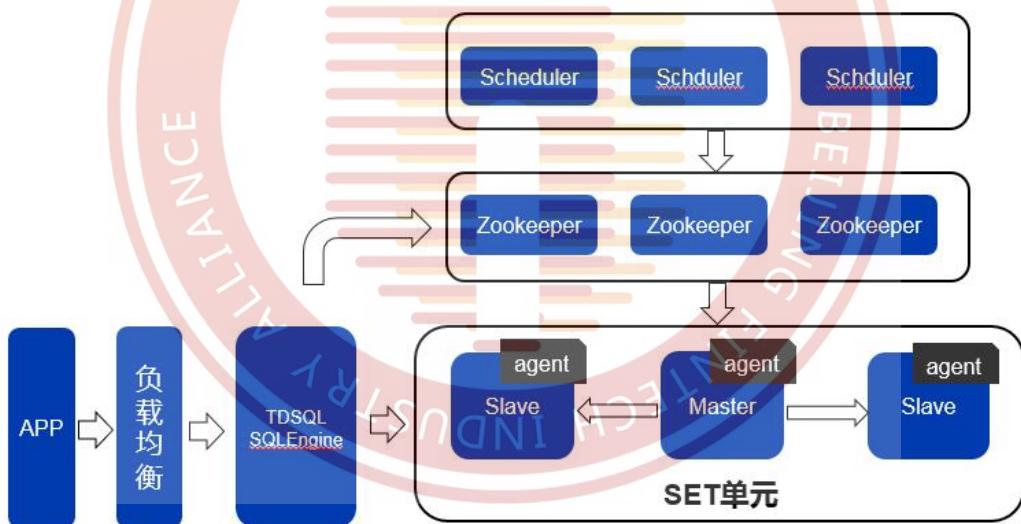


图 38 TDSQL 逻辑架构模型图

TDSQL 的最小服务单元称为一个 SET 实例，一个 SET 实例可以包含一主多备节点（默认是一主二备模式），主备节点之间采用数据强一致复制机制，保证数据的高可靠；在主节点发生故障

宕机时，可以触发自动切换机制，在秒级时间内，完成主备节点切换，以保证数据库的高可用。

我行在同城建设了多个 IDC 数据中心，IDC 之间两互联，并建立多条网络专线，保证 IDC 数据中心之间的网络稳定。数据库的架构需要支持实现同城 IDC 级别的高可用，以及应用的同城多 IDC 多活访问。基于以上需求，我们设计了 TDSQL 同城多机房部署方案，如图 39 所示。

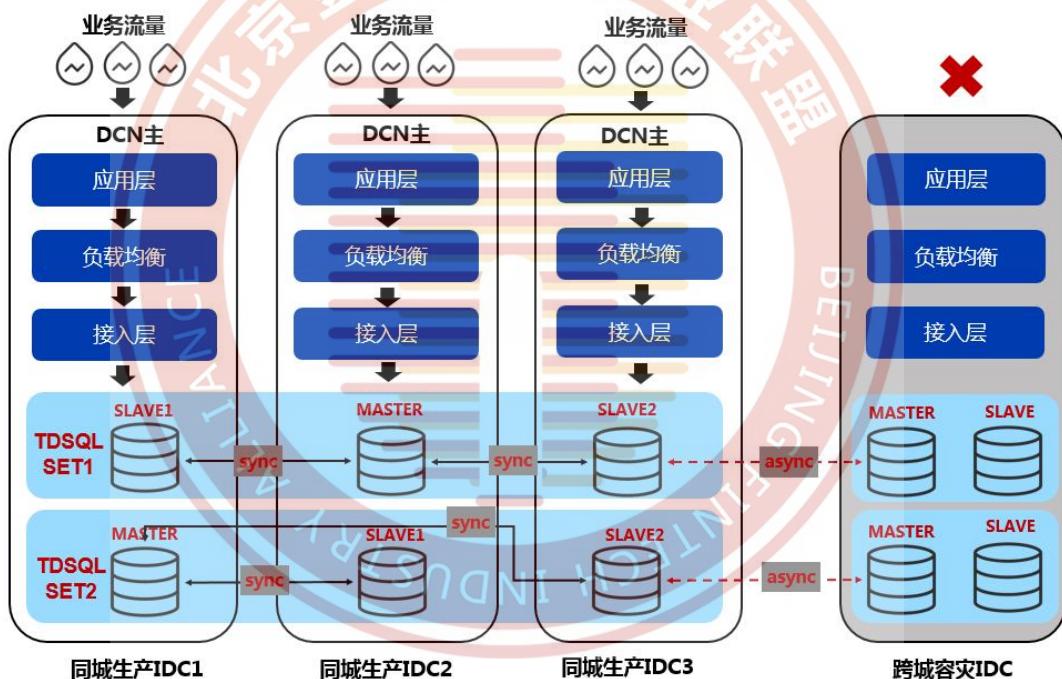


图 39 微众银行基于 TDSQL 的同城多活架构

部署方案采用同城 3 副本 + 跨城 2 副本的 3+2 部署模式（一个 TDSQL SET 内）。同城 3 副本为 1 主 2 备，分别部

署同城的 3 个 IDC 中，副本之间采用 TDSQL 强一致数据同步，保证同城 3 IDC 之间的 RP0=0；TDSQL 的高可用自动主备切换机制，可以保证 RT0 在秒级恢复。

跨城容灾的 2 副本通过同城的一个 slave 节点进行异步复制，实现跨城的数据容灾。基于以上部署架构，在同城可以做到应用多 IDC 多活，即联机的业务流量，可以同时从 3 个 IDC 接入，任何一个 IDC 故障不可用，都可以保证数据 0 丢失，同时在秒级内可以恢复数据库服务。

TiDB 是平凯星辰（北京）科技有限公司自主设计、研发的开源分布式关系型数据库，是一款同时支持在线事务处理与在线分析处理（Hybrid Transactional and Analytical Processing, HTAP）的融合型分布式数据库产品，具备水平扩容或者缩容、金融级高可用、实时 HTAP、云原生的分布式数据库、兼容 MySQL 5.7 协议和 MySQL 生态等重要特性。

如图 40 所示，TiDB 整体由三个模块组成：TiDB Server 负责 SQL 接收、SQL 解析与优化、SQL 转化等；TiDB Server 是无状态的，其本身并不存储数据，只负责计算，可以水平扩展，可以通过负载均衡组件对外提供统一的接入地址。PD Server Placement Driver（简称 PD）为集群管理模块，负责集群的元数据管理、集群调度和负载均、分配全局唯一事务 ID；PD 需部署奇数个节点，一般线上推荐至少部署 3 个节点。TiKV Server

负责数据存储，底层使用 RocksDB 作为 KV 存储引擎；TiKV 支持多副本，使用 Raft 协议做复制，保持数据的强一致性和容灾；应用层的 SQL 请求，由 TiDB Server 转化为 KV 请求，最终写入到 TiKV Server 中存储。

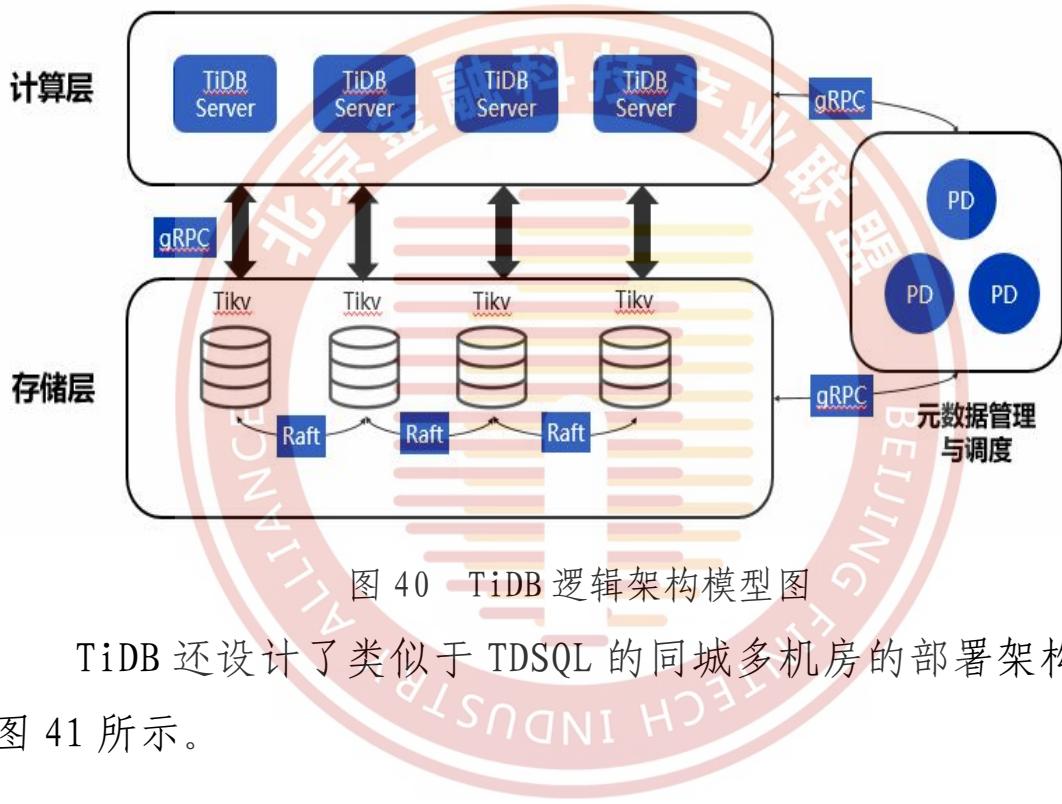


图 40 TiDB 逻辑架构模型图

TiDB 还设计了类似于 TDSQL 的同城多机房的部署架构，如图 41 所示。

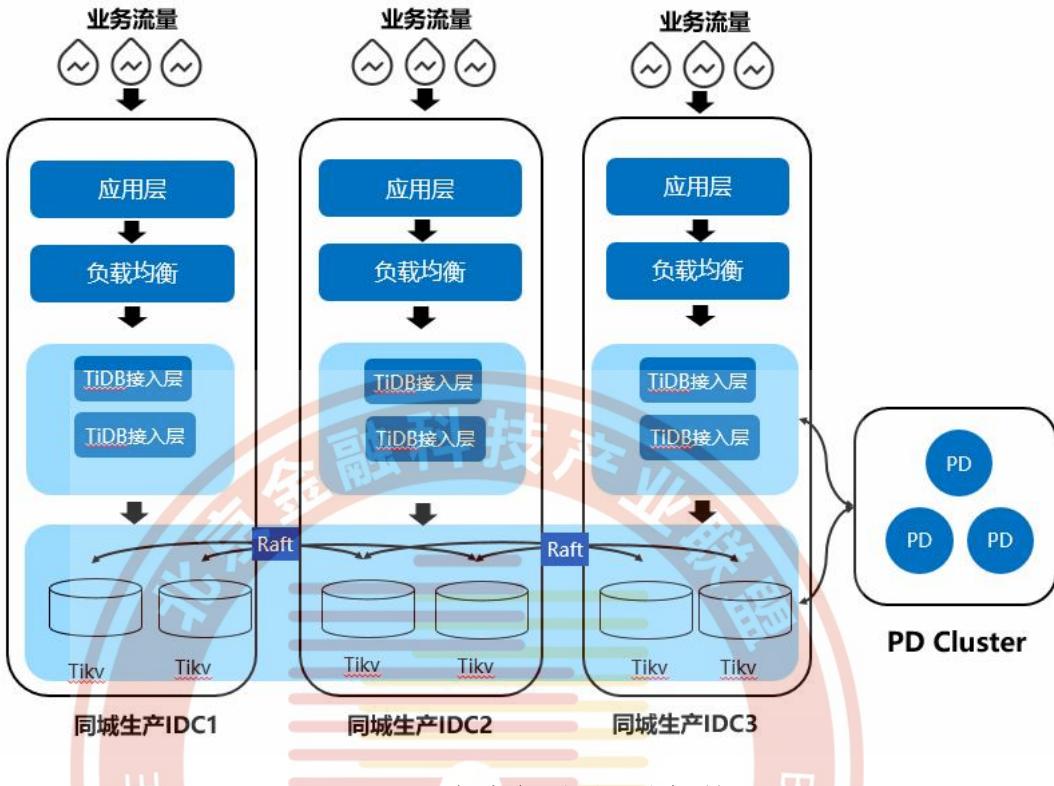


图 41 TiDB 同城多机房部署架构

TiDB 数据库存储引擎层 TiKV 模块，采用跨同城三 IDC 部署，每个 IDC 保存一个数据副本，三个副本之间通过 raft 协议进行强一致性同步，在主副本节点故障或宕机的情况下，可以触发 Raft 协议的自动切换机制，快速重新选出主节点。基于以上机制，可以实现 RP0=0，RT0 在 30 秒以内。

接入引擎层 TiDB Server 模块在每个 IDC 各自部署一套集群（节点数 ≥ 2 个），然后接入到所在 IDC 的负载均衡模块。应用通过各个 IDC 的负载均衡模块的 VIP，接入 TiDB 进行读写访问，实现应用同城多活。

TiDB 的管理与调度模块 PD Server 同样也是跨同城三 IDC

部署，每个 IDC 部署一个节点。在同城多机房部署的架构下，任何一个 IDC 都可以提供业务接入，任何一个 IDC 的服务器节点故障，或者整体 IDC 故障，都不会影响业务可用性，或者只是短暂的影响，然后在 RTO 的预期内快速恢复。

3) 业务扩容策略

微众银行在单元内采用的是单实例主备架构的数据库模型。单元内的扩容主要通过数据节点垂直扩展实现，即通过数据库节点在线滚动升级的方式逐步将一个数据节点组相关的多个数据库的物理资源进行升配，例如提高 CPU、内存、网络、磁盘 IO 等。

当某个业务单元的用户数量达到预设值，就需要进行新增单元扩容。微众银行通过单元自动化扩容平台，可以实现单元一键部署功能，实现高效率的单元扩容。已经满载的单元将不再接受新的用户开户，所有的新开户请求将会被自动路由到新扩容的单元中。

4) 业务跨单元数据操作与分布式事务设计

微众银行通过自研的分布式消息中间件平台进行跨单元间的数据交互。对于跨单元间的事务一致性，由应用层实现跨单元的分布式事务保证。在此不做详述。

3. 试点成果

经过几年的发展，微众银行目前已有数千个 TDSQL 实例，近

百个业务单元，承载行内的所有核心业务系统。TDSQL 管理的数据规模达到 PB 级，每天承载数亿次的金融交易。同时，还建立了数十个 TiDB 集群，承载多单元数据汇聚、核心业务批量、数据归档、全局类业务存储等业务场景，服务器节点达数百个，数据规模达到 500TB 以上。

在微众银行单元化分布式架构的基础上，通过对 TDSQL 和 TiDB 数据库的综合应用，有效的解决了行内各类业务对于数据库存储需求，也进一步提升了行内单元化分布式核心架构的可靠性与完备性。

（五）支付宝案例

1. 应用背景

支付宝业务诞生于 2004 年。到 2013 年，支付宝的应用层已经是无状态的了，可以随意水平扩展，容量足以满足业务需求。此时的支付宝核心数据库依然使用 Oracle，并按用户维度水平拆分。应用层流量是完全随机的，任意一个应用节点都可能访问任意一个数据库节点，因此每个应用都需要占用数据库连接，而数据库连接是非常宝贵的资源，是有上限的，这必然导致 Oracle 数据库的连接数不足。

因此，数据集连接数的瓶颈导致了应用不能扩容，意味着支付宝系统的容量定格了，不能再有任何业务量增长。既无法应对大促活动的洪峰流量，随着业务的高速发展，支撑日常业务也捉

襟见肘。

为解决上述问题，支付宝自研了 OceanBase 分布式数据库，在普通硬件上实现金融级的高可用性。OceanBase 使用 Paxos 协议在保证数据强一致的前提下，实现水平扩展，给上层应用提供充足的数据库连接、计算以及存储资源，满足支付宝业务高速发展的需要，支撑支付宝实现了单元化异地多活的技术架构。

2. 建设方案

1) 单元化业务总体架构

支付宝的技术架构也经历了单活、同城双活、两地三中心三个阶段，随着支付宝业务量的规模和复杂度越来越高，业务对于基础架构的要求也越来越高，包括扩展能力、容灾能力、灰度能力等。最终支付宝发展到了单元化架构，将主要业务拆分单元即分片，装入不同的逻辑单元内，每个分片的数据库实现三地五中心部署即三地五中心的单元化架构。支付宝单元化架构包含 RZone、GZone 和 CZone 三类：

其中 GZone 部署的是无法拆分的数据和业务，GZone 的数据和业务被 RZone 依赖，GZone 全局只部署一份。

CZone 的出现是因为 GZone 全局只有一份，不同城市的 RZone 可能依赖 GZone 服务和数据的时候需要远距离调用，延迟比较大，所以在每个城市部署一个 CZone 作为 GZone 的只读副本，为本城市的 RZone 提供服务。

RZone 部署的是可拆分的业务和对应的数据。每个 RZone 内的数据分片如图所示有五副本，实现三地五中心部署，每个分片内只有一个可写入的主副本，其余副本按照 Paxos 协议做数据强一致。每个 RZone 内实现业务单元封闭，独立完成自己的所有业务。

支付宝对单元化的基本要求是每个单元都具备服务所有用户的能力，即具体哪个单元服务哪些用户是可以动态配置的，当某个单元因为故障而无法提供服务时，客户端的访问可以被重新路由到其他单元，由其他单元的应用及数据库提供服务。如图 42 所示。



图 42 支付宝单元化整体架构

任何应用的 RZone 都可以给所有客户提供服务，对数据库的要求就是需要每个单元都包含全量数据。OceanBase 数据库采用三地五中心部署，每个 IDC 部署集群的一个 Zone，每个 Zone 都包含业务所需要的全量数据。

2) 业务单元的数据切分与路由方案

单元化的必要条件要求全站所有业务数据切分所用的拆分维度和拆分规则都必须一样。支付宝以用户来切分数据，交易、收单、微贷、支付、账务等全链路业务都基于用户维度拆分数据，并且采用一样的规则拆分出同样的切片数，支付宝以用户 id 末 2 位作为标识，将每个业务的全量数据都划分为 100 个切片（00-99）。

把一个或几个数据分片部署在某个单元里，这些数据分片占总量数据的比例，就是这个单元能够承担的业务流量比例。选择数据分片的维度是个很重要的问题，一个好的维度应该粒度合适。粒度过大，会让流量调配的灵活性和精细度受到制约；粒度过小会给数据支撑资源和访问逻辑带来负担。

通过引入数据访问中间件，可以实现对应用透明的分库分表。一个比较好的实践是：逻辑拆分先一步到位，物理拆分慢慢进行。以账户表为例，将用户 ID 的末两位作为分片维度，可以在逻辑上将数据分成 100 份，一次性拆到 100 个分表中。这 100 个分表可以先位于同一个物理库中，随着系统的发展，逐步拆成

2个、5个、10个，乃至100个物理库。数据访问中间件会屏蔽表与库的映射关系，应用层不必感知。这100个表创建时候，可以配置不同的Primary Zone，从而灵活配置其主副本位于哪个单元。比如第一个分表的Primary Zone是Zone1>Zone2>Zone3，表示该表的主副本默认都在Zone1内，而与之对应的业务系统也在Zone1内，保证正常情况下整个处理都在一个单元内闭环。当单元内数据库服务器发生故障时，数据库集群会按照该表配置的Primary Zone的顺序调整其他单元内的从副本为主副本，这个过程是自动完成的，且可以保证RPO=0。

这么多的组件要协同工作，必须共享同一份规则配置信息，如图43所示。



图43 统一路由规则

必须有一个全局的单元化规则管控中心来管理，并通过一个

高效的配置中心下发到分布式环境中的所有节点。规则的内容比较丰富，描述了城市、机房、逻辑单元的拓扑结构，更重要的是描述了分片 ID 与逻辑单元之间的映射关系，如下表 6 所示。

表 6 规则配置内容示例表

城市	数据中心	业务分片 ID	数据分片 ID
城市 1	IDC1	RZ01: 【00-19】	【00-19】
	IDC2	RZ02: 【20-39】	【20-39】
城市 2	IDC3	RZ03: 【40-59】	【40-59】
	IDC4	RZ04: 【60-79】	【60-79】
城市 3	IDC5	RZ05: 【80-99】	【80-99】

单元化是个复杂的系统工程，需要多个组件协同工作，从上到下涉及到 DNS 层、反向代理层、网关/WEB 层、服务层、数据访问层。总体指导思想是“多层防线，迷途知返”。每层只要能获取到足够的信息，就尽早将请求转到正确的单元去，如果实在拿不到足够的信息，就靠下一层。

DNS 层感知不到任何业务层的信息，但支付宝使用“多域名技术”。比如 PC 端收银台的域名是 `cashier.alipay.com`，在系统已知一个用户数据属于哪个单元的情况下，就让其直接访问一个单独的域名，直接解析到对应的数据中心，避免了下层的跨机房转发。例如 `cashiergtj.alipay.com`，`gtj` 就是内部一个数据中心的编号。移动端也可以靠下发规则到客户端来实现类似的效果。

反向代理层是基于 Nginx 二次开发的，后端系统在通过参数识别用户所属的单元之后，在 Cookie 中写入特定的标识。下次请求，反向代理层就可以识别，直接转发到对应的单元。

网关/Web 层是应用上的第一道防线，是真正可以有业务逻辑的地方。在通用的 HTTP 拦截器中识别 Session 中的用户 ID 字段，如果不是本单元的请求，就 forward 到正确的单元。并在 Cookie 中写入标识，下次请求在反向代理层就可以正确转发。

服务层 RPC 框架和注册中心内置了对单元化能力的支持，可以根据请求参数，透明地找到正确单元的服务提供方。

数据访问层是最后的兜底保障，即使前面所有的防线都失败了，一笔请求进入了错误的单元，在访问数据库的时候通过 OBProxy 查询数据主副本所在的单元，也一定会路由到正确的库表，最多耗时变长，但绝对不会访问到错误的数据。即使 OBProxy 无法准确识别主副本所在位置，他也会随机路由到任何一个 OBServer，这个 OBServer 可以在集群范围内找到应用所需的数据并返回。

3) 分布式数据库部署方案

支付宝系统中任何一个单元都可以给所有客户提供服务，对数据库的要求就是需要每个单元都包含全量数据。因此 OceanBase 数据库采用三地五中心部署，每个 IDC（也即是每个单元）部署集群的一个 Zone，每个 Zone 都包含全量数据。

如图 44 所示，RZone 的数据是可分片的数据，需要确保数据与应用部署到一个单元内。以【00-19】分片为例，这些分片的数据在 5 个 IDC 都有副本，默认情况下，所有的读写操作都在主副本完成，其他的从副本通过 Paxos 协议与主副本组成 Paxos 组。当要对数据进行修改时，主副本会同步 Redo-Log 日志给从副本，当主副本确认多数派的副本已经完成落盘后才会返回给应用。这样可以确保在任何少数派故障情况下都可以保证 RP0=0。为了保证主副本与应用都在一个单元内，可以对这些表配置 Primary Zone (Zone1 优先级最高)，从而让【00-19】分片内的所有主副本都位于 Zone1，与【00-19】分片的应用 RZone 在一个单元内，减少网络的开销。

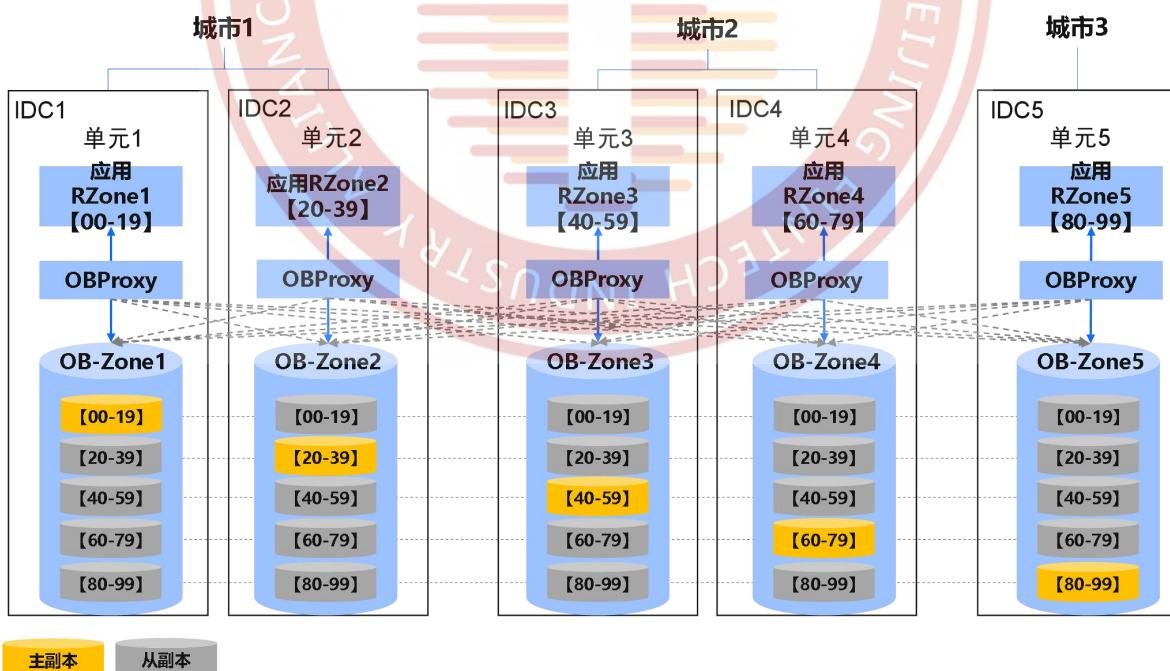


图 44 OceanBase 数据库 RZone 数据分布示例图

在这种情况下，由于所有的写操作需要满足多数派落盘，对于一个 5 副本的集群来说，OceanBase 需要满足 3 个副本的落盘才能返回给应用，而每个城市最多只有 2 个副本，因此对于任何一次写操作都需要跨城市的同步 Redo-Log 日志，对性能造成影响，因此城市 1 和城市 2 的距离不能太远。

为了降低存储成本，城市 3 的数据中心也可以部署日志型副本，该副本没有基线数据，只有日志数据，因此他只能参与投票和帮助其他副本恢复数据，而自己不能转为主副本提供服务。因此这种情况下，城市 3 的数据中心基础设施可以要求低一些，但他永远也无法成为主副本。

GZone 和 CZone 的数据一般是全局数据，且不可以分片，这些数据被 RZone 所依赖。如图 45 所示，支付宝将 GZone 数据部署为一个五副本，也就是每个单元都有 GZone 的全量数据，Primary Zone 配置为 (Zone1=Zone2 > Zone3>Zone4>Zone5)，将 GZone 数据的主副本汇聚到单元 1 和单元 2 内（位于一个城市），主副本会通过 Redo-Log 日志的方式强同步到其他单元内的副本，确保 RP0=0。所有单元内的应用要修改 GZone 的数据，都需要访问单元 1 和单元 2 的数据，会产生跨单元的延迟，好在一般应用较少更新 GZone 的数据。

应用访问 GZone 数据较为频繁，为了减少跨单元的网络延迟，OceanBase 集群可以在每个集群创建一个 GZone 的只读副本，

也就是 CZone。只读副本不参与 Paxos 组的投票，只是作为一个观察者实时追赶 Paxos 成员的日志，并在本地回放。因为 CZone 并没有加入 Paxos 组，不会造成投票成员增加导致事务提交延时的增加。但由于 CZone 与 GZone 之间强同步，当对 GZone 进行修改时，主副本需要收到所有 CZone 副本的反馈才会返回给应用，因为会有一定的延迟，不过 GZone 的数据修改不多，这个影响可以忍受。

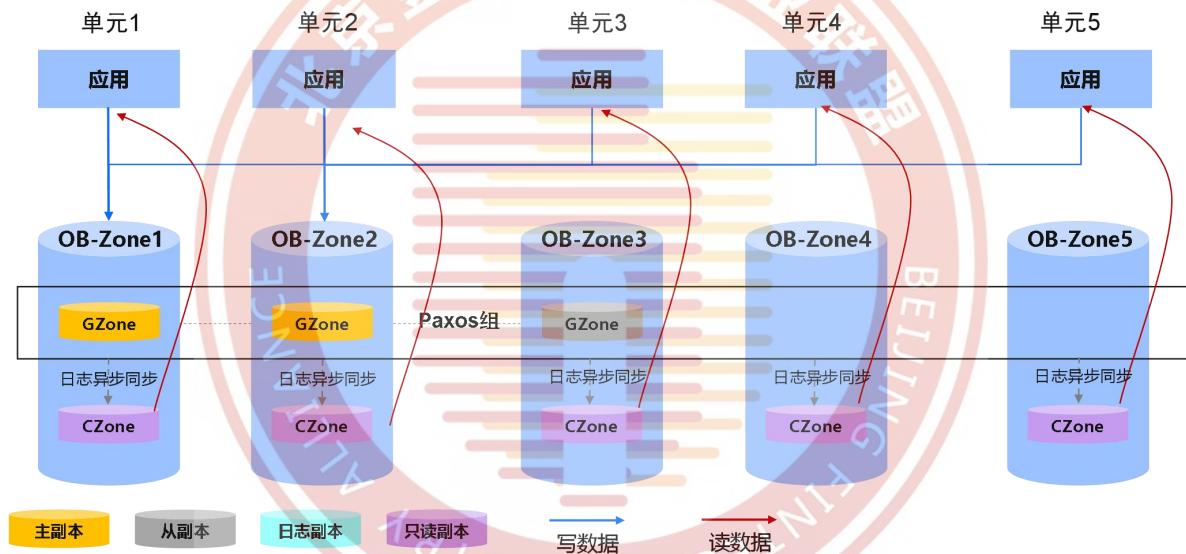


图 45 OceanBase 数据库 GZone 及 CZone 数据分布示例图

4) 业务高可靠与业务连续性架构

借助 OceanBase 多副本的能力，支付宝业务可以实现在少数派故障的情况下，实现 $RP0=0$, $RT0<30$ 秒的高可用能力。我们从几个场景分析下。

如图 46 所示，在每一个单元内，支付宝部署 2 套应用，例

如 RZ01-A 和 RZ01-B，他们都处理用户 ID 为【00-19】用户的业务，并连接【00-19】分片的数据。RZ01-A 和 RZ01-B 日常是双活的，各承担 50% 的用户访问流量，任何一套应用的故障不影响业务，同时这样的部署在蓝绿发布和线上灰度仿真时候也非常有用，可以先升级一套应用，观察稳定后再升级另外一套应用。



图 46 OceanBase 数据库单元内故障切换示意图

当单元 1 内数据节点故障而无法访问时，借助 Paxos 协议及配置的 Primary Zone，系统会让单元 2 内的【00-19】数据的从副本转正为主副本为单元 1 内的应用提供服务，但这样增加了两个单元内的网络延迟，影响性能，好在单元 1 和单元 2 一般部署在同城的两个机房，网络延时可以接受。待单元 1 内的故障恢复后，单元 1 内会补齐副本并加入 Paxos 组，然后从主副本那里

追平数据后重新成为主副本，继续让【00-19】用户的访问收敛到单元 1 内。这样的主从副本的切换，对应用是透明的，应用通过连接 OBProxy 将访问路由到最新的主副本的位置。

当一个数据中心断电或者光缆中断，那么整个单元的应用和数据均无法访问。例如单元 1 负责【00-19】分片的用户，单元 1 的整体故障将影响这些用户。如图 47 所示，支付宝会根据预先配置的策略，将【00-19】用户的请求路由至单元 2，此时 RZ02 将负责【00-19】和【20-39】两个分片的用户。单元 2 内【00-19】分片的数据将根据 Primary Zone 的配置，成为新的主副本，使得用户【00-19】的访问可以在单元 2 内终结，相当于把单元 1 负责的业务无缝搬迁到了单元 2 内。

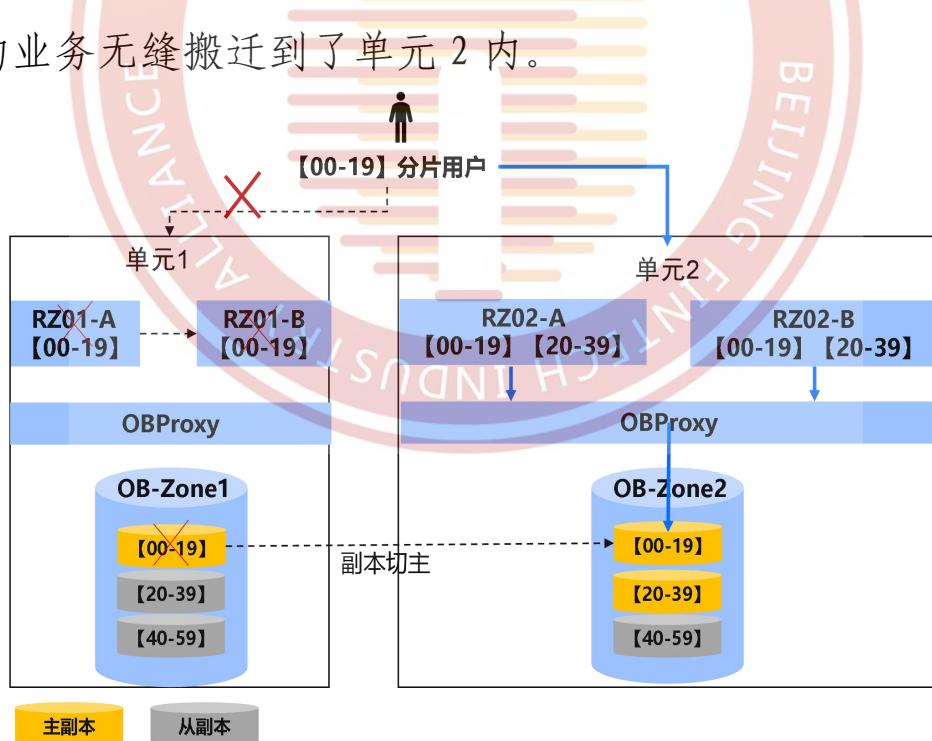


图 47 OceanBase 数据中心故障容灾切换示意图

一个城市整体故障的情况下，应用层流量通过规则的切换，由事先规划好的其他单元接管。支付宝采用三地五中心的设计，在城市 3 只有一个单元，因此该城市的灾难只影响一个单元，系统会自动将业务及数据搬迁到其他单元内。但对于城市 1 和城市 2 来说，由于每个城市都有 2 个单元，任何一个城市的灾难都会影响 2 个单元。比如城市 1 故障，那么单元 1 和单元 2 所服务的【00-19】和【20-39】分片的用户的服务会受到影响，通过规则切换，受影响用户的流量将由单元 3 和单元 4 接管，此时 RZ03 和 RZ04 将分别负责两个分片用户的数据。

OceanBase 数据库内的数据都有 5 个副本，虽然损失了 2 个副本，但剩余的 3 个副本依然构成多数派，依然可以服务。如图 48 所示，对于单元 1 来说，由于【00-19】分片数据配置的 Primary Zone 是【Zone1>Zone2>Zone3>Zone4>Zone5】，此时由于第一优先级 Zone2 也无法提供服务，那么单元 3 内的【00-19】分片的副本将成为新的主副本，此时【00-19】应用也由单元 3 内的 RZ03 接管，所以【00-19】用户的访问可以在单元 3 内部完成。同理，对于【20-39】分片的数据，他的 Primary Zone 是【Zone2>Zone1>Zone4>Zone3>Zone5】，由于 Zone2 和 Zone1 都已无法提供服务，因此 Zone4【20-39】分片的数据将成为主副本，刚好与应用都在单元 4 内。因此配置 Primary Zone，OceanBase 可以很好的根据应用单元化的切换来灵活的切换主副本的位置，

实现业务在一个单元内完成。

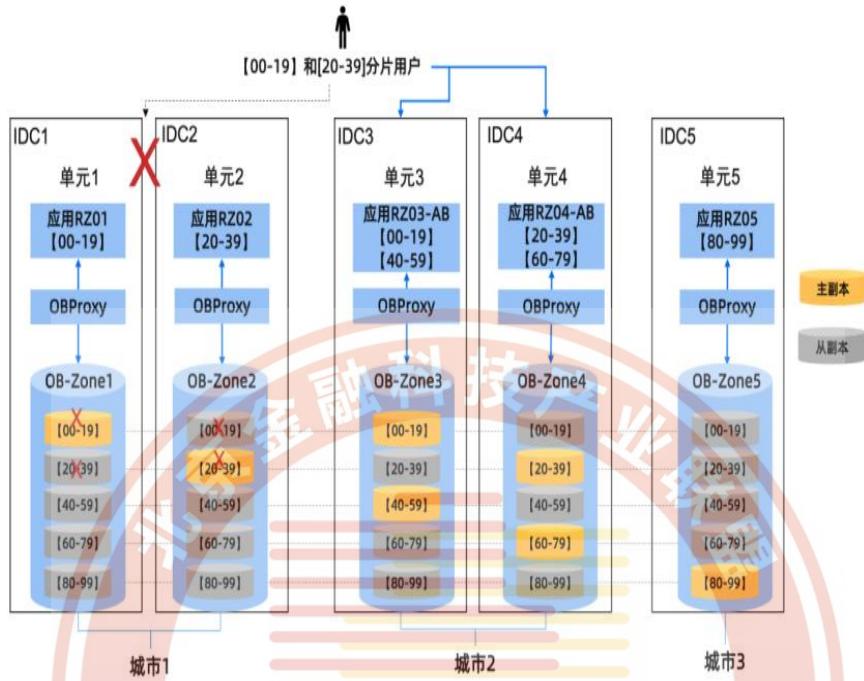


图 48 OceanBase 数据库城市级故障容灾切换示意图

但此时由于只剩 3 个副本，对于一个 5 副本的集群来说，需要满足多数派落盘（3 个副本落盘），所以任何一次事务的完成都需要跨越城市 2 和城市 3，这会带来较大的延时。这时可以将这个 5 副本的集群降级为 3 副本的集群，对于 3 副本的集群来说只要有 2 个副本落盘即可，可以在城市 2 内的两个单元完成，从而大幅降低网络延时。

5) 业务扩容策略

随着支付宝业务的不断扩展，当单元内的资源不足以应对日常流量时，可以对各个单元内的资源进行扩容。应用服务器无状态，可以线性的扩展，扩展后按比例切流量到扩容服务器即可。如图 49 所示，OceanBase 数据库也可以支持平滑扩容，比如之前每个单元内有 2 台数据库服务器，形成一个 2-2-2-2-2 的集群，可以在每个单元内扩容 2 台服务器，形成一个 4-4-4-4-4 的集群，使得各个单元内数据库资源充足，满足单元内业务需要。

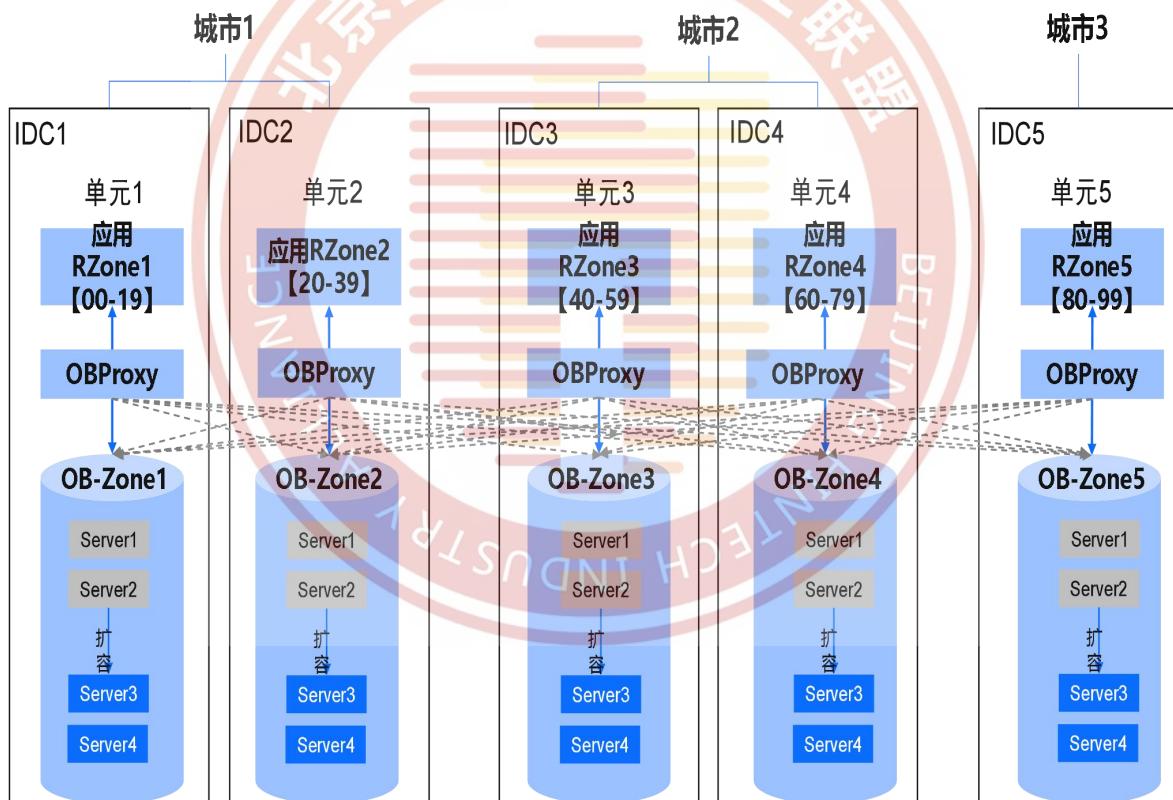


图 49 OceanBase 数据库单元内扩容示意图

扩容的服务器加入数据库集群后，集群会基于负载均衡的策略在单元内进行负载均衡。如图 50 的示例，单元 1 负责【00-19】

一共 20 个分片用户的数 据，扩容前，每台服务器各有 10 个分片用户的主副本，用来供应用服务器访问，同时每台服务器有 40 个分片用户数据的从副本来自别的单元的主副本提供备份服务。扩容后，基于负载均衡策略，旧服务器将自动迁移部分数据到新服务器，当新服务器上的数据追平后，部分从副本将切换为主副本。最终每台服务器可能有 5 个分片用户的主副本来供应用来读写，同时每台服务器可能有 20 个分片用户数据的从副本，来给别的单元的主副本提供备份服务。这个过程是集群自动完成的，无需管理员手工调整数据的分布。单元内数据库服务器的扩容并不影响整体分片的规则和流量路由的规则，单元 1 还是继续给

【01-19】分片的用户提供服务。

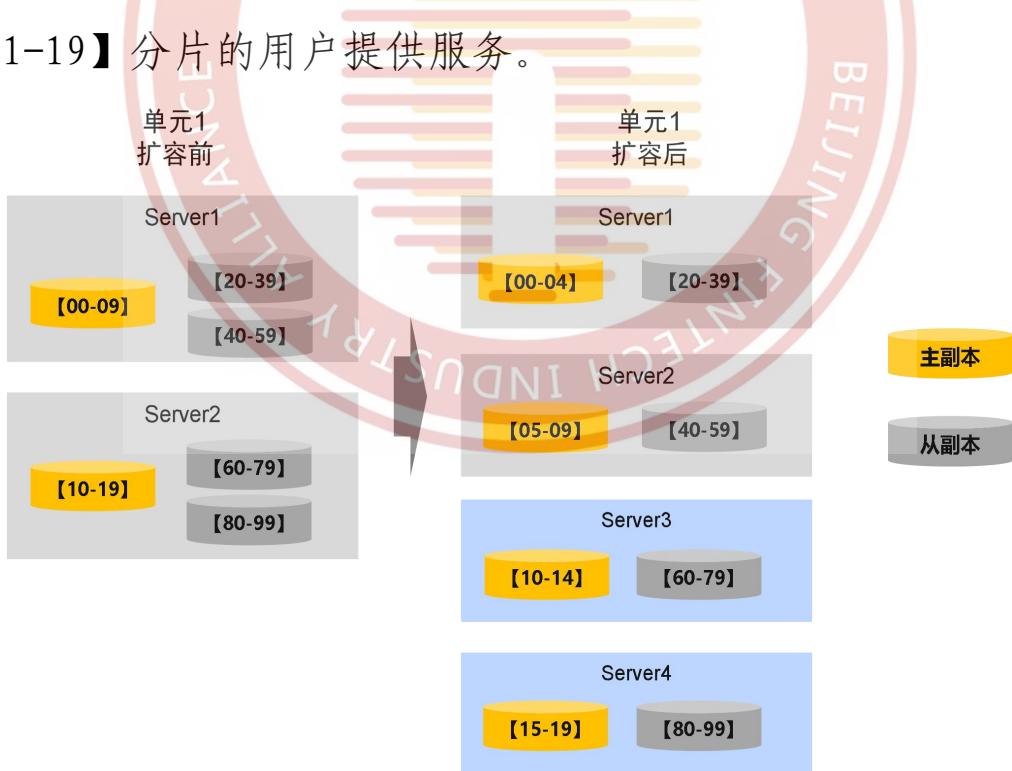


图 50 OceanBase 数据库单元内扩容后负载均衡示意图

支付宝为了应对双十一等促销活动的洪峰流量，如果扩容线下机房（现有单元）的话比较浪费，此时可以使用阿里公有云服务快速的扩容一个新的 Zone，并将部分单元在促销前弹出到扩容的新 Zone 中。

例如图 51 所示，原有的数据库集群是一个 3 个 Zone 的集群，Zone1 为单元 1 【00-39】分片的用户提供数据服务，Zone2 为单元 2 【40-79】分片的用户的提供数据服务，Zone3 为单元 3 的【80-99】分片的用户提供数据服务。扩容数据库集群，从 3 个 Zone 扩容到 4 个 Zone，并将【20-39】分片用户的数据由单元 1 弹出到单元 4。当然流量路由规则等配置也要同步修改，才能够将【20-39】分片用户的访问流量路由到扩容的单元 4 中。

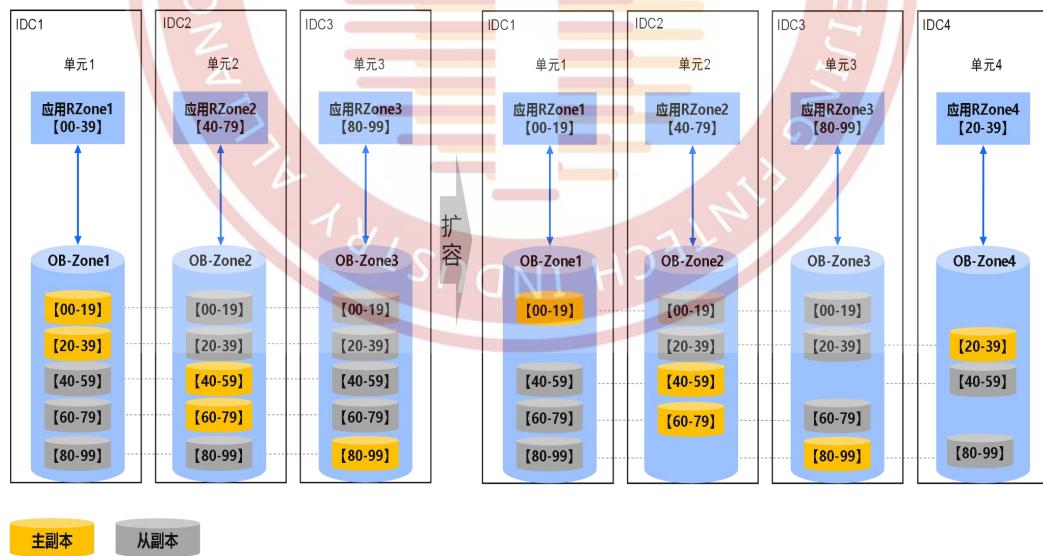


图 51 OceanBase 数据库扩容新单元示意图

管理员可以调整库表的 Locality 来实现上述扩容，比如

【20-39】相关库表的 Locality 需要首先在单元 4 内增加一个副本，等单元 4 内的从副本跟随单元 1 内的主副本追平数据后，再调整 Primary Zone，将 Zone4 的优先级设置为最高，从而将【20-39】分片数据的主副本汇聚到单元 4 内。OceanBase 集群将能够自动完成数据在单元间的传输和切主等操作。促销活动结束后，管理员在进行反向的动作，实现集群的缩容。

6) 试点业务跨单元数据操作与分布式事务设计

跨单元数据操作时会产生分布式事务，支付宝使用分布式事务中间件来保障在大规模分布式环境下业务活动的最终一致性，应用于交易、转账、红包等核心资金链路。分布式事务与服务框架、OceanBase 数据库以及消息队列等产品配合使用，实现服务链路级事务、跨库事务和消息事务等各种组合。

以支付宝三地五中心的部署结构为例，如图 52 所示，假设一个转账的业务场景。付款方用户 ID 是 12345666，分片号是 66，应该属于 RZ04；收款方用户 ID 是 54321233，分片号 33，应该属于 RZ02。付款方用户的请求会被路由到 RZ04 的收银台，RPC 框架层会自动识别业务参数上的分片位，将请求发到正确的单元。业务设计上，系统会保证交易流水号的分片位跟付款用户的分片位保持一致，所以绝大部分微服务调用都会收敛在 RZ04 内部。但是收款方的账号就刚好位于另一个城市的 RZ02。当交易系统调用账务系统给收款方的账号加钱的时候，就必须跨单元

调用 RZ02 的账务服务，图中这条跨城访问链路用红色表示。这时分布式事务的 ACID 保障由中间件来完成。

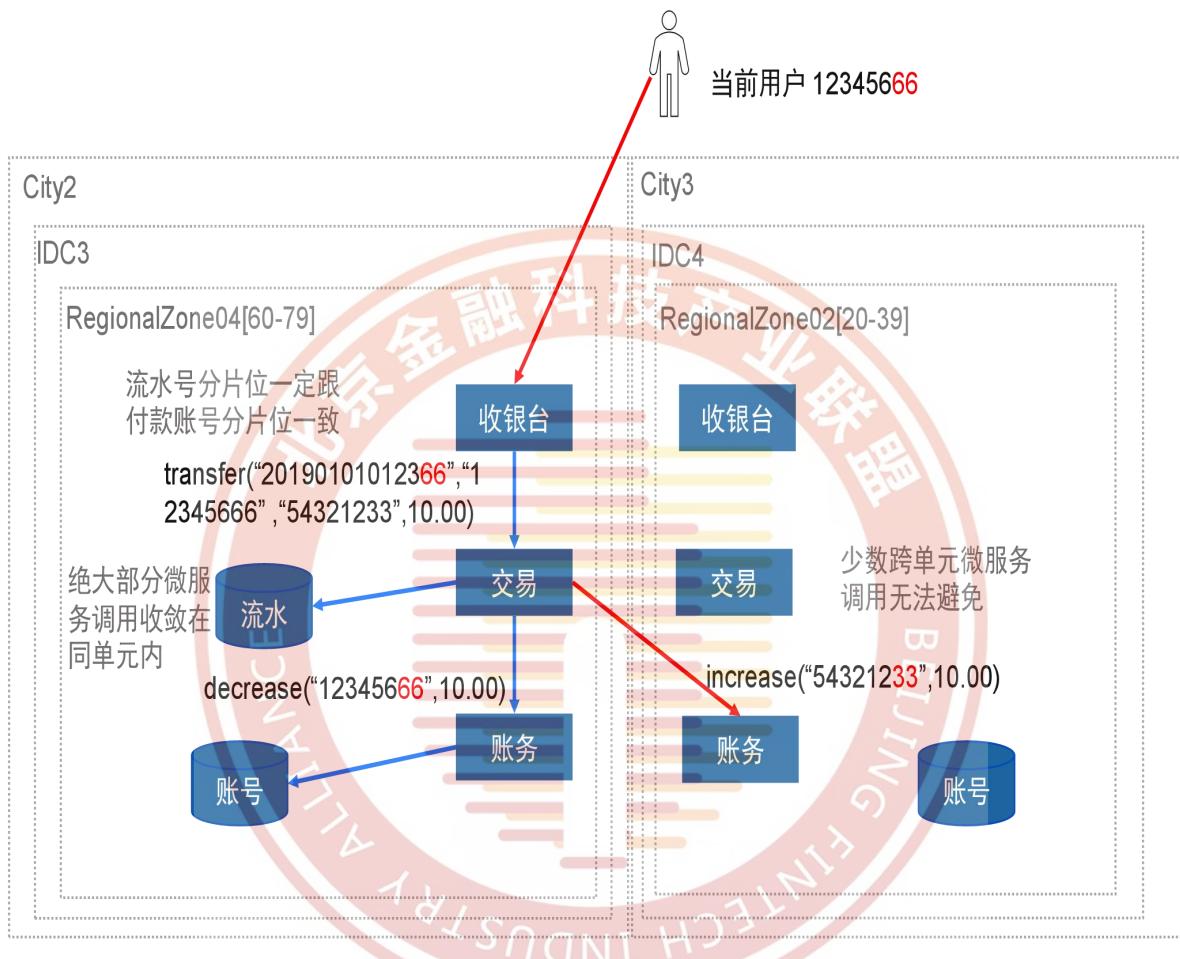


图 52 跨单元分布式事务示例图

3. 应用成果

支付宝试点业务已完成了单元化整体改造，在数据库、中间件及应用层都实现了自主研发。支付宝单元化的思想是单元内高内聚、单元间低耦合、跨单元调用无法避免，但应该尽量限定在

少数的服务层调用，大量的数据访问操作控制在同城，把整体耗时控制在可接受的范围内。支付宝通过单元化架构实现了平滑的扩容缩容、实现了蓝绿发布、实现了异地多活容灾，OceanBase分布式数据库在其中是核心的基础设施，为蚂蚁单元化改造提供了诸多核心能力：

无限可伸缩微服务架构：能够通过快速搭建一个业务完整的逻辑部署单元对系统进行整体扩容，对新机房扩容等操作带来巨大的便利。突破接入层、应用层和数据层的瓶颈，可支持无限扩展。

异地多活部署：除了具备异地容灾能力以外，还能做到异地多城市多活，可随时在多个城市间调配流量比例，在提升容灾能力的同时，降低了成本。

全站蓝绿发布和线上灰度仿真：通过多单元之间灵活的流量调配机制，可以实现大规模集群的蓝绿发布，极大的提升了发布效率。同时，通过单元内自包含的服务发现/路由和数据层的单元化分片，保证故障被切割的更小且具备独立性，不会传播到其他机房，从而实现发布时的故障自包含，支付宝基于这个机制实现了线上全链路压测和灰度仿真环境，为业务提供了更真实的验证环境，这对充分验证业务正确性，降低技术故障起到了关键的作用。