软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007442] [CSTR: 32375.14.jos.007442]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

分布式数据库高可用研究进展*

向清平, 燕 钰, 程思佳, 王宏志

(哈尔滨工业大学 计算学部, 黑龙江 哈尔滨 150001)

通信作者: 王宏志, E-mail: wangzh@hit.edu.cn

摘 要: 在分布式系统环境中, 数据库的高可用性面临诸多挑战, 如网络延迟、节点故障、数据一致性维护等问题. 这些挑战不仅要求先进的技术支持, 还需要灵活的架构设计和精密的管理策略. 高可用性不仅维护了数据库的数据完整性和一致性, 还在提升系统性能和增强容错能力等方面发挥了关键作用. 全面概述当前数据库高可用面临的挑战与相关问题; 强调重要的概念、理论以及相关的技术方法; 从系统与网络、数据与计算、应用与服务这 3 个层面对数据库高可用研究现状进行分析回顾和总结, 以深入探讨数据库高可用问题需要克服的挑战以及现有的解决方法, 并针对相关技术给出建议.

关键词: 高可用; 分布式数据库; 云数据库

中图法分类号: TP311

中文引用格式: 向清平, 燕钰, 程思佳, 王宏志. 分布式数据库高可用研究进展. 软件学报. http://www.jos.org.cn/1000-9825/7442.htm

英文引用格式: Xiang QP, Yan Y, Cheng SJ, Wang HZ. Research Progress on High Availability of Distributed Databases. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7442.htm

Research Progress on High Availability of Distributed Databases

XIANG Qing-Ping, YAN Yu, CHENG Si-Jia, WANG Hong-Zhi

(Faculty of Computer, Harbin Institute of Technology, Harbin 150001, China)

Abstract: In distributed system environments, ensuring high availability of databases poses multiple challenges, including network latency, node failures, and the maintenance of data consistency. Addressing these challenges requires not only advanced technical solutions but also flexible architectural design and refined management strategies. High availability plays a crucial role in maintaining data integrity and consistency, as well as in improving system performance and enhancing fault tolerance. This study provides a comprehensive review of the current challenges and issues associated with high availability in distributed databases. Important concepts, theoretical foundations, and technical approaches are examined, and the current state of research is analyzed across three levels: system and network, data and computing, and application and service. The study aims to deepen the understanding of the difficulties to be addressed and the existing solutions while offering recommendations for future research and technological advancements in the field.

Key words: high availability; distributed database; cloud database

传统基于集中式数据库在应对海量数据及复杂分析处理时, 存在数据库的横向扩展能力受限、数据存储和计算能力受限、不能满足业务瞬时高峰的性能等根本性的架构问题. 利用分布式计算和内存计算等新技术设计的分布式数据库能够解决上述遇到的性能不足等问题. 分布式数据库的数据分散在网络上多个互联的节点上, 数据量、写入读取的负载均衡分散到多个单机中, 集群中某个节点故障整个集群仍然能继续工作, 数据通过分片、复制、分区等方式实现分布存储, 每个数据节点的数据会存在一个或者多个副本, 提供数据冗余. 云计算技术的不断发展催生出将数据库部署在云上的需求, 通过云服务形式提供数据库功能的云数据库应运而生. 云与数据库的融合, 减

* 基金项目: 国家自然科学基金 (62232005)
收稿时间: 2024-10-18; 修改时间: 2024-12-28; 采用时间: 2025-03-25; jos 在线出版时间: 2025-10-15

少了数据库参数的重复配置, 具有快速部署、高扩展性、高可用性、可迁移性、易运维性和资源隔离等特点. 云原生数据库能够随时随地从多前端访问, 提供云服务的计算节点, 并且能够灵活及时调动资源进行扩容缩容, 助力企业降本增效. 以亚马逊 AWS、阿里云、Snowflake 等为代表的企业, 开创了云原生数据库时代. 未来, 数据库将深度结合云原生与分布式特点, 帮助用户实现最大限度资源池化、弹性变配、超高并发等能力, 更加便捷、低成本实现云上数字化转型与升级.

然而, 当前云原生数据库发展面临许多关键性问题, 高可用问题首当其冲. 随着“云原生+分布式”的不断发展, 数据库高可用问题的重要性呈现不断上升趋势, 已成为制约其发展的重要因素. 数据库系统会因不同原因而出现故障: 硬件故障、网络通信故障、软件错误、人为错误等. 高度可用的数据库系统确保即使面对此类故障, 系统仍保持运行且停机时间接近于零. 而近年来, 谷歌、微软、阿里等著名公司都有因可用性不足造成的严重宕机事件, 2022 年 6 月, 谷歌 Cloud 发生服务中断事件, 限制了用户访问云和相关项目[1]; 2023 年 5 月, 微软 Azure DevOps 在巴西的一处 Scale-Unit 发生故障, 导致宕机约 10.5 小时[2]; 2023 年 11 月, 阿里云出现严重故障, 全线产品受影响[3]. 因此, 需要保障业务的连续性, 即在用户眼里, 业务永远是正常 (或者说基本正常) 对外提供服务的. 数据库高可用的研究可以最大程度地减少数据中断和服务不可用的风险, 尤其是在金融交易、医疗系统等对服务不中断要求高的领域.

目前, 云原生数据库高可用问题已得到越来越多的关注, 许多企业组织和研究团体都启动了相关研究. 本文通过分析当前数据库高可用所面临的挑战以及研究现状, 提出未来数据库高可用技术的建议及重要的科研方向, 以期为我国未来高可用数据库的科研、产业发展做出有益的探索.

本文第 1 节介绍数据库高可用面临的挑战. 第 2 节从系统与网络、数据与计算、应用与服务这 3 个层面总结数据库高可用的研究现状. 第 3 节分析数据库高可用研究的挑战和未来的发展趋势. 第 4 节针对存储、参数配置、查询优化、事务方面对高可用技术提出建议. 第 5 节对本研究内容做出总结.

1 数据库高可用挑战

本节探讨数据库高可用性面临的主要挑战, 重点分析硬件和网络问题以及可用性和一致性之间的冲突. 硬件和网络的挑战部分介绍数据库系统常见的硬件故障以及网络故障, 可用性和一致性的冲突部分讨论数据库系统如何在高可用性和数据一致性之间取得平衡, 通过分析 CAP 理论以及可用性的类别, 阐明在一些场景下如何选择以确保系统在保持高可用性的同时尽量减少数据不一致的风险. 两部分阐述当下数据库系统高可用性主要面临的挑战以及在实际应用中需要综合考虑的因素, 以实现系统的最佳性能和可靠性.

1.1 硬件和网络的挑战

数据库的核心能力要求除了功能完善、使用方便以外, 还需要绝对安全、足够健壮, 能够满足“数据不能丢, 服务不能停”的要求. 传统的数据库产品中虽然有一些行之有效的软件技术用来提高数据可靠性和服务可用性, 但整体来说对硬件与网络的稳定性也有很强的依赖.

硬件故障是数据库系统常见的问题之一, 包括硬盘故障、内存故障、CPU 故障等. 磁盘是故障发生的主要来源[4], 电压变化等因素也可能引起电源故障. 此外, 当 CPU 和存储芯片过热时, 系统也可能发生故障, 很多系统并没有对其冷却进行任何形式的监控, 因此冷却故障让很多系统管理员措手不及. 服务器、磁盘、内存等硬件元件的故障可能导致数据丢失或服务中断, 为了弥补这些缺点, 现代系统配备有额外的风扇电源和优越的硬件诊断功能, 以便尽早发现和识别问题. 为了实现极致的高可用性, 往往需要严格测试硬件器件并且实现硬件冗余设计. 从第 2 节介绍的研究现状来看, 硬件冗余设计被广泛应用于现代数据库系统中, 尤其是在多节点系统中, 以保证即使某个节点发生故障, 系统仍能继续正常运行, 最大限度减少服务中断的风险.

在网络方面, 网络故障也是影响数据库高可用性的一个重要因素. 错误的路由信息、重复的主机名或 IP 地址以及错误解释广播地址的机器都可能导致出现网络错误, 网络中断、丢包、延迟等问题都会影响数据库系统的可用性, 可能导致数据库服务器无法与客户端通信, 无法完成读写等操作. 这些问题不仅影响单一节点的服务, 还可

能对整个集群的高可用性构成威胁, 尤其是在分布式数据库系统中. 这一问题将在第 2.1 节进行详细的探讨, 特别是关于如何通过网络优化和容错机制来保证数据库的高可用性. 在多数据中心环境中, 数据在不同地理位置之间的同步和一致性维护变得更加复杂, 不同数据中心之间的网络延迟和带宽差异也会影响数据库的高可用性.

1.2 可用性和一致性的冲突

为了正确地描述一个事务处理系统是否具有高可用性, 我们需要描述一个客户必须接触哪些服务器以及服务器能够提供哪些响应, 特别是考虑到中止的可能性. 传统上, 一个系统如果能在服务器之间存在任意、无限长的网络分区的情况下, 每个用户能够访问正确服务器且最终都能收到来自服务器的响应, 那么这个系统就被认为是高可用的.

CAP 理论包括强一致性、可用性和分区容忍性这 3 个属性. 强一致性是指在分布式系统中的同一数据多副本情形下, 对于数据的更新操作体现出的效果与只有单份数据相同. 可用性指客户端在任何时刻对大规模数据系统的读写操作都应该保证在限定的时延内完成. 分区容忍性指在大规模分布式数据系统中出现网络分区故障时系统仍然能够继续运行处理请求.

对于一个大规模分布式数据系统来说, CAP 三要素不可兼得, 同一个系统至多只能实现其中的两个. 例如, 在存在网络分区的情况下, 支持高可用性的系统就不能同时保证数据的强一致性. 分布式环境具有较高的不稳定性, 通常可能会出现两类故障: 节点故障和链路故障. 链路故障可能会导致网络分区, 即一个分布式系统的集群被划分为若干个区域. 在同一个区域中, 节点间可以通信, 跨区域的通信则出现困难. 当多个副本分布在不同的网络分区中时, 对一个副本的写入可能会无法同步到其他副本. 那么, 读取不同的副本将会返回不一致的结果. 因此, 由于分布式环境下存在的链路故障, 系统通常需要在读写操作的一致性和节点的可用性之间加以权衡.

文献 [5] 将可用性分为粘性可用性、事务可用性和副本可用性这 3 种, 可实现的高可用事务 (high availability transaction, HAT) 语句包括 ACID 隔离保证、ACID 原子性保证和会话保证. 在非粘性可用性的情况下, 系统必须处理在分区下出现不幸的客户被迫向分区、过时服务器发出请求的可能. 粘性可用性允许 3 个额外的保证: 读所写、PRAM (流水线随机存储)、因果一致性. 两种额外的高可用事务保证包括收敛性 (即最终一致性) 和一致性 (一个高可用性事务系统可以做出有限的应用级一致性保证).

文献 [6] 将系统可用性分为: 节点级别高可用、服务级别高可用和人工介入后可用. 节点级别高可用指当出现网络分区后, 所有节点都能继续对外提供服务; 服务级别高可用指当出现网络分区后, 只要分区的严重程度有限 (比如某个分区中存在多数派), 那么整个系统依然可以继续对外提供服务; 人工介入后可用指当出现网络分区后, 系统暂时不可对外服务, 只有在人工介入确认系统状态或采取某种补救措施后, 系统才能继续提供服务.

文献 [6] 提到, 当系统在操作一致性方面提供强一致性, 但不支持事务一致性时, 系统无法实现节点级别的高可用性. BigTable 是这种系统的一个典型代表. BigTable 主要由 3 部分构成: Chubby、Tablet Server 和 GFS. BigTable 将数据库划分为多个 Tablet, Chubby 将每个 Tablet 分配到唯一的 Tablet Server 上. 因此, 对 Tablet 的多次操作会由相同的 Tablet Server 服务, 从而保证了数据访问的强一致性. BigTable 能够实现服务级别的高可用性, 这主要得益于 Chubby 和 GFS 提供的高可用性保障. 当发生故障时, 虽然 Chubby 和 GFS 能够提供服务级别的高可用性保障, 但在网络分区, 单个节点上的 Tablet 数据可能无法立即恢复, 导致节点级别的可用性受限. 例如当发生网络分区时, 如果某个 Tablet Server 与 Chubby 之间的通信变得困难, 该 Tablet Server 会自动停止工作. Chubby 会认为其出现故障并指定一个新的节点来恢复 Tablet 数据, 继续提供服务. 此外, GFS 中的数据在多个节点上保存副本, 全局 Master 节点会监控节点故障并指定新的节点替换故障节点, 从而进一步确保系统的高可用性. 当系统提供较强的操作一致性和事务一致性时, 系统仅能在人工介入下实现高可用性. 这类系统包括以 VoltDB 为代表的 NewSQL 数据库以及传统的基于主从复制技术的关系型数据库. VoltDB 依赖两阶段提交协议 (two-phase commit, 2PC) 来协调多个线程共同完成事务执行. VoltDB 为每个数据分区维护了多个副本, 这些副本分布在多个物理节点上, 因此可以容忍部分节点故障. 然而, 系统本身无法区分节点故障和链路故障. VoltDB 依赖外部的网络分区检测工具来判定是否发生了链路故障. 如果检测到网络分区, 外部工具会仅保留拥有节点数量最多的那个网络分区, 并强制

关闭其他网络分区中的节点. 这种外部工具通常需要 DBA 进行控制和维护. 因此, VoltDB 只能在人工干预下提供高可用性.

一致性指的是系统中所有副本的数据保持一致的状态, 而可用性指的是系统能够正常响应请求的能力. 为提高系统的可用性, 通常会在一致性上做出让步. BASE 原则就是通过牺牲强一致性来获得高可用性, 即基本可用、软状态、最终一致性. 基本可用指系统在一定条件下保证提供基本的服务, 虽然不保证强一致性, 但可以在某些故障情况下提供部分可用性. 软状态指允许系统中的数据存在中间状态, 并认为该中间状态不会影响系统整体可用性, 即允许系统不同节点的数据副本之间进行同步的过程存在延时. 最终一致性指在一段时间内所有副本数据最终都会达到一致的状态. 如在 Amazon Dynamo 中, 系统放弃了强一致性要求, 而选择最终一致性模型. 这样一来, 系统能够在节点或网络出现故障时, 继续对外提供服务, 即使部分数据某些时刻可能存在不一致的情况, 但是最终能达到一致.

其余更强的一致性包括线性一致性 (强一致性)、因果一致性、会话一致性、单调读一致性、单调写一致性等. 线性一致性系统看起来只有一个数据副本, 只要一个客户端成功完成写操作, 所有客户端从数据库中读取数据必须能够看到刚写入的值. 因果一致性发生在进程之间有因果依赖关系的情形下. 会话一致性对于同一个用户会话的所有请求都能获得相同的结果. 单调读一致性保证进程读取到数据的某个版本, 那么后续所有读取操作都不会看到比该数据更早版本的数值. 单调写一致性保证多次写操作的序列化.

CAP 理论指出在网络分区的情况下, 系统需要在一致性和可用性之间做出权衡. 收敛性允许我们形式化地描述这种权衡, 不同强度的收敛性属性具体取决于系统的设计和需求. 文献 [7] 在权衡一致性与可用性引入了收敛性的概念, 其中的收敛性指一个实现能够确保一个节点发出的写被其他节点观察到的能力, 文中介绍两种不同强度的收敛特性, 最终一致性和两两收敛. 最终一致性指存储系统保证如果没有对对象进行新的更新, 最终所有访问都将返回最后更新的值. 两两收敛指两个节点对于同一个对象的读返回相同的结果, 那么这两个节点两两收敛. 如果一个系统中的任意两个节点没有发出任何写操作, 也没有接收到其他节点的消息, 那么最终会这两个节点会交换一组消息, 接收这些消息使节点两两收敛, 那么这个系统是两两收敛的.

2 数据库高可用研究现状

第 1 节介绍了数据库高可用性面临的主要挑战, 包括硬件故障、网络问题以及在分布式环境中可用性与一致性之间的冲突. 具体而言, 硬件故障 (如磁盘、内存和 CPU 故障) 和网络问题 (如路由错误、延迟和分区) 直接影响数据库系统的稳定性和可用性. 此外, 在分布式系统中, 如何在高可用性与数据一致性之间做出平衡, 也是一个长期存在的难题. 本节将详细回顾现有研究如何应对这些挑战, 并总结相关的技术进展. 本节总结数据库高可用性研究的现状, 从系统与网络、数据与计算、应用与服务这 3 个层面进行全面分析. 系统与网络层面重点探讨数据中心的可用性系统和网络架构对高可用性的影响; 数据与计算层面分析存储与冗余备份、事务管理以及负载均衡的关键技术; 应用与服务层面深入讨论查询处理、中间件架构以及虚拟化的相关技术和方法在高可用性保障中的应用. 这些层面的综合分析全面呈现当前数据库高可用性的多维度研究视角和解决方案, 为后续优化提供理论基础和实践参考.

2.1 系统与网络

本节从数据中心和网络两个层面展开分析系统的可用性. 在数据中心层面, 讨论单主系统和多主系统的可用性, 分析这两种架构在故障转移中的问题和优势, 并探讨数据库高可用性对基于云的应用程序和服务的影响; 在网络层面, 主要从 RDMA (远程直接内存访问) 和网络拓扑结构的角度进行深入分析, 阐明网络架构对系统可用性的关键作用.

2.1.1 数据中心层面下的可用性系统

为了确保数据中心级别中断时的高可用性, 不同类型的系统设计在服务等级协议、一致性保证、资源使用等方面都有不同的权衡[8].

对于单主系统, 主要在单个数据中心中运行. 主备系统是这样的设计: 在无共享环境中运行, 许多数据库系统实现某种形式的主备高可用. 在某些情况下, 主数据库和备份数据库可以在主动-主动配置中运行, 从而允许对从数据库执行一些只读应用程序工作, 而从数据库相对于主数据库可能稍微过时. 在主备系统中, 更新传播可以是物理的、逻辑的 (基于行) 或基于语句的, 可以是同步的或异步的. 在同步的情况下, 直到主系统和备用系统都持久记录更新后, 数据库客户端才会确认事务提交, 从而形成所谓的两阶段安全系统[9]. 两阶段安全系统可确保单个服务器故障不会导致更新丢失, 但同步更新传播可能会带来大量性能开销. 相反, 异步传播允许事务在主节点提交后立即得到确认. 这在正常操作期间产生的开销要少得多, 但如果主数据库发生故障, 一些最近提交的事务可能会丢失.

对于基于故障转移的系统, 处理仍然在单个主数据中心进行, 但捕获已处理内容的关键系统状态将作为检查点复制到备用数据中心. 如果主数据中心发生故障, 处理可以转移到备用数据中心并从最新的检查点重新启动. 对于大型复杂系统, 故障转移过程 (具有同步或异步检查点) 往往非常复杂且存在风险, 复杂的系统有许多组件和许多依赖项, 所有这些都需要重新启动并可能重新配置. 此外, 随着系统的发展, 故障转移脚本往往随着每次变化而更新, 使得基于故障转移的系统本质上具有较高的维护开销.

对于多宿主系统, 设计为在多个数据中心中运行. 基于故障转移的系统的许多问题源于故障转移通常是作为附加功能构建在固有的单宿主设计之上. 相比之下, 多宿主系统故障转移概念使得每个数据中心始终处理工作, 并且工作在数据中心之间动态共享以平衡负载. 当一个数据中心速度缓慢时, 部分工作会自动转移到速度更快的数据中心. 当一个数据中心完全不可用时, 其所有工作都会自动分配到其他数据中心. 除了持续动态负载平衡之外, 没有任何故障转移过程. Google 的团队[8]通过多年多代的系统设计构建, 认为构建本机多宿主系统在大多数情况下是最佳解决方案. 多宿主系统以更好的可用性和更低的成本运行, 并导致整体系统更加简单, 应用程序开发人员可以直接获得高可用性和一致性, 并可以专注于构建应用程序. F1[10]是 Google 构建的容错全球分布式关系数据库, 其构建在 Spanner[11]之上. Spanner 是一个完全多宿主的系统, 使用 Paxos[12]协议进行同步更新, 通常部署具有 5 个副本的 Spanner 实例, 因此最多可以丢失两个数据中心, 5 个副本被认为是高可用性的最低限度, 因为在一个数据中心完全中断期间, 在任何剩余数据中心发生机器级故障或暂时故障期间仍必须保持可用性. Photon[13]是一个地理分布式系统, 构建在 Paxos 协议上, 旨在容忍基础设施退化和数据中心级中断, 在谷歌广告系统中的关键应用之一是连接多个数据中心的数据流, 并生成连接日志, 鉴于输出连接日志的业务关键性质, Photon 通过 Paxos 协议确保即使在数据中心级中断的情况下, 系统仍能继续提供服务. Mesa[14]是一个高度可扩展的分析数据仓库系统, 旨在以每秒数百万次的更新速度在 PB 级表中提供一致的事务更新, 并使用多宿主来提供高可用性.

数据库的高可用性 (high availability, HA) 对于基于云的应用程序和服务的高可用性至关重要, 长期以来采取主从复制来解决此类问题, 但是主从复制采用异步或半同步复制, 当其在事务中崩溃时, 可能会遇到数据不一致问题, 现代通过基于集群的多主同步复制解决这个问题[15]. HA 数据库通常通过使用集群的多主架构来实现, 其中多个服务器节点组合在一起. 集群中的任何节点都可以响应读取和写入请求. 节点中的数据更改会立即复制到集群中的所有节点, 从而提供系统冗余并最大限度地减少停机的可能性. 这种架构中复制是同步的, 如果集群的一个节点发生任何更改, 该更改也会同时应用到其他节点. 因此, 即使一个节点发生故障, 其他节点也会继续工作, 因此故障不会造成重大后果. 当故障节点稍后再次加入集群时, 数据将从现有节点复制, 从而保持数据的一致性.

随着云计算的快速发展, 云数据库的应用逐渐成为主流, 而传统云数据库通常仅支持一写多读, 这种架构在高并发写入场景下容易成为瓶颈, 无法满足大规模、低延迟的业务需求, 因此它需要能够横向扩展写入能力的云原生多重写入技术. 近年来出现了支持多写的数据库系统, 如 Taurus MM[16]采用了共享存储架构, 通过优化算法减少网络流量, 从而提高了写入吞吐量; PolarDB-MP[17]则利用分布式共享内存和共享存储, 支持多主节点并通过 RDMA 优化了数据同步和事务协调. 这些系统通过横向扩展写入能力, 显著提升了云数据库的高可用性和性能. 还有一些采用共享缓存技术的数据库系统, 如 Oracle RAC 和 IBM DB2 pureScale, 它们通过全局锁管理器和分布式缓存一致性协议, 实现多个节点之间的数据共享与事务一致性. 此外, 为了提供高可用性 (例如零停机时间) 的云数据服务, 需要能够利用多云部署能力的多云数据库, 目前有效确定数据放置策略并在多个云之间高效迁移数

据仍具有挑战性.

2.1.2 网 络

高可用性通常通过分布式数据复制来实现, 其中每个数据库记录驻留在一个主副本以及一个或多个备份副本中. 对主副本的更新同步传播到所有备份副本, 以便任何发生故障的主服务器都可以由备份服务器替换. 分布式系统设计的传统观点认为网络是严重的性能瓶颈. 主动-被动和主动-主动这两种主要的高可用性方法通常都采用最小化网络开销作为优化目标. 主动-被动复制是最常用的复制技术之一. 每个数据库分区都包含一个活动副本 (称为主副本) 和一个或多个备份副本, 前者处理事务并对数据进行更改, 后者使副本与主副本保持同步. 在学术项目[18−21]和商业数据库 (如 Postgres Replication[22]、Oracle TimesTen[23]和 Microsoft SQL Server Always On[24]) 中, 主动被动复制有许多不同的实现. 主动-被动方案通常通过日志传送来实现, 其中主数据库执行事务, 然后将其日志传送到所有备份副本. 备份副本重播日志, 以便新的更改反映在其副本中. 采用主动-主动协议[25]的系统允许任何副本接受事务, 然后将更改广播到所有其他副本. 由于每个副本的事务与其他副本之间可能存在冲突, 因此在副本之间同步更新需要比主动-被动更多的协调. 传统的主动-主动数据库通过消除协调并用副本之间的执行顺序的确定性取代协调来解决这个问题 (如 H-store[26]、VoltDB[27]和 Calvin[28]), 确定性主动-主动复制旨在减少在主动-被动方案中传送日志以及与其他副本协调所需的网络通信. 此外, 数据库不仅依赖于通过副本间执行顺序的确定性来降低协调需求, 还在数据结构层面引入了免冲突技术, 使得副本更新过程能够以无锁、高效的方式进行, 能够在一定程度上减少网络通信的开销. 例如, Berkeley 的 Anna 系统采用了基于格结构 (lattice-based structure) 的协调自由模型, 使得不同副本之间的状态更新无须显式同步即可保持一致[29]; Starry 系统则在多主架构中引入半领导者协议, 通过将冲突事务和非冲突事务分开处理, 既提高了高并发环境下的性能, 又减少了事务冲突的中断率[30]; SLOG 系统结合数据本地性和确定性执行框架, 通过在本地事务与跨区域事务之间建立灵活的动态处理路径, 既保证了严格的事务一致性, 又在多区域环境中实现了低延迟[31].

然而, 随着下一代网络的兴起, 尤其是在局域网环境中, 例如, 最新的基于远程直接内存访问 (RDMA) 的网络可实现与主内存相似的带宽, 性能瓶颈已从网络转移到 CPU 的计算开销, 传统的网络优化方案可能不再是最佳选择. 目前存在新的协议 Active-Memory[32]来充分释放 RDMA 网络的潜力, 这是一种专为局域网设置中的下一代 RDMA 网络而设计的新高可用性协议. Active-Memory 的优化目标是最小化执行数据复制的 CPU 开销, 而不是最小化网络流量. Active-Memory 的核心思想是利用 RDMA 的单向特性, 直接更新远程备份服务器上的记录, 而不涉及远程 CPU.

网络拓扑对高可用也会产生一定的影响. 网络拓扑是网络内节点的排列, 基本网络拓扑有总线、环形、星形、网状、树形和混合拓扑, 不同网络拓扑的特点如表 1 所示. 总线拓扑通过单根电缆连接所有计算机和网络设备, 沿一个方向传输数据. 在这种结构中若电缆出现故障, 整个网络就会出现故障; 环形拓扑中, 计算机系统以环形结构相互连接, 这种结构当网络流量大时或增加额外节点都不会对其可用性造成影响, 但一台计算机系统的故障可能会影响整个网络的可用性; 星形拓扑借助电缆将所有计算系统连接到单个集线器, 该集线器用作中心节点, 如中心节点发生故障, 整个网络就会停止, 但是这种拓扑结构提供快速的性能, 更易于故障排除、设置和修改; 网状拓扑中, 所有节点相互连接使得整个结构比较稳健, 易于诊断故障, 可用性高; 树形拓扑包含根节点, 所有其他节点都连接到它, 易于管理和错误检测; 混合拓扑是两个或两个以上拓扑的组合, 提供可靠性、可扩展性和灵活性. 以上基本拓扑结构衍生出的常用于云计算数据环境的拓扑结构还包含 Fat tree 拓扑、FT-RUFT-212、FT-RUFT-222 等结构, 它们也都具有很好的容错性与可用性[33].

此外, 基于 RDMA 的存储系统在提高可用性方面也有一些优势. 典型的键值存储系统包括基于哈希的键值存储和基于树的键值存储, RDMA 能够用于构建快速且持久的分布式键值存储系统, 在数据传输和存储过程中, 数据丢失的风险降低, 增强了系统的可靠性和持久性. 现有文件系统架构可分为解耦架构和耦合架构两类. 解耦架构将存储和网络设备解耦, 类似于传统的使用块设备的分布式文件系统, 耦合架构将存储设备和网络设备耦合起来, 文件系统可以通过 RDMA 网络直接访问远程持久内存, 减少了网络通信的开销, 大幅降低了延迟和 CPU 消耗, 提高了系统的可用性; 随着 RDMA 网络性能的提高, 基于 RDMA 的分布式存储系统可以提升内存处理应用程序的

性能; 对于数据库系统来说, RDMA 同样可以提高数据库系统各个模块的性能, 包括存储引擎、并发控制、事务处理和查询执行等. 通过减少数据传输的延迟和 CPU 负载, RDMA 使得数据库系统能够更快速地处理事务和查询, 减少了等待时间和系统因处理时间过长发生死锁、资源竞争或超时错误的可能性, 从而提高系统的可用性和可靠性[34].

表 1 不同的网络拓扑的特点

拓扑结构	特点	优势	不足
总线型拓扑	通过单根电缆连接所有计算机和网络设备; 沿着一个方向传输数据	成本低; 需要的电缆少; 适用于小型网络; 易于扩展	当电缆发生故障时, 整个网络都会发生故障; 当网络流量很大或添加更多节点时, 网络性能会下降; 电缆长度有限; 传输速度慢于其他结构
环形拓扑	计算机系统以环形结构相互连接	网络传输不会因节点多或流量高而受到影响; 易于扩展; 成本低	添加或删除计算机节点可能扰乱网络活动; 一台计算机节点出现故障会影响整个网络
星形拓扑	用电缆将所有计算系统连接到单个集线器; 该单个集线器作为中心节点	当计算机节点少和网络流量低时性能很好; 易于升级集线器; 易于排除故障; 易于修改和设置	安装和使用成本高; 当中央集线器发生故障时整个网络将停止工作; 整个系统性能取决于集线器
网状拓扑	所有节点相互连接; 系统结构稳健; 易于诊断故障	每根电缆都可以传输自己的数据负载; 结构稳健性好; 故障诊断容易; 提供隐私和安全	难于安装和配置; 布线成本高且需要大量布线
树形拓扑	结构中包含根节点, 其他节点连接到根节点; 易于管理和错误检测	是总线拓扑和星型拓扑的扩展; 易于扩展节点; 易于管理和维护; 易于检测错误	布线成本高; 多个节点维护困难; 当中央集线器发生故障时整个网络都会发生故障
混合拓扑	2个或2个以上的拓扑结构的组合	可靠、有效、可扩展、灵活	设计复杂、成本高

2.2 数据与计算

本节主要从存储与冗余备份、事务和负载均衡方面展开讨论. 对于存储与数据备份层面, 本文分析数据库系统可能遇到的灾难和问题并列举一些容灾方法技术. 对于数据层面, 本文主要从基础理论、事务调度、确定性数据库系统这 3 个方面展开讨论. 对于负载均衡, 本文主要介绍云环境中的一些负载均衡算法及其应用.

2.2.1 存储与冗余备份

在大数据时代, 随着数据量的增长, 用户对云服务的需求也随之增加, 云服务提供商需要在数据涌入的情况下维持系统可用性. 复制可以在云计算中确保跨服务器的数据可访问性、安全性和可用性, 但过多的副本会增加存储成本, 而重复数据删除可能会损害对于云服务至关重要的可用性和可靠性. 文献 [35] 尝试在重复数据删除和复制之间取得平衡, 在不增加过多费用的情况下实现高效、高可靠的存储系统. 其提出名为 FASTEN 的新型云存储数据分散方案, 用于平衡“重复数据删除”和“复制”, 包括容错子集和服务器评级算法, 前者组织数据块以获得最大的容错能力, 后者则根据用户定义的冗余选择最佳服务器.

在大规模系统中, 如部署上千个节点遍布全球的多数据中心, 硬件故障和网络中断不再罕见, 而是成为两个主要故障源. 对于微信这样的大规模系统, 其存储系统除了需要满足大数据的 3V 特性 (容量 (volume)、速度 (velocity)、多样性 (variety)) 之外, 高可用性也是存储服务需要重点考虑的. 可用性对于用户体验至关重要, 微信中的大多数应用程序都要求延迟开销低于 20 ms, 因此, 微信团队为支持微信综合业务开发了高可用性存储系统 PaxosStore[36]. 它在存储层采用组合设计, 针对不同存储模型构建多个存储引擎, 将基于 Paxos 的分布式共识协议提取为底层多模型存储引擎通用访问的中间件, 同时 PaxosStore 的无租约 Paxos 实现有利于快速故障转移, 而不会导致系统服务停机.

随着云计算和大数据应用的普及, 传统文件系统在处理大规模数据和高性能需求方面面临挑战. PolarFS[37]利用新兴的硬件和最先进的优化技术, 例如操作系统旁路和零拷贝, 使其具有与 SSD 上的本地文件系统相当的延迟, 通过开发一种新的共识协议 ParallelRaft, PolarFS 在不牺牲存储语义一致性的情况下放松了 Raft 严格的顺序写入

限制, 从而提高了 PolarFS 并行写入的性能. 此外, PolarFS 在用户空间实现了类似 POSIX 的接口, 这使得 POLARDB 可以通过较小的修改实现性能提升. 这些特点使 PolarFS 能够在大规模数据处理和高性能需求下依然保持高效运行, 在可用性方面表现出色.

在大数据和云计算的背景下, 传统的数据库系统在处理大规模数据时常常遇到性能瓶颈, 尤其是在需要快速查询和数据分析的场景下. PolarDB-IMCI[38]作为一种云原生 HTAP 数据库, 通过先进的 OLAP 性能和最小化对 OLTP 的干扰, 优化了可见性延迟以提高数据的新鲜度, 其采用内存列索引作为补充存储来加速分析查询, 并引入了 CALS 和 2P-COFFER 两项关键技术来高效传播更新. 此外, PolarDB-IMCI 利用共享存储上的检查点来增强计算资源的快速横向扩展, 并通过新的查询优化流程提高了复杂查询的处理效率. 这些技术的结合不仅提升了系统在处理大规模数据时的性能, 还大幅提高了系统的高可用性. 通过优化写入和查询性能、减少延迟和资源消耗, PolarFS 和 PolarDB-IMCI 能够在大数据和云计算环境中提供更可靠、更高效的服务, 确保系统在面对大量并发访问和数据处理需求时依然能够保持高可用性.

互联网规模的应用程序通常分层在数据中心 (data center, DC) 中运行的高性能分布式数据库引擎之上. 最近的一个趋势是使用跨多个 DC 的地理复制来避免用户跨区域访问数据, 用户可以在当前区域访问所需数据, 从而减少了因跨区域访问带来的延迟和网络不稳定性. 这种情况对分布式数据库提出了很大的挑战. 为解决这一问题, 文献 [39] 中引入了 Cure. Cure 是一个在保持高可用性的同时提供强一致性语义的分布式存储系统, 主要通过因果一致性、原子性以及对高级数据类型的支持来实现这一点. Cure 提供了一种新颖的编程模型: 因果+一致性, 并通过交互事务接口提供 CRDT (高级复制数据类型) 支持. Cure 提供了与可用性保持一致的最高水平的保证. 这些保证包括: 因果一致性 (无有序性异常)、原子性 (一致的多密钥更新) 以及支持安全解析并发更新的高级数据类型 (开发者友好的 API) (保证收敛). 对 Cure 的评估结果表明, 当将 Cure 与现有的在不同工作负载下提供相似但较弱语义的因果一致系统进行比较时, 它表现出更高的性能, 同时实现了更好的更新可见性延迟和对完全 DC 和网络故障的容忍度. 此外, GeoGauss[40]系统也使用 CRDT 并实现副本顺序一致性. 在 GeoGauss 中, CRDT 确保了即使在不同数据中心之间的网络延迟和分区情况下, 也能保持数据一致性, 并且通过副本顺序一致性提供高可用性.

随着互联网技术的快速发展, 数据库系统可能遇到各种类型的灾难, 数据备份、数据恢复等大规模在线服务越来越多. 容错对于一个系统的高可用性来说是至关重要的, 以便允许它即使在组件故障或一个或多个故障存在的情况下也能提供所需的服务. 导致数据丢失的原因可能有自然灾害、应用程序故障、网络故障、网络入侵、黑客攻击、系统故障甚至是人为错误等[41]. 灾难恢复包括 3 种类型: 热备份站点、温备份站点和冷备份站点[42]. 热备份站点计算机配置并配备了一系列软件和数据, 以在主服务器关闭时接受生产负载. 故障转移通常 (如果需要) 通过集群配置获得. 备用集群配置是独立的, 并且与主数据库配置有区别. 温备份站点计算机硬件已预先配置并提供了一系列软件. 一旦灾难发生, 域名系统就会切换并重定向到备份站点, 服务器承担生产负载. 必须手动重新启动服务. 在冷备份站点中, 计算机的硬件元素需要一组与要生成或恢复的数据集相关联的软件, 然后才能将系统提升到生产状态.

一些研究给出了数据复制、备份实现高可用性容灾和灾难恢复. Pokharel 等人[43]完成的云系统灾难恢复的地理冗余方法 (geographical redundancy approach, GRA) 通过采用马尔可夫模型对 GRA 进行分析实现了高可用性, 同时保持了低停机时间和低成本. Sengupta 等人[44]提出了多站点灾难恢复数据分发计划 (data distribution plan for disaster recovery, DDP-DR), 根据保护级别和防止约束提供不同的数据分发计划从而解决了数据备份驻留在多个分布式位置的多站点架构中的灾难恢复问题. Saquib 等人[45]针对云计算系统中的数据库应用程序提出了一种新模型. 该模型提供了一种零数据丢失和快速恢复的灾难恢复解决方案, 利用同步技术进行数据复制, 以确保最小的恢复点目标 (recovery point objective, RPO) 和恢复时间目标 (recovery time objective, RTO). Lenk 等人[46]提出一种数据部署方式利用云备份灾难恢复在云中进行热备份, 恢复时间明显缩短.

容错机制在满足可用性需求方面起着至关重要的作用. 人们提出了多种可用性模型来评估计算系统, 其中贝叶斯网络模型由于其强大的建模形式而在工业界和研究中得到了广泛的应用. 文献 [47] 引入一种高层次的建模方法来自动构建贝叶斯网络, 解决了利用贝叶斯网络评估大规模冗余和复制服务可用性的实际建模挑战. 贝叶斯

网络模型统一了在服务的高层模型描述中定义的故障方面. 高层模型由 3 个子模型组成: 故障依赖图、网络模型和容错需求模型. 故障依赖图用于表达基础设施和执行环境组件之间的故障关系, 故障依赖图被转化为贝叶斯网络. 网络模型用于解决通信和网络分区故障. 容错需求模型用于定义服务的容错需求. 在存在可能的网络故障的情况下, 评估两个实例之间的可达性. 从可用性的角度来看, 当某条路径发生故障时, 由于该条路径上的某个网络组件发生了故障, 如果仍然存在, 则可以沿着不同的路径建立一个通道. 因此, 当所有潜在的路由都失败时, 一个通道被认为是不可用的. 在贝叶斯网络基础上可以使用冗余服务模型扩展. 在现实生活中, 一个客户端应用程序最可能会尝试连接到一个实例, 贝叶斯网络表示连接到其中任何一个实例的概率. 对于复制服务, 客户端首先向一个实例发送请求, 然后与剩余的实例进行通信. 这种通信模式包含并实现了访问至少一个能够与足够多的剩余实例通信的实例的可能性. 通过评估实例验证了贝叶斯网络方法在表示和评估具有数百个故障影响和服务实例的大规模服务可用性方面的可行性.

2.2.2 事　务

事务管理对系统高可用性有着复杂的影响, 尤其是在分布式系统中. ACID 事务保证了数据的一致性和可靠性, 但强隔离级别 (例如可串行化) 可能需要严格的同步机制, 这会导致性能瓶颈和高延迟, 尤其是在发生网络分区或故障时, 这种严格的同步要求可能会影响系统的可用性, 因为在部分系统出现故障时, 事务可能会被阻塞或无法完成, 从而影响整个系统的响应能力. 为了提高可用性, 许多系统在事务管理中采用高可用事务设计、降低事务的隔离级别、优化事务调度和确定性事务等策略, 通过放宽一致性要求或灵活处理事务提交来保证系统在部分故障情况下依然能继续运行, 合理的事务调度可以减少冲突和延迟, 增强系统的并行性和响应能力. 因此, 系统需要在事务一致性和高可用性之间找到平衡, 采用灵活的设计来确保在各种故障条件下系统能够持续提供服务.

为了最大限度地减少网络延迟并在服务器故障和网络分区期间保持在线, 许多现代分布式数据存储系统通常牺牲部分事务保证来提升系统的可扩展性和可用性. 文献 [5] 考虑提供 HAT 的问题: 事务保证不会在系统分区期间出现不可用或导致高网络延迟. 分析现有的 ACID 隔离和分布式数据一致性保证, 以确定哪些可以在 HAT 系统中实现, 哪些不能实现. 通过分析和实验来量化 HAT 的可用性和性能优势, 将许多先前提出的数据库隔离和数据一致性模型与高可用性的目标联系起来, 这保证了每个非故障服务器在它们之间存在任意网络分区的情况下都能做出响应. 文中还对一系列可以实现高可用性的模型进行了分类, 将它们表示为 HAT. 其调查表明, 除了可串行性之外, 快照隔离和可重复读取隔离不符合 HAT, 而大多数其他隔离级别都可以通过高可用性来实现. 文献 [48] 研究了 CAP 定理和 ACID 事务之间的关系, 即在存在网络分区的情况下是否可以使 ACID 事务高可用, 并证明了事务与高可用性并不冲突, 关键在于事务的实现方式. 如全局锁管理需要多个节点协调, 导致网络分区时事务无法执行, 降低可用性; 严格的线性一致性也要求所有副本严格同步更新, 可能因某个副本不可用导致整个系统不可用. 因此, 提出了 HAT, 它始终提供高可用性并且通常提供低延迟.

对于事务调度, 文献 [49] 研究如何提高主存多核 OLTP 系统执行有冲突事务的性能. 其开发了一种有效的调度算法来提高并行性并减少运行时冲突, 而不是基于传统的冲突概念进行分区. 基于运行时的冲突分析, 通过动态调整事务的执行顺序来减少冲突, 采用动态推迟可能导致冲突的事务来优化事务调度. 在事务缓存方面, 文献 [50] 研究了事务缓存策略. 与关注延迟的传统缓存不同, 事务缓存主要用于增强数据库系统, 并通过将读取负载卸载到缓存上来提高其事务吞吐量. 对于一大类基于批处理的事务系统, 可以通过事务一致性感知缓存策略来打破传统缓存策略的理论性能障碍. 其开发了一个一致的缓存策略, 理论上在常见的缓存方案下具有竞争力, 进一步利用批处理, 可以对批次内的交易进行重新排序, 同时保证每个交易看到的数据值具有有限的陈旧性. 批处理的事务调度和事务缓存策略为事务层面的“始终在线”以及低延迟提供了广泛的优化空间.

为了推动在易于分区的环境中通过高可用性和低延迟实现的事务保证, 文献 [48] 提出了分区存在时可用的 HAT, 其支持对事务序列的读和写操作的许多理想的 ACID 保证, 允许低延迟操作. 其中提到了两种产生中止的情况, 内部中止和外部中止. 事务自身选择的中止为内部中止 (如, 事务本身的操作或由于可能违反声明的完整性约束); 由外部引起 (如由数据库服务器的可用性和配置引起的) 的中止称为外部中止. 副本可用性、事务可用性和高可用性三者之间存在联系. 如果一个事务可以为它试图访问的每个项至少联系一个副本, 称作副本可用性; 如果系

统中的每个事务最终提交或者内部中止, 那么我们称这个系统提供事务可用性; 如果一个系统给定副本可用性, 提供事务可用性, 则称这个系统提供高可用性. 在事务隔离层面, 可串行化与高可用性矛盾, 大多数数据库提供弱隔离性而非可串行化.

近年来, 确定型数据库系统越来越受到数据库研究界的关注. 但确定性分布式事务处理系统中确保高可用性的问题很少受到关注, 文献 [51] 设计并实现了一个面向队列的复制事务处理系统 QR-Store. 该系统提供了复制层的两种实现方法. 一种实现方法是中间件复制使用 ZooKeeper 作为中间件来实现复制层. Leader 服务器和副本服务器充当 ZooKeeper 集群的客户端. 当 Leader 服务器发出写请求时, 副本服务器发出读请求以获取复制的数据. ZooKeeper 集群是一个高可用的系统, 它不会构成单点故障, 因为它使用内部复制和共识协议来确保正确的故障转移. ZooKeeper 使用的一致性协议称为 ZAB, 该方法简化了复制层在服务器节点的实现但是增加了中间件的开销. 另一种实现方法是集成复制, 这种方法将基于 quorum 的复制协议与事务处理协议相融合, 有效地消除了中间件开销. 采用这种实现方式, 领导者服务器节点向副本发送复制的数据消息. 在接收到复制的数据消息后, 副本以确认消息的形式回复到领导服务器. 根据所能容忍的节点故障的数量, 存在最少数量的确认消息来确认成功的复制. 令节点故障次数记为 f, 使得给定节点的副本总数为 $n=2f+1$. 确认消息的个数为 $f+1$. 这种方式可以确保系统的高可用性同时减少开销.

电子商务、金融科技和云应用等领域对高并发交易量的需求日益增加. 这与多核机器的日益占优凸显了追求更高的多线程事务处理的吞吐量和并行性的需求. 由于读取和写入相同数据项的竞争操作, 事务执行必须防范并发异常, 以维持所需的隔离级别. 事务在冲突时的高效处理对于数据库高可用性很重要. 文献 [49] 在事务分区的基础上, 考虑调度事务的执行顺序来增加并发度. 为此, 将分区重新写入捆绑事务的调度来对事务的执行顺序进一步细化. 此外, 对于并发控制协议所针对的非捆绑交易, 提出了一种主动延迟策略来减少连带惩罚. 分区数据库 (和变体) 在静态可分区的高争用工作负载上表现良好, 但时变工作负载通常使它们不切实际. 为了解决这个问题, 文献 [52] 提出了 Strife——一种新的事务处理方案. 它动态地将事务聚集在一起, 并在没有任何并发控制的情况下执行大多数事务. Strife 使用一种快速动态聚类算法, 该算法利用随机抽样和并发联合查找数据结构的组合, 在执行批处理之前对批处理进行在线分区. 实验结果表明, 在高争用工作负载上, Strife 的性能比基于锁的协议和乐观协议高出 2 倍.

2.2.3 负载均衡

随着计算技术的发展, 云计算为用户服务增加了一种新的范式, 允许在任何时间、任何地点以按需付费的方式访问信息技术服务. 由于云服务的灵活性, 许多组织正在将其业务转移到云上, 云中的负载均衡是云计算领域的一个挑战. 一个有效的负载均衡技术应该通过有效地利用虚拟机资源来优化和确保高用户满意度, 从而提供系统的高可用性[53].

文献 [54] 指出, 负载均衡提供了将工作负载平均分配到可用资源上的便利. 其目标是在任何服务组件出现故障的情况下, 通过提供和取消提供应用程序实例并合理利用资源来提供持续的服务. 此外, 负载均衡通常以最小化任务响应时间和提高资源利用率为目标, 以较低的成本提升系统性能和可用性. 文中主要讨论了依赖于负载均衡器标准的各种动态负载均衡技术, 并将其分为通用负载均衡 (包括虚拟机迁移和基于负载估计的算法)、基于自然现象的负载均衡 (如遗传算法、蜜蜂启发、ACO 和 PSO 等)、混合负载均衡、基于代理的负载均衡 (代理用于云资源发现、协商、组合和管理. 多个代理可以协同工作, 以满足用户的 QoS 要求和资源使用)、基于任务的负载均衡 (包括一些基于任务的调度算法等)、基于集群的负载均衡 (根据服务器性能、存储容量等参数划分不同集群).

常用的负载均衡算法通常根据底层环境可以分为 3 类: 静态、动态和自然启发算法[55]. 静态负载均衡算法的执行过程依赖于系统状态的先验知识及其属性和能力. 先验信息的例子可以包括记忆力、存储容量和处理能力. 这些信息代表了系统的负载情况. 动态负载平衡算法在动态环境下负载平衡算法考虑了系统前面的状态. 自然启发的负载平衡算法表示基于人类本性的生物过程或活动进行数学建模, 以采用云计算中的自然过程进行负载均衡. 这些智能算法的发展对于复杂、动态的系统具有更好的表现.

基于 Throttled 算法 (throttled algorithm, TA) 的负载平衡是一种动态算法, 负载均衡器在收到客户端的请求时,

寻找一个合适的虚拟机 (virtual machine,VM) 来执行任务. TA 将保存所有 VM 的列表以及它们的索引值, 这被称为索引表或分配表, 其中 VM 的各自状态 (如可用/忙/闲) 也被存储. 如果一个 VM 是可用的并且有足够的空间, 则任务被接受并分配给该 VM; 如果没有找到可用的 VM, 则 TA 返回 1, 否则请求排队进行快速处理.

基于均衡扩展电流执行 (equally spread current execution, ESCE) 的负载均衡是一种动态算法. 它将作业的大小作为优先级, 然后将工作负载随机地分配给一个轻负载的 VM. ESCE 依赖于使用队列来存储请求, 并在存在过载的 VM 时将负载分配给 VM. Aliyu 等人[56]提出了一种混合方法, 为每个 VM 保持一个阈值作为优先级, 以实现相同的负载分配. 减少响应时间, 降低成本, 提高系统可用性.

基于加权轮询 (weighted round robin, WRR) 的负载均衡与传统的轮询类似, 但该算法考虑了每个节点的权重. 权重由开发者分配, 基于容量最大的虚拟机. 在此基础上, Chen[57]提出了一种基于聚类的 WRR 方法. 该云负载均衡 (cloud load balancing, CLB) 算法可以基于 CPU、内存对虚拟机进行聚类, 并为每个虚拟机分配不同的权重值. 结果表明, 在虚拟 Web 服务器数量相同的情况下, 减少了响应时间, 提高了系统的可用性.

2.3 应用与服务

本节通过分析查询处理、中间件和虚拟化这 3 个层面的技术, 展示高可用数据库在不同的层面上的应用与服务. 本文针对每个层面介绍相关的技术和方法. 这些技术既可以独立应用也可以结合使用, 形成一个更加完善的高可用性体系, 从而在云计算和大规模分布式系统中, 保障服务的连续性和用户体验的优化.

2.3.1 查 询

随着系统规模的增长, 节点故障变得很常见, 文献 [58] 介绍了一种新的框架, 允许主机数据库有效地管理大规模辅助分布式系统的可用性, 并描述了一种通过在辅助分布式系统上透明地重新执行查询 (子) 计划来实现主数据库查询容错的机制. 该框架减少由于故障引起的系统停机时间, 并在查询执行过程中实现对用户的故障透明性, 使用户无感知地获得完整的查询结果, 达到改善 分布式系统的可用性的目标. 该框架的提出基于一种为 RDBMS 加速器系统开发和部署的新可用性框架, 称为 RAPID[59]. RAPID 是一个可插拔的横向扩展关系查询处理引擎, 可以连接到关系数据库系统以卸载分析查询. 该框架指出, 影响系统可用性和用户体验的重要参数是错误率、平均恢复时间和透明故障率. 错误率因素取决于系统软件和硬件组件的弹性, 透明性意味着用户在 RAPID 中执行查询期间不会收到失败通知, 并且无缝地接收查询结果. 其对于任何给定的错误率, 通过低延迟故障转移实现高可用性 (改善“平均恢复时间”), 在查询执行期间将故障与最终用户隔离 (改善“透明故障率”). 当 RAPID 节点发生故障时, 由备用节点替换, 并从主机数据库中重新加载其数据分区. 查询计划中存在一个 RAPID 运算符, 该运算符可以访问为 RAPID 编译的可分配负载子计划和另一个相同的为主机数据库编译的子计划. 为了确保用户查询的透明执行, 系统引入了一种自动重新执行查询的机制: 当 RAPID 节点发生故障时, 系统会自动检测并重新执行受影响的查询, 确保查询结果的完整性和一致性. 此外, 系统还实现了一个托管机制, 在主机数据库中的 RAPID 操作符处保存从每个节点到达的结果行, 直到该节点完成其所有结果的发送. 主机数据库中的 RAPID 操作符与 RAPID 节点之间的通信基于使用零拷贝机制的网络协议.

2.3.2 中间件

文献 [60] 将高可用性解决方案分为中间件方法和基于虚拟化的方法两类, 针对不同层的故障提供不同的保护机制, 提出了一个评估可用性解决方案的框架, 针对不同的故障类型 (应用程序故障、VM 故障、主机故障) 设置评审标准. 在文献 [61] 的研究中, 又进一步将解决方案组织为 3 层 (底层技术、服务和中间件), 并且上层可以由底层的一个或多个解决方案组成. 在云环境中, 数据库系统与中间件的高可用性机制密切相关. 虽然中间件通过冗余、负载均衡和故障转移等机制保证应用层的高可用性, 但若数据库系统出现故障或不可用, 则会直接影响整个系统的稳定性和数据一致性. 数据库的容错性需要与中间件层的容错机制协同工作, 保证在数据库节点发生故障时, 中间件能够自动进行备份和故障转移, 确保数据的高可用性. 中间件架构也会通过集成数据库的分布式副本、数据同步和恢复机制来实现高可用. 数据库系统和中间件的高可用性之间的协作和融合在提供云服务时显得尤为重要. 高效的高可用性设计需要同时考虑数据库的冗余复制、故障转移机制与中间件的容错处理策略, 确保云平

台中各层次的高可用性得以保障.

中间件方法是独立于平台和应用程序的可用性解决方案. 在云环境中, 这些中间件可以作为平台即服务 (platform as a service, PaaS) 提供. 文献 [62] 中提出的 Cloud-Niagara 中间件是一种确保基于云的应用程序的容错性的高可用低延迟中间件. 其架构分为应用层, 中间件层和基础设施层. 应用层运行用户的应用程序并提供对底层容错机制的透明支持; 中间件层实现容错和高可用机制; 基础设施层负责底层的云计算资源和服务. Cloud-Niagara 使用有效的描述性集理论来建模云环境上运行的实际应用的故障检测模型. 在检测到系统故障后, Cloud-Niagara 中间件通过确定性算法自动分配备份节点到系统中, 确保云应用能够在发生故障时平滑过渡, 持续运行. 文中指出, 实现中间件目标的关键在于无延迟通知和资源监控, 这是通过识别故障性质的分析模型来实现的. 针对故障的检测, 该文献提出了一种基于 watchdog、检查点和日志的确定性算法. 文献 [63] 的 OpenStack 是一个用于公共和私有云的开源平台, 用于控制大量计算、存储和网络资源. OpenStack 有多个组件, 每个组件负责云环境的特定方面, 在 HA 方面, 通过 Heat 组件可以从 3 个级别监控资源和应用程序: 应用程序级别、实例级别和堆栈级别 (VM 组). 如果出现失败, Heat 会尝试在当前水平上解决问题. 如果问题仍然存在, 它将尝试在更高的层面上去解决. 但是, 重新启动资源最多可能需要 1 min 的时间. 文献 [64] 在应用程序中实现 HA 的解决方案是在云中部署 SAForum 中间件. HA 的专用中间件提供了一种替代解决方案, 该解决方案基于监视应用程序, 并通过隔离其故障组件来对其故障做出反应, 并将服务故障转移到可以恢复服务供应的冗余副本.

云计算框架提供了成本效率, 更好的资源利用率和可扩展性. 可用性仍然是云面临的主要挑战之一. Heat 等解决方案已被提出, 并与 OpenStack 等云控制器集成. 这些解决方案可以保护服务免受应用程序和基础设施故障的影响. 然而, 应用程序的恢复与底层基础设施的恢复联系在一起, 因此, 服务恢复和中断时间可能是相当大的. 文献 [65] 提出了一种架构, 它集成了现有的高可用性中间件解决方案 OpenSAF 和 OpenStack, 用于管理应用程序和基础设施的可用性. 目标是在发生故障的情况下, 减少基础设施即服务 (infrastructure as a service, IaaS) 云服务模型中 VM 服务的中断时间以及 VM 中运行的应用程序所提供服务的中断时间, 以提高部署在云中的服务的可用性. 该方案取得了比 Heat 更好的效果 (主要是因为更快的故障检测和故障转移恢复选项, 允许应用程序服务的恢复独立于故障组件和底层的修复), 并在冗余模型和恢复操作方面提供了灵活性.

尽管迄今为止大多数基于云的应用程序都是企业级的, 但在云中托管实时流应用程序的需求正在增加, 这需要同时满足高可用性和低延迟的要求. 当前的云计算研究很少集中在以优化数据中心资源消耗的方式为这些应用提供高可用性和实时保证的解决方案上, 文献 [66] 应对这一挑战提出了一个容错云计算基础设施的中间件框架的体系结构细节, 该框架可以根据用户定义的灵活算法自动部署虚拟机的副本. 该中间件扩展了 Remus VM 故障转移解决方案, 并与 OpenNebula 云基础架构软件和 Xen 管理程序集成, 提出了一个可插拔的框架的设计, 使应用程序开发人员能够提供他们的策略来选择物理主机进行副本 VM 放置.

2.3.3 虚拟化

在云计算中采用服务器虚拟化的主要因素是在虚拟化提供的物理资源上重新分配工作负载的灵活性[67]. 允许云提供商在不停止开发人员应用程序 (在虚拟机上运行) 的情况下执行维护, 并通过虚拟机迁移实施更好的资源使用策略. 虚拟化还可以用于在 VM 级别实现 HA 机制, 例如故障和攻击隔离、检查点和回滚作为恢复机制. 除此之外, 虚拟化还可以通过虚拟化网络功能在网络级别上使用, 以实现相同的目标. 为了给非 HA 应用迁移到云端提供通用的可用性解决方案, 人们提出了各种基于虚拟化技术的可用性解决方案. 在这些解决方案中, 故障检测和故障服务的恢复是在虚拟机 (VM) 级别执行的. 在云环境中, 这些基于虚拟化的解决方案可以在 IaaS 提供. 例如, VMware 提出了两种可用性解决方案: VMware HA 和 VMware Fault Tolerance (FT). 这两种解决方案都可以保护应用程序免受虚拟机故障的影响, 并通过重新启动发生故障的虚拟机或将其故障转移到另一台主机来恢复服务. Amazon EC2 等 IaaS 提供商也提供可用性机制来保护虚拟机的可用性. 但是, 用户有责任正确使用这些机制.

文献 [68] 中的 Remus 是一个为运行在 Xenon 上的普通虚拟机提供透明高可用性的软件系统. 它通过将正在运行的虚拟机的副本持续实时迁移到备份服务器来实现这一点, 如果主服务器出现故障, 备份服务器会自动激活. 其主要功能包括: 备份虚拟机是主虚拟机 (磁盘/内存/网络) 的精确副本; 备份完全是最新的, 即使活动的 TCP 会话

也能不间断地维持; 无须以任何方式修改即可保护现有访客. Remus 以主动-被动配置运行配对服务器. 采用了两种主要技术: 使用虚拟化基础架构来简化整个系统的复制、异步复制允许主服务器继续运行的同时异步执行与复制服务器的同步[69].

但是, 在文献 [68] 中提到 Remus 和类似 VM 检查点系统下的数据库系统所经历的性能开销的两个原因. 首先, 数据库系统密集使用内存, 因此在检查点期间需要从主 VM 传输到备份 VM 的状态量很大. 其次, 数据库工作负载可能对网络延迟很敏感, 而用于确保客户端与服务器通信能够在故障中幸存的机制会增加通信路径的延迟. RemusDB[70]采用 Remus 来解决这两个问题. 在这两种情况下, 都会产生开销, 因为 Remus 提供对 DBMS 完全透明的高可用性. 这种完全透明的限制可以放宽, 从而大大减少开销. 文中提出了一种主动-备用 HA 解决方案, 该解决方案基于在虚拟机中运行 DBMS, 并将与 HA 相关的大部分复杂性从 DBMS 中排除, 而是依赖于虚拟化层的功能. 虚拟化层捕获活动主机 (包括 DBMS) 上整个 VM 状态的变化, 并将其传播到备用主机, 并在备用主机上应用到备份 VM. 虚拟化层还检测故障并管理从活动主机到备用主机的故障转移, 这对 DBMS 是透明的. 在故障转移期间, 将维护所有事务 ACID 属性并保留客户端连接, 从而使故障对 DBMS 客户端透明.

在云计算和虚拟化环境中, 传统的容错技术常常面临性能开销大、延迟高的问题. 文献 [71] 中提出的 Phantasy 是旨在通过异步预取技术, 减少容错机制带来的延迟, 实现低开销的高可用性的虚拟化软件系统. 首先确定了现有方法中的两个瓶颈, 即软件中跟踪脏页的开销和检查点系统状态中的长顺序依赖. 为了解决这些瓶颈, 设计了一种新的机制, 在基于虚拟化的容错系统中使用 PML (page-modification logging) 和网络 (RDMA) 的特性实现高效低延迟容错. 首先使用 PML 来降低脏页跟踪开销, 然后在 RDMA 的帮助下, 使用拉取模型异步地预取脏页而不破坏主要的 VM 执行, 以缩短检查点执行中的顺序依赖性. 即通过主动地将 PML 记录的脏页拉取到次级 VM, 从而推测性地预取 PML 记录的脏页, 而不中断主 VM 的执行.

云资源使用量的大幅增加导致服务可用性下降, 从而导致中断、资源争用和过度功耗. 现有的方法主要通过提供多云、虚拟机迁移以及运行每个虚拟机的多个副本来解决这一问题, 这会导致云数据中心的高昂费用, 在此背景下, 文献 [72] 提出了基于 VM 重要性排名和资源估计的高可用性管理模型, 通过优化云数据中心的成本来增强用户的服务可用性, 其估计基于服务器故障的资源争用, 并预先组织所需的资源以维持所需的服务可用性水平, 为每个虚拟机引入并计算重要性排名参数, 执行关键或非关键任务, 然后根据其重要性和用户指定的约束选择可接受的高可用性策略.

数据库系统中使用多种类型的 HA 技术, 有时会组合使用. 在共享访问方法中, 两个或多个数据库服务器实例共享一个保存数据库的公共存储基础架构. 存储基础设施冗余地存储数据, 例如通过将数据镜像到多个设备上, 以确保数据的可靠性. 此外, 服务器访问存储数据的存储互连 (例如 SAN) 必须通过使用冗余访问路径来提高可靠性. 如果数据库服务器发生故障, 可以访问同一数据库的其他服务器可以接管故障服务器的工作负载. 如 Oracle RAC 实现了跨服务器实例的虚拟共享缓冲池、Microsoft SQL Server 中的故障转移集群以及通过 MySQL Cluster 中的 NDB 后端 API 访问的同步数据节点.

云服务正在成为向需要更高可用性服务的用户提供计算服务的最流行的手段之一. 虚拟化是云基础设施的关键使能因素之一. 虚拟机的可用性和托管软件组件的可用性是云中实现高可用服务的基本要素. 目前已有一些虚拟化厂商推出的可用性解决方案, 如 VMware HA 和 VMware FT. 同时, SAForum 规范和 OpenSAF 作为一个兼容的实现, 为服务高可用性提供了一个基于标准的开放解决方案. 为解决虚拟化环境中的高可用性, 文献 [73] 中选择了开源的高可用性解决方案 OpenSAF 和 VMware 作为虚拟化解决方案, 最初的实验表明, VMware HA 对故障的响应能力不如 OpenSAF. 为了结合它们的特点并利用它们的优势, 设计了两种新的架构, 使用非裸机和裸机管理程序来解决所发现的缺点. 主要的目标是通过 OpenSAF 来管理 VM 的生命周期, 以减少 VM 在发生故障时的停机时间和修复时间, 同时运行在 VM 中的服务继续提供服务. 目前为止, 已经实现并实验了使用非裸机管理程序的架构.

3 数据库高可用研究的挑战和趋势

数据库高可用性的研究是一个动态发展的领域, 随着技术进步和业务需求的演变, 新的挑战和趋势不断出现. 随着云计算和分布式系统的广泛应用, 数据库高可用性研究面临的挑战和趋势也日益复杂和多样化.

随着服务器和存储设备的复杂性增加, 硬件故障的诊断和恢复变得更加困难, 研究需要关注如何利用预测性维护和自动化工具来提前识别和修复硬件问题. 同时, 数据库系统对网络的依赖性越来越高, 网络故障可能导致数据同步延迟和访问中断, 研究应关注如何设计更加健壮的网络架构和协议, 以减少单点故障和提高网络恢复速度. 云环境的动态性以及多租户特性也给数据库的高可用性带来了新的挑战, 如资源争用和隔离性问题, 云原生数据库通过微服务架构、容器化技术等实现了高度的灵活性和可扩展性, 然而, 在支持动态扩展的同时保证系统的高可用性仍然存在困难, 尤其是在多租户环境和多云部署中, 跨区域的数据复制、同步、故障恢复等问题变得更加复杂.

通过智能化的故障检测与恢复, 利用机器学习算法分析系统日志和性能指标, 实现故障的预测和自动恢复. 人工智能正在逐渐渗透到数据库管理系统中, AI 技术可以用于预测系统故障、优化负载平衡、自动化故障恢复等. 未来, 基于 AI 的数据库管理系统有望实现更高水平的自治, 减少人工干预, 提高系统的鲁棒性和可用性, 研究如何使数据库系统能够根据实时数据和负载情况自适应地调整配置和资源分配, 以提高可用性. 随着企业在多个云平台上部署应用的需求增加, 多云数据库的高可用性研究将成为重点, 如何在不同的云环境中高效管理数据复制、同步以及处理跨云的故障转移, 是未来研究的重要方向. 为了满足特定应用场景的高可用性需求, 设计符合行业标准的高可用性解决方案, 为特定应用场景提供定制化的数据库服务, 如物联网设备的实时数据处理和分析.

4 数据库高可用技术建议

本节提供实现数据库高可用性的一系列技术建议, 涵盖存储、参数配置、查询优化和事务等关键方面. 存储部分介绍现有的一些典型数据库使用的存储方法以及可靠的存储设备和技术, 以确保数据的持久性和快速访问; 参数配置部分列举一些数据库系统中配置参数的调整方法, 以优化系统性能和稳定性; 查询优化部分分析现有针对查询优化领域研究的不足, 并对此提出合理的建议; 事务部分主要说明在分布式数据库架构中各种一致性协议在事务处理中的应用及其面临的挑战. 其中, 存储和参数配置是数据库系统性能的基础, 事务和查询优化提升了系统的响应速度和数据一致性.

4.1 存　储

良好的数据分区技术能够帮助故障快速恢复, 从而构建高可用、可扩展的存储架构. 现在的商业数据库大多通过基于 Hash 的数据分区方式来实现分布式存储. 例如: TBase 通过复制表和 Hash 分布技术进行数据冗余存储; 达梦数据库采用非对称数据分片技术将数据进行分布式存储. 虽然避免了单节点存储过多数据的问题, 但数据分区仅由数据驱动, 当机器发生故障时, 如果工作负载涉及的数据分布较为分散, 则会导致 RTO 过大. 为此, 可以提出数据自适应分区技术, 采用数据驱动与查询驱动相结合的方式来减小 RTO.

通过数据智能分区选取最优分区可以降低单点故障, 重分区涉及节点间数据迁移, 好的重分区策略可以提高备份效率, 加快数据恢复和容错查询处理, 降低 RTO; 自适应压缩提高存储空间利用率和数据传输效率, 更多的空间可以用于备份复制、故障恢复, 更快的传输可以在数据备份与主备切换时提高响应速度, 降低 RTO.

文献 [74] 介绍了 Megastore. Megastore 是为了满足当今交互式在线服务的要求而开发的存储系统. 其以一种新颖的方式融合了 NoSQL 数据存储的可扩展性和传统 RDBMS 的便捷性, 同时提供了强一致性保证和高可用性. 在物理上使用 BigTable 作为一个数据中心的拓展容错存储, 通过将操作分散至多行来支持随机读写吞吐. 让应用自身控制数据安放位置来减小延迟提高吞吐量. 实现了 Paxos 复制和共识算法, 优化了跨地理分布数据中心的低延迟操作, 为系统提供了高可用性.

文献 [75] 提出的 Dynamo 是一个高可用的键值存储系统, 不保证一致性, 不支持事务, 去中心化, 有更好的高

可用性. 它是一个 NoSQL 的非关系型数据库. Dynamo 使用一致性 Hash 对数据进行分区和复制, 通过对象版本控制保证一致性, 通过仲裁技术和分散的副本同步协议维护一致性. 遵循基于 gossip 的分布式故障检测和成员协议. 当遇到正常节点故障时, 直接切换即可. 某些复杂的情况, Dynamo 实现了一个反熵协议保证副本同步.

文献 [76] 提出的 CANDStore 是一个强一致的、分布式的、可复制的键值存储, 使用了一种新的快速崩溃恢复协议. 该协议利用工作负载特性和现代 NVMe (non-volatile memory express) SSD 技术实现在线故障恢复. CANDStore 使用非易失性主存 (non-volatile main memory, NVMM) 和非易失性存储 (NVMe) 固态硬盘技术来提供低延迟的分布式存储, 这比现有的基于主存的分布式存储更便宜, 有更高的可用性.

文献 [18] 提出了一个面向现代数据中心的分布式主存计算平台 FaRM, 其具有高吞吐量, 低延迟和高可用性的严格可串行化事务. 根据第一性原理, 以利用数据中心中出现的两种硬件趋势: 具有 RDMA 的快速商品网络和提供非易失性 DRAM 的廉价方法. FaRM 的协议遵循 3 个原则来解决 CPU 瓶颈: 减少消息数量、使用单边 RDMA 读写代替消息、有效利用并行性. FaRM 使用 ZooKeeper 协调服务, 使用乐观并发控制, 采用两阶段提交协议. 通过软硬件层面改进消除存储和网络瓶颈.

在文件存储的高可用性上, 文献 [77] 引入了 HAIL, 是一个高可用性和完整性层, 它将 RAID 的基本原理扩展到云的对抗设置中. HAIL 统一了可证明可恢复性 (proof of retrievability, POR) 和可证明数据持久性 (provable data possession, PDP) 的两种方法, 通过一组服务器或独立的存储服务来管理文件的完整性和可用性. 它利用 POR 作为构建模块, 在检测到故障时可以测试和重新分配存储资源. 它为 POR 协议的直接多服务器应用和以前提出的分布式文件可用性建议提供了效率、安全性和建模方面的改进. 通过仔细交错不同类型的纠错层, 并受主动密码模型的启发, HAIL 确保了文件的可用性, 以抵御强大的移动敌手. 上述几种典型存储系统的特性总结如表 2 所示.

表 2　不同存储系统的特点

存储系统	特点	优势	不足
Megastore	跨数据中心复制; 通过Paxos 协议实现全局一致性	易于扩展; 便捷	延迟可能较大
Dynamo	使用一致性Hash对数据进行分区和复制; 遵循基于gossip的分布式故障检测和成员协议	去中心化, 具有更好的高可用性	一致性较弱
CANDStore	支持多种存储模型; 使用快速崩溃恢复协议利用工作负载特性和现代NVMe SSD技术实现在线故障恢复	提供强一致性; 灵活性高, 易于扩展; 成本低	系统架构相对复杂
FaRM	基于内存的分布式计算系统, 支持事务处理; 利用RDMA技术实现较低的网络延迟	高吞吐量、高性能、低延迟	可处理数据规模受内存容量限制; 对底层网络硬件要求较高
HAIL	将RAID的基本原理扩展到云的对抗设置中; 结合了可证明可恢复性(POR)和可证明数据持久性(PDP)的方法, 确保文件的完整性和可用性; 采用了多种类型的纠错层交错排列, 增强了系统对抗敌手的能力	提高了安全性; 效率高	性能开销大

4.2　参数配置

现有的 OtterTune、腾讯云、阿里云等在自动化参数配置方面仅考虑设计算法找到最优的参数配置, 如使用强化学习、贝叶斯优化等黑盒模型来进行云数据库参数调优, 并未考虑高可用指标, 仅仅考虑负载效率的参数调优很可能带来数据库宕机、负载实例启动失败、性能下降等风险. 因此提出面向可用性的参数适配方法进行优化.

数据库内参数不仅对用户负载效率有至关重要的影响, 而且还关系到数据库的可用性. 通过设计面向提升可用性的参数适配方法, 让用户输入负载特性和高可用指标 (如误差容忍范围), 通过基于机器学习的安全调整机制来推荐数据库参数配置, 能够在提高用户负载效率的同时, 大大提高用户实例的可用性, 减小数据库宕机和实例失败.

在云系统中, 高可用性 (HA) 是一个至关重要的问题. 然而, 由于复杂的运行时云环境、大量可用的 HA 机制、易出错的 HA 配置以及不断需要的动态调整, 保证 HA 是具有挑战性的. 文献 [65] 利用运行时系统架构 (runtime system architecture, RSA) 来自动配置云中的 HA. 基于 RSA 的 HA 配置框架由 3 个组件组成, 分别是运行时系统

监控器、HA 机制选择器和 HA 配置器. 运行时信息由运行时系统监控器及时反映在 RSA 上. 为了从多个 HA 机制中选择合适的机制, 提出了一个分类并实现了一个选择器 (但并不完善). HA 配置器是通过合并 RSA 和 HA 机制选择两种模型实现的. 与现有工作相比, 该方法允许根据系统的运行时状态不断调整 HA 机制同时系统运行时的变化被不断地抽象、收集并反映在 RSA 上, 简化了 HA 信息的收集.

4.3 查询优化

商用大规模数据库系统具有网络延时高、传输代价高、计算开销高特点. 然而, 在查询优化领域, 目前数据库的前沿研究多数只针对单个查询性能做优化, 而极少考虑整体负载平衡. 仅考虑单查询性能可能会带来数据库部分节点计算、I/O 负载过高, 进而影响数据库可用性. 针对该问题, 可以提出兼顾系统负载平衡的智能查询优化器, 并以高可用性为约束进行多目标查询优化来解决上述问题.

现有智能查询优化器大多仅能优化单一查询, 当大规模查询发起时, 其无法兼顾系统负载的平衡, 间接增加了数据库节点因负载过大而造成查询阻塞甚至宕机的可能; 针对这一痛点, 通过研究智能查询优化算法, 以低时延、低传输代价、高可用性为约束设计基于强化学习的多目标查询优化, 在兼顾系统整体查询代价的同时, 通过平衡系统负载减少单节点因过载而出现故障的频率, 进而减少 RTO.

4.4 事　务

现有数据库主要是基于一致性协议构建分布式架构, 例如 OceanBase 提出的基于 Paxos 的 2PC, MongoDB 基于日志复制、集群协议选主的分布式架构, 但是上述方案在面对故障时选主或者节点切换时, 不可避免地存在故障延迟、日志延迟、主从同步延迟导致事务中断, 恢复时间增加. 为了保证数据一致性, 部分数据库提供最大可用模式, 由主机将日志同步到备机成功后再应答客户, 例如 MySQL、Oracle 等, 但是这样引入 HA 仲裁组件会由于备机故障或网络抖动导致失败, 影响可用性. 采用强同步复制方案的数据库是当备机数据完全同步后, 才由主机给予应用事务应答保障一致性, 例如 TDSQL 的异步多线程强同步复制方案, 引入了线程池模型进行异步化改造可以提高可用性. 但是, 等待备机应答的机制势必会导致性能损耗.

5 总　结

本文针对当下数据库高可用面临的挑战深入探讨了数据库高可用性的问题, 概述了现有的技术和解决方案, 包括分布式计算、内存计算及云数据库的应用. 通过对系统与网络、数据与计算以及应用服务这 3 大层面的研究现状的分类总结分析, 我们揭示了当前技术在确保高可用性方面的贡献与不足. 此外, 本文还从存储、参数配置、查询优化和事务的方面给出了技术建议. 本综述为未来的研究和实践提供了有价值的见解和建议, 期望能推动数据库高可用技术的进一步发展.

References

[1] Shuquan. Top 20 software & cloud quality accidents and their lessons in 2022. 2023 (in Chinese). https://www.sohu.com/a/635149128_453160

[2] Thomas C. Typo blamed for Microsoft Azure DevOps outage in Brazil. 2023. https://https://www.theregister.com/2023/06/03/microsoft_azure_outage_brazil/

[3] Cai FF. Alibaba cloud suffers a sudden severe global outage, restored after 2.5 hours. 2023 (in Chinese). https://www.infoq.cn/article/K5JfjZy6Qmc70YFPluVi

[4] Marcus E, Stern H. Blueprints for High Availability. 2nd ed., Indianapolis: John Wiley & Sons, 2003.

[5] Bailis P, Davidson A, Fekete A, Ghodsi A, Hellerstein JM, Stoica I. Highly available transactions: Virtues and limitations. Proc. of the VLDB Endowment, 2013, 7(3): 181–192. [doi: 10.14778/2732232.2732237]

[6] Zhu T, Guo JW, Zhou H, Zhou X, Zhou AY. Consistency and availability in distributed database systems. Ruan Jian Xue Bao/Journal of Software, 2018, 29(1): 131–149 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5433.htm [doi: 10.13328/j.cnki.jos.005433]

[7] Mahajan P, Alvisi L, Dahlin M. Consistency, availability, and convergence. Technical Report, UTCS TR-11-22, Austin: University of

Texas at Austin, 2011.

[8] Gupta A, Shute J. High-availability at massive scale: Building Google's data infrastructure for Ads. In: Proc. of the 2019 Int'l Workshops on Real-time Business Intelligence and Analytics. Springer, 2019. 63–81. [doi: 10.1007/978-3-030-24124-7_5]

[9] Gray J, Reuter A. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers Inc., 1992.

[10] Shute J, Vingralek R, Samwel B, Handy B, Whipkey C, Rollins E, Oancea M, Littlefield K, Menestrina D, Ellner S, Cieslewicz J, Rae I, Stancescu T, Apte H. F1: A distributed SQL database that scales. Proc. of the VLDB Endowment, 2013, 6(11): 1068–1079. [doi: 10.14778/2536222.2536232]

[11] Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li HY, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems, 2013, 31(3): 8. [doi: 10.1145/2491245]

[12] Lamport L. Paxos made simple. ACM SIGACT News, 2001, 32(4): 18–25.

[13] Ananthanarayanan R, Basker V, Das S, Gupta A, Jiang HF, Qiu TH, Reznichenko A, Ryabkov D, Singh M, Venkataraman S. Photon: Fault-tolerant and scalable joining of continuous data streams. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 577–588.

[14] Gupta A, Yang F, Govig J, Kirsch A, Chan K, Lai K, Wu S, Dhoot SG, Kumar AR, Agiwal A, Bhansali S, Hong MS, Cameron J, Siddiqi M, Jones D, Shute J, Gubarev A, Venkataraman S, Agrawal D. Mesa: Geo-replicated, near real-time, scalable data warehousing. Proc. of the VLDB Endowment, 2014, 7(12): 1259–1270. [doi: 10.14778/2732977.2732999]

[15] Shrestha R. High availability and performance of database in the cloud. In: Proc. of the 7th Int'l Conf. on Cloud Computing and Services Science. Porto: SciTePress—Science and Technology Publications, 2017. 413–420. [doi: 10.5220/0006294604130420]

[16] Depoutovitch A, Chen C, Larson PA, Ng J, Lin S, Xiong GZ, Lee P, Boctor E, Ren SM, Wu LD, Zhang YC, Sun C. Taurus MM: Bringing multi-master to the cloud. Proc. of the VLDB Endowment, 2023, 16(12): 3488–3500. [doi: 10.14778/3611540.3611542]

[17] Yang XJ, Zhang YQ, Chen H, Li FF, Wang B, Fang J, Sun C, Wang YH. PolarDB-MP: A multi-primary cloud-native database via disaggregated shared memory. In: Companion of the 2024 Int'l Conf. on Management of Data. Santiago AA Chile: ACM, 2024. 295–308. [doi: 10.1145/3626246.3653377]

[18] Dragojević A, Narayanan D, Nightingale EB, Renzelmann M, Shamis A, Badam A, Castro M. No compromises: Distributed transactions with consistency, availability, and performance. In: Proc. of the 25th Symp. on Operating Systems Principles. Monterey: ACM, 2015. 54–70. [doi: 10.1145/2815400.2815425]

[19] Kim J, Salem K, Daudjee K, Aboulnaga A, Pan X. Database high availability using SHADOW systems. In: Proc. of the 6th ACM Symp. on Cloud Computing. ACM, 2015. 209–221. [doi: 10.1145/2806777.2806841]

[20] Kalia A, Kaminsky M, Andersen DG. FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In: Proc. of the 12th USENIX Conf. on Operating Systems Design and Implementation. Savannah: USENIX Association, 2016. 185–201.

[21] Wang TZ, Johnson R, Pandis I. Query fresh: Log shipping on steroids. Proc. of the VLDB Endowment, 2017, 11(4): 406–419. [doi: 10.1145/3186728.3164137]

[22] Kemme B, Alonso G. Database replication: A tale of research across communities. Proc. of the VLDB Endowment, 2010, 3(1-2): 5–12. [doi: 10.14778/1920841.1920847]

[23] Lahiri T, Neimat MA, Folkman S. Oracle TimesTen: An in-memory database for enterprise applications. IEEE Data Engineering Bulletin, 2013, 36(2): 6–13.

[24] Mistry R, Misner S. Introducing Microsoft SQL Server 2014. Redmond: Microsoft Press, 2014.

[25] Gray J, Helland P, O'Neil P, Shasha D. The dangers of replication and a solution. In: Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data. Montreal: ACM Press, 1996. 173–182. [doi: 10.1145/233269.233330]

[26] Kallman R, Kimura H, Natkins J, Pavlo A, Rasin A, Zdonik S, Jones EPC, Madden S, Stonebraker M, Zhang Y, Hugg J, Abadi DJ. H-Store: A high-performance, distributed main memory transaction processing system. Proc. of the VLDB Endowment, 2008, 1(2): 1496–1499. [doi: 10.14778/1454159.1454211]

[27] Stonebraker M, Weisberg A. The voltDB main memory DBMS. IEEE Data Engineering Bulletin, 2013, 36(2): 21–27.

[28] Thomson A, Diamond T, Weng SC, Ren K, Shao P, Abadi DJ. Calvin: Fast distributed transactions for partitioned database systems. In: Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM, 2012. 1–12. [doi: 10.1145/2213836.2213838]

[29] Wu CG, Faleiro JM, Lin YH, Hellerstein JM. Anna: A KVS for any scale. IEEE Trans. on Knowledge and Data Engineering, 2021, 33(2): 344–358. [doi: 10.1109/TKDE.2019.2898401]

[30] Zhang ZH, Hu HQ, Zhou X, Wang J. Starry: Multi-master transaction processing on semi-leader architecture. Proc. of the VLDB

Endowment, 2022, 16(1): 77–89. [doi: 10.14778/3561261.3561268]

[31] Ren K, Li D, Abadi DJ. SLOG: Serializable, low-latency, geo-replicated transactions. Proc. of the VLDB Endowment, 2019, 12(11): 1747–1761. [doi: 10.14778/3342263.3342647]

[32] Zamanian E, Yu XY, Stonebraker M, Kraska T. Rethinking database high availability with RDMA networks. Proc. of the VLDB Endowment, 2019, 12(11): 1637–1650. [doi: 10.14778/3342263.3342639]

[33] Kumari P, Kaur P. A survey of fault tolerance in cloud computing. Journal of King Saud University—Computer and Information Sciences, 2021, 33(10): 1159–1176. [doi: 10.1016/j.jksuci.2018.09.021]

[34] Ma SN, Ma T, Chen K, Wu YW. A survey of storage systems in the RDMA era. IEEE Trans. on Parallel and Distributed Systems, 2022, 33(12): 4395–4409. [doi: 10.1109/TPDS.2022.3188656]

[35] Ahmed S, Nahiduzzaman M, Islam T, Bappy FH, Zaman TS, Hasan R. FASTEN: Towards a fault-tolerant and storage efficient cloud: Balancing between replication and deduplication. In: Proc. of the 21st IEEE Consumer Communications & Networking Conf. Las Vegas: IEEE, 2024. 44–50. [doi: 10.1109/CCNC51664.2024.10454894]

[36] Zheng JJ, Lin Q, Xu JT, Wei C, Zeng CW, Yang PG, Zhang YF. PaxosStore: High-availability storage made practical in WeChat. Proc. of the VLDB Endowment, 2017, 10(12): 1730–1741. [doi: 10.14778/3137765.3137778]

[37] Cao W, Liu ZJ, Wang P, Chen S, Zhu CF, Zheng S, Wang YH, Ma GQ. PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. Proc. of the VLDB Endowment, 2018, 11(12): 1849–1862. [doi: 10.14778/3229863.3229872]

[38] Wang JY, Li TL, Song HZ, Yang XJ, Zhou WC, Li FF, Yan BY, Wu QQ, Liang YK, Ying CJ, Wang YJ, Chen BK, Cai C, Ruan YB, Weng XY, Chen SB, Yin L, Yang CZ, Cai X, Xing HY, Yu NL, Chen XF, Huang DP, Sun JL. PolarDB-IMCI: A cloud-native HTAP database system at alibaba. Proc. of the ACM on Management of Data, 2023, 1(2): 199. [doi: 10.1145/3589785]

[39] Akkoorath DD, Tomsic AZ, Bravo M, Li ZM, Crain T, Bieniusa A, Preguiça N, Shapiro M. Cure: Strong semantics meets high availability and low latency. In: Proc. of the 36th IEEE Int'l Conf. on Distributed Computing Systems. Nara: IEEE, 2016. 405–414. [doi: 10.1109/ICDCS.2016.98]

[40] Zhou WX, Peng Q, Zhang ZJ, Zhang YF, Ren Y, Li SH, Fu G, Cui YL, Li Q, Wu CY, Han SJ, Wang SY, Li GL, Yu G. GeoGauss: Strongly consistent and light-coordinated OLTP for geo-replicated SQL database. Proc. of the ACM on Management of Data, 2023, 1(1): 62. [doi: 10.1145/3588916]

[41] Tamimi AA, Dawood R, Sadaqa L. Disaster recovery techniques in cloud computing. In: Proc. of the 2019 IEEE Jordan Int'l Joint Conf. on Electrical Engineering and Information Technology. Amman: IEEE, 2019. 845–850. [doi: 10.1109/JEEIT.2019.8717450]

[42] Abualkishik AZ, Alwan AA, Gulzar Y. Disaster recovery in cloud computing systems: An overview. Int'l Journal of Advanced Computer Science and Applications, 2020, 11(9): 702–710. [doi: 10.14569/IJACSA.2020.0110984]

[43] Pokharel M, Lee S, Park JS. Disaster recovery for system architecture using cloud computing. In: Proc. of the 10th IEEE/IPSJ Int'l Symp. on Applications and the Internet. Seoul: IEEE, 2010. 304–307. [doi: 10.1109/SAINT.2010.23]

[44] Sengupta S, Annervaz KM. Planning for optimal multi-site data distribution for disaster recovery. In: Proc. of the 8th Int'l Workshop on Economics of Grids, Clouds, Systems, and Services. Paphos: Springer, 2012. 161–172. [doi: 10.1007/978-3-642-28675-9_12]

[45] Saquib Z, Tyagi V, Bokare S, Dongawe S, Dwivedi M, Dwivedi J. A new approach to disaster recovery as a service over cloud for database system. In: Proc. of the 15th Int'l Conf. on Advanced Computing Technologies. Rajampet: IEEE, 2013. 1–6. [doi: 10.1109/ICACT.2013.6914704]

[46] Lenk A. Cloud standby deployment: A model-driven deployment method for disaster recovery in the cloud. In: Proc. of the 8th IEEE Int'l Conf. on Cloud Computing. New York: IEEE, 2015. 933–940. [doi: 10.1109/CLOUD.2015.127]

[47] Bibartiu O, Dürr F, Rothermel K, Ottenwälder B, Grau A. Availability analysis of redundant and replicated cloud services with bayesian networks. Quality and Reliability Engineering Int'l, 2024, 40(1): 561–584. [doi: 10.1002/qre.3414]

[48] Bailis P, Fekete A, Ghodsi A, Hellerstein JM, Stoica I. HAT, not CAP: Towards highly available transactions. In: Proc. of the 14th USENIX Conf. on Hot Topics in Operating Systems. USENIX Association, 2013. 24.

[49] Cao Y, Fan WF, Ou WJ, Xie R, Zhao WY. Transaction scheduling: From conflicts to runtime conflicts. Proc. of the ACM on Management of Data, 2023, 1(1): 26. [doi: 10.1145/3588706]

[50] An S, Cao Y, Zhao WY. Competitive consistent caching for transactions. In: Proc. of the 38th IEEE Int'l Conf. on Data Engineering. Kuala Lumpur: IEEE, 2022. 2154–2167. [doi: 10.1109/ICDE53745.2022.00207]

[51] Qadah TM, Sadoghi M. Highly available queue-oriented speculative transaction processing. arXiv:2107.11378, 2021.

[52] Prasaad G, Cheung A, Suciu D. Handling highly contended OLTP workloads using fast dynamic partitioning. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 527–542. [doi: 10.1145/3318464.3389764]

[53] Shahid MA, Islam N, Alam MM, Su'ud MM, Musa S. A comprehensive study of load balancing approaches in the cloud computing

environment and a novel fault tolerance approach. IEEE Access, 2020, 8: 130500–130526. [doi: 10.1109/ACCESS.2020.3009184]

[54] Kumar P, Kumar R. Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Computing Surveys, 2019, 51(6): 120. [doi: 10.1145/3281010]

[55] Shafiq DA, Jhanjhi NZ, Abdullah A. Load balancing techniques in cloud computing environment: A review. Journal of King Saud University—Computer and Information Sciences, 2022, 34(7): 3910–3933. [doi: 10.1016/j.jksuci.2021.02.007]

[56] Aliyu AN, Souley PB. Performance analysis of a hybrid approach to enhance load balancing in a heterogeneous cloud environment. Int'l Journal of Advances in Scientific Research and Engineering, 2019, 5(7): 246–257. [doi: 10.31695/IJASRE.2019.33430]

[57] Chen SL, Chen YY, Kuo SH. CLB: A novel load balancing architecture and algorithm for cloud services. Computers & Electrical Engineering, 2017, 58: 154–160. [doi: 10.1016/j.compeleceng.2016.01.029]

[58] Pasupuleti KK, Klots B, Nagarajan V, Kandukuri A, Agarwal N. High availability framework and query fault tolerance for hybrid distributed database systems. In: Proc. of the 31st ACM Int'l Conf. on Information & Knowledge Management. Atlanta: ACM, 2022. 3451–3460. [doi: 10.1145/3511808.3557086]

[59] Balkesen C, Kunal N, Giannikis G, Fender P, Sundara S, Schmidt F, Wen J, Agrawal S, Raghavan A, Varadarajan V, Viswanathan A, Chandrasekaran B, Idicula S, Agarwal N, Sedlar E. RAPID: In-memory analytical query processing engine with extreme performance per Watt. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 1407–1419. [doi: 10.1145/3183713.3190655]

[60] Hormati M, Khendek F, Toeroe M. Towards an evaluation framework for availability solutions in the cloud. In: Proc. of the 2014 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Naples: IEEE, 2014. 43–46. [doi: 10.1109/ISSREW.2014.50]

[61] Endo PT, Rodrigues M, Gonçalves GE, Kelner J, Sadok DH, Curescu C. High availability in clouds: Systematic review and research challenges. Journal of Cloud Computing, 2016, 5(1): 16. [doi: 10.1186/s13677-016-0066-8]

[62] Imran A, Gias AU, Rahman R, Seal A, Rahman T, Ishraque F, Sakib K. Cloud-Niagara: A high availability and low overhead fault tolerance middleware for the cloud. In: Proc. of the 16th Int'l Conf. Computer and Information Technology. Khulna: IEEE, 2014. 271–276. [doi: 10.1109/ICCITechn.2014.6997344]

[63] OpenStack. Open source cloud computing infrastructure—OpenStack. 2024. http://www.openstack.org

[64] Kanso A, Lemieux Y. Achieving high availability at the application level in the cloud. In: Proc. of the IEEE 6th Int'l Conf. on Cloud Computing. Santa Clara: IEEE, 2013. 778–785. [doi: 10.1109/CLOUD.2013.24]

[65] Heidari P, Hormati M, Toeroe M, Al Ahmad Y, Khendek F. Integrating open SAF high availability solution with open stack. In: Proc. of the 2015 IEEE World Congress on Services. New York: IEEE, 2015. 229–236. [doi: 10.1109/SERVICES.2015.41]

[66] An K, Shekhar S, Caglar F, Gokhale A, Sastry S. A cloud middleware for assuring performance and high availability of soft real-time applications. Journal of Systems Architecture, 2014, 60(9): 757–769. [doi: 10.1016/j.sysarc.2014.01.009]

[67] Endo PT, De Almeida Palhares AV, Pereira NN, Goncalves GE, Sadok D, Kelner J, Melander B, Mangs JE. Resource allocation for distributed cloud: Concepts and research challenges. IEEE Network, 2011, 25(4): 42–46. [doi: 10.1109/mnet.2011.5958007]

[68] Cully B, Lefebvre G, Meyer D, Feeley M, Hutchinson N, Warfield A. Remus: High availability via asynchronous virtual machine replication. In: Proc. of the 5th USENIX Symp. on Networked Systems Design and Implementation. San Francisco: USENIX Association, 2008. 161–174.

[69] Sharma YK, Singh AS. High availability of databases for cloud. In: Satapathy SC, Joshi A, Modi N, Pathak N, eds. Proc. of the 2016 Int'l Conf. on ICT for Sustainable Development. Singapore: Springer, 2016. 501–509. [doi: 10.1007/978-981-10-0135-2_49]

[70] Minhas UF, Rajagopalan S, Cully B, Aboulnaga A, Salem K, Warfield A. RemusDB: Transparent high availability for database systems. The VLDB Journal, 2013, 22(1): 29–45. [doi: 10.1007/s00778-012-0294-6]

[71] Ren SR, Zhang YQ, Pan LC, Xiao Z. Phantasy: Low-latency virtualization-based fault tolerance via asynchronous prefetching. IEEE Trans. on Computers, 2019, 68(2): 225–238. [doi: 10.1109/TC.2018.2865943]

[72] Saxena D, Singh AK. A high availability management model based on VM significance ranking and resource estimation for cloud applications. IEEE Trans. on Services Computing, 2023, 16(3): 1604–1615. [doi: 10.1109/TSC.2022.3206417]

[73] Nikzad A, Khendek F, Toeroe M. OpenSAF and VMware from the perspective of high availability. In: Proc. of the 9th Int'l Conf. on Network and Service Management. Zurich: IEEE, 2013. 324–331. [doi: 10.1109/CNSM.2013.6727853]

[74] Baker J, Bond C, Corbett JC, Furman JJ, Khorlin A, Larson J, Leon JM, Li YW, Lloyd A, Yushprakh V. Megastore: Providing scalable, highly available storage for interactive services. In: Proc. of the 5th Biennial Conf. on Innovative Data Systems Research. 2011. 223–234.

[75] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. ACM SIGOPS Operating Systems Review, 2007, 41(6): 205–220. [doi: 10.1145/1323293.1294281]

[76] Kim T, Wong DLK, Ganger GR, Kaminsky M, Andersen DG. High availability in cheap distributed key value storage. In: Proc. of the

11th ACM Symp. on Cloud Computing. ACM, 2020. 165–178. [doi: 10.1145/3419111.3421290]

[77] Bowers KD, Juels A, Oprea A. HAIL: A high-availability and integrity layer for cloud storage. In: Proc. of the 16th ACM Conf. on Computer and Communications Security. Chicago: ACM, 2009. 187–198. [doi: 10.1145/1653662.1653686]

附中文参考文献

[1] 书圈. 2022 年软件&云 Top20 质量事故及其经验教训. 2023. https://www.sohu.com/a/635149128_453160

[3] 蔡芳芳. 阿里云突发全球性严重故障, 历经 2.5 小时恢复. 2023. https://www.infoq.cn/article/K5JfjZy6Qmc70YFPluVi

[6] 朱涛, 郭进伟, 周欢, 周烜, 周傲英. 分布式数据库中一致性与可用性的关系. 软件学报, 2018, 29(1): 131–149. http://www.jos.org.cn/1000-9825/5433.htm [doi: 10.13328/j.cnki.jos.005433]

作者简介

向清平, 硕士生, 主要研究领域为云边端协同数据库, 数据库高可用.

燕钰, 博士, 副研究员, CCF 专业会员, 主要研究领域为 AI4DB.

程思佳, 本科生, CCF 学生会员, 主要研究领域为 AI4DB, 数据库高可用.

王宏志, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库管理系统, 大数据分析与治理.