SteelDB：云端 OLTP 数据库内核空间瓶颈诊断

Mitsumasa Kondo
NTT, Inc. 日本东京
m
itsumasa.kondou@ntt.com

arXiv:2603.29052v1 [cs.DB] 30 Mar 2026

摘要

现代云端 OLTP 数据库主要通过用户空间优化来提升性能——分离存储层与计算层，或利用共识算法在多个节点间分发事务。本文将目光转向一个此前鲜少探索的层面：内核空间的 I/O 行为。从本地部署的视角来看，单台配备本地存储的服务器即可提供出色性能，因此这些复杂的设计似乎令人费解。云数据库为何需要如此复杂的架构？我们通过对依赖操作系统级 I/O 控制、运行于云专用存储环境中的数据库进行病态分析，探究这一问题。我们发现，通常被归因于网络或存储架构的瓶颈，实际上源自内核空间的 I/O 行为。基于这一诊断，我们总结出处理原则，并将其实现为 SteelDB：一种零补丁架构，无需修改内核或数据库补丁，通过有针对性的 I/O 优化，提升数据库在通用云分布式块存储上的性能。TPC-C 评测表明，SteelDB 在不增加成本的情况下，性能最高可提升 9 倍。与 Amazon Aurora 相比，SteelDB 的性能高出 3.1 倍，同时成本降低 58%，成本效益提升 7.3 倍。Aurora 由于需要为新发布的 OSS 数据库应用专有补丁，主要版本升级平均耗时 254 天；而我们的零补丁架构可将此类软件维护成本降至近乎为零。

关键词

云 OLTP、内核刷新线程、合并 I/O、分布式块存储、零补丁架构

1 引言

数据库市场持续稳步增长，到2024年，云数据库占据了总市场份额的 64% [56]。随着云采用加速，对高性能且经济高效的数据库解决方案的需求，成为推动采用的关键因素。

本文聚焦于用于 OLTP 的云数据库。现有云端 OLTP 数据库主要通过两种架构方法应对性能挑战。第一种是存算分离数据库，代表产品包括 Amazon Aurora [60] 和谷歌 AlloyDB [27]。这些系统将计算层与存储层分离，仅将预写日志从计算节点传输到专用存储节点，并在后者重建数据页。第二种方法是分布式事务数据库，例如谷歌 Spanner [22] 和 TiDB [37]，它们使用 Raft 和 Paxos 等共识算法将数据分布到多个节点，以实现水平扩展和容错。

这两种方法都会带来不小的软件复杂性，并增加用户承担的成本。在云环境中，Amazon RDS 等托管数据库服务的成本约为标准虚拟机实例的两倍 [6]，，主要源于专用存储和管理开销。Aurora 等存算分离数据库的成本还会进一步升至标准实例的约三倍，这反映了专有存储系统带来的额外费用。除基础设施成本外，这些系统还需要投入大量工程工作，以保持与上游开源版本的兼容性。分布式数据库则会产生事务协调开销，并且难以支持常见的 SQL 模式，例如分布式连接操作。

一个根本性问题仍未得到解答：为什么云数据库需要如此复杂的架构？在本地部署环境中，单台配备本地存储的服务器使用 PostgreSQL 等标准数据库即可提供出色的性能 [52]。云环境为何如此不同？此前的研究 [50, 60] 将云数据库性能方面的挑战主要归因于网络限制——具体而言，是访问分布式存储时每秒数据包数（PPS）和带宽限制。存算分离数据库旨在解决这些瓶颈。然而，现代云基础设施已发生了显著变化。如今，虚拟机实例可提供 10–100 Gbps 的网络带宽，核心网络设备则可达到 100–400 Gbps。如果网络性能不再是主要限制因素，那么根本瓶颈究竟是什么？

本文有两项贡献。首先，我们对依赖操作系统级 I/O 控制、运行于云专用存储环境中的数据库进行了病理分析，并发现了先前研究中被忽视的内核空间瓶颈。我们证明，广泛被归因于网络或存储架构的性能限制，实际上源于缓冲 I/O 机制与高延迟云分布式块存储之间的次优交互。其次，我们基于这一诊断推导出处理原则，并将其实现为 SteelDB：一种零补丁架构，无需修改内核或数据库补丁，即可通过有针对性的 I/O 优化提升性能。TPC-C 评估结果表明，SteelDB 在不增加成本的情况下，性能最高可提升 9 倍。与 Amazon Aurora 相比，SteelDB 的性能提高了 3.1 倍，同时成本降低了 58%，使成本效益提升了 7.3 倍。

本文的结构如下。第 2 节介绍基础知识，说明数据库 I/O、分布式块存储和内核 I/O 控制机制在正常情况下如何运行。第 3 节展示我们的病态分析，通过在分布式存储上进行受控实验，记录性能下降的症状。第 4 节诊断根本原因。第 5 节推导处理原则，并介绍 SteelDB 架构

及其实现。第6节通过 TPC-C 基准测试以及与商业云数据库的比较，评估处理效果。第7节讨论相关工作，第8节总结全文。

2 技术背景

本节介绍数据库 I/O、分布式存储和内核 I/O 控制机制的常规运作方式，这些内容为后续章节中的异常分析奠定了基础。

2.1 数据库

*2.1.1 数据库 I/O 控制。*数据库 I/O 控制方法主要分为数据库控制方式和操作系统控制方式。在数据库控制方式中，数据库在内部管理 I/O，而不依赖操作系统级缓冲 I/O。这通常涉及通过直接 I/O 访问本地存储，也包括完全绕过本地存储的架构，例如 Aurora 将预写日志传输至专用存储节点。Aurora、AlloyDB 等存储与计算分离的数据库，以及 Oracle Database [47], 都采用这种方法。然而，当许多数据库功能仍有待实现时，单独投入开发资源来控制 I/O 并不现实。因此，许多数据库会将 I/O 控制委托给操作系统 [39, 54]。

PostgreSQL 和 RocksDB 等由操作系统控制的数据库 [24] 默认依赖操作系统的缓冲 I/O，将成本高昂的 I/O 控制实现交由内核负责，并将开发资源集中用于数据库特有的功能。这种设计体现了 UNIX 哲学 [41]：“让每个程序做好一件事”和“小即是美”——这一策略以放弃 I/O 控制为代价，换取跨平台可移植性。直接 I/O 若实现得足够完善，性能可以超过缓冲 I/O，但其采用门槛较高；即使在高性能计算环境 [54]中，缓冲 I/O 仍然十分普遍。遵循这一原则，拥有 20 多年历史的 PostgreSQL 将数据库功能开发置于优先地位，并把 I/O 控制交由操作系统，因此成为全球最符合标准的 SQL 数据库之一 [18]。MySQL [46]采用混合方案，以数据库端的直接 I/O 配合操作系统的缓冲 I/O。

将 I/O 操作交由操作系统处理也可能很高效：缓冲 I/O 避免了用户空间直接 I/O 每次操作都要承担的系统调用开销。本文聚焦于将 I/O 控制委托给操作系统的数据库，并以 PostgreSQL 为代表案例。PostgreSQL 以 8 KB 页面为单位执行 I/O 操作，其中包括预写日志、表和索引等关键数据结构。无论是随机 I/O 还是顺序 I/O，I/O 系统调用都以 8 KB 为单位发出。预写日志执行顺序写入，而表和索引的访问模式则会根据 SQL 操作，在顺序 I/O 和随机 I/O 之间变化。运行 BCC [38]（BPF 编译器集合）工具中的 biopattern 工具，可以在 PostgreSQL 执行操作时观察到这种 I/O 模式；该工具使用 eBPF [25]（扩展伯克利包过滤器）。

*2.1.2 预写日志。*WAL [42] 通过在将更改应用到数据文件之前按顺序记录这些更改，确保数据持久性。在 PostgreSQL 中，WAL 记录存储在 8 KB 页面中，并在每次事务提交时使用缓冲 I/O 执行写入，随后执行 fsync 操作以确保数据持久化。WAL 写入进程每秒可能执行超过 100 次 fsync 操作，使其成为影响写入性能的主要因素。由于 WAL 写入是同步处理的，较低的存储延迟可直接带来更好的写入性能。

*2.1.3 表、索引和表空间。*预写日志中的数据会存储在共享缓冲区中，并在检查点操作期间或需要释放缓冲区空间时，与索引数据一起写入磁盘。这些磁盘写入采用缓冲 I/O：数据首先作为脏页写入内存，然后由操作系统的定期刷新操作或检查点完成时最后执行的 fsync 刷新到磁盘。除这最后一次 fsync 外，其他写入均以异步方式处理。表空间允许用户将表和索引分配到特定存储位置，从而将数据分布到多个磁盘上。表空间曾经是实现 I/O 并行性的常见方式，但随着硬件 RAID 和高性能存储的出现，逐渐不再使用。SteelDB 为了不同的用途，重新启用了这一标准功能。

2.2 分布式块存储

云数据库依赖网络连接的分布式块存储来实现持久化。在云环境中，本地存储通常不具备持久性——虚拟机可能会在重启或重新平衡期间迁移到其他主机服务器——因此，分布式块存储成为数据库的标准做法。某些实例类型提供本地连接的 NVMe 存储，但存储在这些设备上的数据会在实例停止或重启时丢失，因此不适用于需要持久性的数据库场景。分布式存储既包括对象存储（例如 Amazon S3 [61]），也包括块存储（例如 Amazon EBS [7]）；本文重点讨论分布式块存储。由于主要云服务提供商只公开部分存储内部信息（例如 EBS 的 Physalia [15]），而不公开服务质量强制实施等数据路径细节，我们采用广泛使用的开源实现 Ceph [1, 62], 作为代表性示例。这些系统通过 Raft [45] 和 Paxos [40], 等分布式共识算法，在多台服务器和磁盘之间复制数据，从而确保高可用性和数据一致性。在 Ceph 中，CRUSH 算法通过放置组将数据分布到多个物理磁盘上，从而实现类似 RAID 的吞吐量。

虽然分布式块存储对用户而言表现为标准块设备，但其性能特征与本地存储截然不同。分布式存储旨在通过 mClock 等 QoS 机制 [33] 为多个并发用户提供稳定的 I/O [34], ，而不是让单个用户充分利用底层硬件。此外，通过网络对多块磁盘执行同步 I/O 操作会导致较高的 I/O 延迟——这对小型 I/O 操作尤其不利，因为此时每次操作的延迟会主导吞吐量。相比之下，本地存储通过内部总线（例如 SATA、NVMe）直接访问磁盘，即使 I/O 大小较小，也能实现极低的延迟和较高的吞吐量。本地部署还可受益于配备电池备份写缓存的硬件 RAID；而云分布式存储无法提供此功能，只提供软件 RAID。这些差异构成了本地部署的性能优势，也促使我们在第 3 节中进一步探讨。

2.3 内核 I/O 控制

本节介绍内核提供的磁盘 I/O 特性，重点讨论广泛应用于 [58], 的 Linux 内核。

许多数据库系统中。Linux 内核主要提供两种磁盘 I/O 接口：缓冲 I/O 和直接 I/O。缓冲 I/O 是处理文件系统数据的标准方法，PostgreSQL、MySQL 和 RocksDB 等主流数据库都广泛采用它 [24]。由于内核会透明地应用自身的 I/O 优化，缓冲 I/O 的性能往往优于直接 I/O。如第 2.1 节所述，大多数数据库都依赖缓冲 I/O。

2.3.1 脏页与回写。在缓冲 I/O 写入操作中，数据不会立即写入磁盘，而是先作为脏页存储在内核页缓存中。内核刷新线程（KFT）会定期将这些脏页刷新到磁盘，这一过程称为回写。回写会批量处理多次写入，并根据系统负载和内存使用量动态调整其行为。对于同步磁盘写入，fsync 系统调用会刷新脏页并等待操作完成，以确保数据持久性。Linux 内核的回写架构如图 1 所示。

2.3.2 内核刷新线程。内核刷新线程（KFT）是操作系统内核中的后台进程，会定期将脏页从内存刷新到磁盘。写入系统调用会生成脏页，这些脏页会留在内存中，直到 KFT 将其回写到持久化存储。KFT 还负责处理由 fsync 系统调用触发的磁盘写入。通常，每个块设备分配一个 KFT；即使配置了多个物理磁盘组成 RAID 阵列，也只有一个 KFT 负责处理整个阵列。正如后续章节所述，这些特性在云环境中尤为重要。通过进程列表（例如使用 top 或 ps）可以看到 KFT，显示为 askworker/<workqueue>+flush- 259:0，其中 259:0 是目标块设备的标识符。

2.3.3 无 *I/O* 脏页节流。无 I/O 脏页节流 [64]是 Linux 内核版本 3.2 引入的一项操作系统功能。当脏页数量持续超过某个阈值时，它会逐步限制 write 系统调用的速度，以防止内存耗尽。尽管该机制对系统稳定性至关重要，但在高延迟存储上，它可能会严重降低数据库性能。通过限制写入系统调用的频率，该机制可确保操作系统进程稳定运行，而不会停止输入/输出操作。然而，如果脏页刷写到磁盘的速度仍然不足，操作系统可能会完全暂停执行写入系统调用。该机制虽然对系统稳定性至关重要，但在存储延迟较高时，可能会显著影响数据库性能。

2.3.4 I/O 队列。 内核的 I/O 队列对于高效处理磁盘和网络存储的 I/O 请求至关重要。对于 NVMe 这类高速设备，Linux 使用块层多队列（blk-mq）框架，从而能够并行处理多个 I/O 请求，提高吞吐量并降低延迟。在可用的调度算法中，none 调度器按照到达顺序处理 I/O 请求，不进行重新排序；它在支持合并 I/O 的同时，以极低开销实现低延迟磁盘 I/O。在云环境中，推荐使用 none 调度器；本文所有实验均使用该调度器进行。Linux 多队列的总数由 vCPU 核心数和设备队列数量中的较小值决定。关键的是，如果设备公开的队列数少于可用的 vCPU 核心数，用户就无法通过任何配置增加队列数——这是块设备施加的硬件级限制。每个

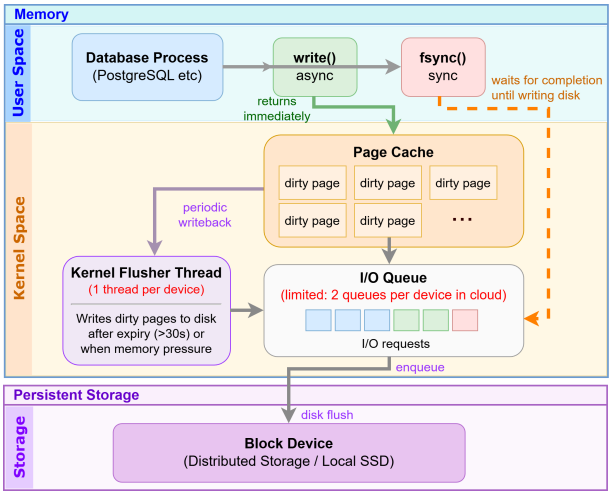


图 1：Linux 内核回写架构

队列分配给特定的 vCPU，在该 vCPU 上执行的 I/O 命令通过其对应的队列进行处理。多块磁盘时也适用这一机制，每块磁盘都有自己的一组队列分配。可通过 /sys/block/<device name>/mq 查看 I/O 队列的数量。

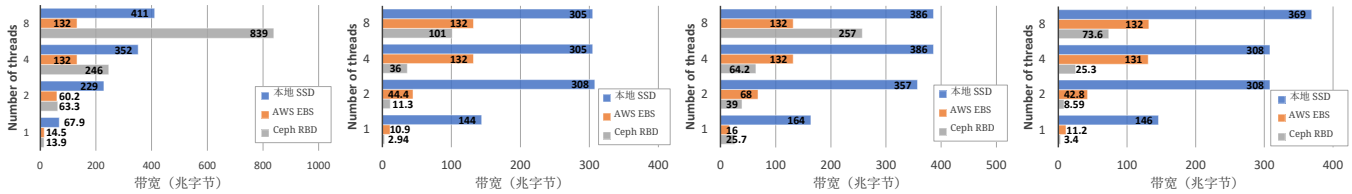
2.3.5 合并 *I/O*。合并 I/O 是 Linux 内核的一项功能，可在缓冲 I/O 处理期间，将多个读取或写入请求合并为一个更大的请求。该功能最早于 Linux 内核 1.0 版（1994 年）实现，旨在优化 HDD 等磁头驱动存储设备的 I/O，在现代内核中仍然有效。聚合多个 I/O 操作会增大每条 I/O 命令的规模，从而提升磁盘 I/O 效率。例如，若向连续块发出 30 个各为 8 KB 的写入请求，合并写入会将其整合为一个 240 KB 的写入操作。这最多可减少 30× 的 IOPS 消耗。该功能对采用分布式块存储的云环境尤其有益，因为更高的 IOPS 配置会带来更高的成本。因此，合并 I/O 在提升吞吐量的同时减少 IOPS 消耗；在按 IOPS 计费的云环境中，这兼具性能与成本优势。此外，只要目标块相邻，合并 I/O 甚至可以应用于来自不同进程的 I/O 命令。要执行合并 I/O，这些 I/O 命令必须位于同一个 I/O 队列中，且目标块必须连续。

主动利用合并 I/O 是我们方法的核心——SteelDB 并非将合并 I/O 视为被动的内核优化，而是主动创造条件，以最大限度地发挥其效用。

2.3.6 内核 *I/O* 控制概要。在正常情况下——本地存储延迟低且 I/O 队列容量充足——这些内核机制能够协同工作。刷新线程能够跟上脏页的生成速度，I/O 队列能够高效地处理请求，而合并 I/O 则会整合顺序写入。然而，正如我们在第 3 节中所展示的，云端分布式块存储打破了这些假设。

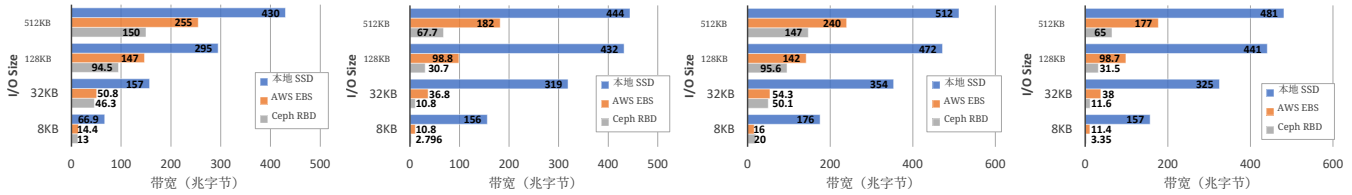
3 病态分析

本节将考察云分布式块存储的输入/输出行为，以识别类数据库负载下的性能异常



(a) 随机读 (b) 随机写 (c) 顺序读 (d) 顺序写

图 2：不同线程数下的直接 I/O 带宽测量



(a) 随机读 (b) 随机写 (c) 顺序读 (d) 顺序写

图 3：不同 I/O 大小下的直接 I/O 带宽测量

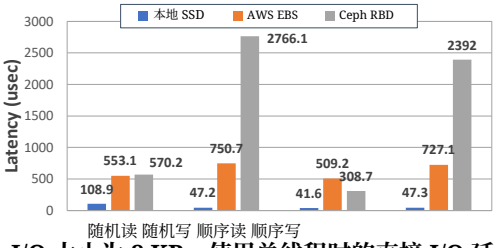


图 4：I/O 大小为 8 KB、使用单线程时的直接 I/O 延迟测量

服务器： ProLiant DL325 Gen10 Plus x6 **磁盘（操作系统）：** SATA 固态硬盘 1TB**CPU：** EPYC 7302P 16C/32T(3GHz) **磁盘（Ceph）：** SATA 固态硬盘 1TB x 7 **内存：** DDR4 192GB **网络：** Mellanox ConnectX-6 DX 100Gbps x2 **操作系统：** Rocky Linux 9.3 (x86_64) **交换机：** Edgecore AS7726-32X (SONiC)

图 5：Ceph 集群的规格

工作负载。我们以本地存储作为健康基线，识别会加剧或缓解这些症状的条件。这里所说的“健康”行为，是指其反映了典型操作系统 I/O 启发式策略所隐含假设的性能特征，而非对存储系统作出规范性判断。

3.1 检查设置

我们将两种分布式存储系统——AWS EBS（gp3）和 Ceph RBD（RADOS 块设备）——与作为健康基准的本地 SATA 固态硬盘进行比较。所有配置均使用 ext4 文件系统。Ceph 集群的规格如图 5 所示。本文所有实验均在 Rocky Linux 9.3 和 Linux 内核 5.14 版上进行。EBS 配置的 IOPS 限制设为 16,000，带宽限制设为 1,000 MB/s。对于 Ceph RBD，Ceph 端的块大小配置为 1 MB，使用的 Ceph 版本为 17.2.4。我们使用 fio [14] 和 io_uring 引擎，在同一环境中同时测量直接 I/O（D-IO）和缓冲 I/O（B-IO）。输入/输出操作在一个预先使用 dd 命令创建的 200 GB 文件上进行，测量时长为 300 秒。

3.2 压力条件

我们识别出会加剧这些症状的条件。作为一项基准观察，分布式存储的延迟比本地固态硬盘高出 10 倍以上（针对 8 KB I/O 操作，见图 4）。这种延迟差距从根本上改变了内核 I/O 行为，以下压力条件便展示了这一点。

压力条件 1：并行度受限。 图 2 展示了实验结果：I/O 大小固定为 8 KB，线程数分别为 1、2、4 和 8。将 I/O 大小设为 8 KB，是为了与 PostgreSQL 的典型 I/O 行为保持一致；PostgreSQL 通常以 8 KB 为块进行操作。

在小型 I/O 大小下进行单线程访问时，分布式存储的吞吐量会降至本地存储性能的一小部分。具体而言，对于单线程随机写，本地 SSD 的速度为 144 MB/s，而 EBS 为 10.9 MB/s，Ceph RBD 则约为 3 MB/s。这一差距证实，并行度受限是主要的加剧因素。

这一现象与数据库工作负载直接相关：内核刷新线程在每个设备上都以单线程方式运行，因此在延迟开销较高的存储上会受延迟限制。

压力条件 2：小型 I/O 大小。 图 3 展示了线程数固定为 1、I/O 大小分别为 8 KB、32 KB、128 KB 和 512 KB 时的结果。我们使用单线程来隔离每操作延迟开销，这对于依赖少量内核回写线程的工作负载尤为重要。

本地固态硬盘即使在 I/O 大小较小时也能保持高带宽；相比之下，EBS 和 Ceph RBD 的吞吐量会大幅下降。即使在 32 KB 时，EBS 的随机写入吞吐量仍低于 50 MB/s，直到 512 KB 才恢复到具有竞争力的水平——这表明分布式存储需要较大的 I/O 大小，才能摊销每次操作的延迟。因此，较小的 I/O 大小是加剧分布式存储问题的第二个因素。

压力条件 3：使用缓冲 I/O 持续写入负载。 在直接 I/O 测试中，fio 的结果反映了存储端和客户端的 I/O 性能。然而，缓冲 I/O 会先将数据写入内存，因此 fio 的结果不足以准确衡量存储端回写性能。因此，我们使用带有扩展统计选项（-x）的 iostat [26] 命令，测量缓冲 I/O 下的存储端 I/O 行为，重点关注有效活动时段并分析合并 I/O 统计。缓冲 I/O 测试中的线程数设为 8，I/O 大小固定为 8 KB。

表 1：使用 fio 和 iostat 对缓冲 I/O 进行性能分析，I/O 大小为 8 KB，线程数为 8（a）本地 SSD 结果（b）AWS EBS 结果（c）CephRBD 结果

	直接 I/O（MB/s）	缓冲 I/O（MB/s）	缓冲 I/O 平均大小（KB）
随机读	411	400	8.005
随机写	305	251	8.819
顺序读	386	539	878.9
顺序写	369	343	998.1

	直接 I/O（MB/s）	缓冲 I/O（MB/s）	缓冲 I/O 平均大小（KB）
随机读	132	128	8.000
随机写	132	144	8.948
顺序读	132	215	170.5
顺序写	132	487	122.0

	直接 I/O（MB/s）	缓冲 I/O（MB/s）	缓冲 I/O 平均大小 (KB)
随机读	839	614	8.000
随机写	101	51.1	8.395
顺序读	257	131	113.9
顺序写	73.6	225	327.0

表 1 展示了测试结果。在使用缓冲 I/O 的持续随机写负载下，Ceph RBD 的吞吐量降至 51.1 MB/s，明显低于直接 I/O 测得的 101 MB/s。进一步检查发现，脏页会随时间不断累积，最终触发无 I/O 脏页节流，导致写入吞吐量下降。内核刷新线程无法跟上高延迟存储上的持续写入。KFT 的周期性激活机制进一步加剧了这一问题：KFT 并非持续运行，而是按照内核的脏页回写参数设定的间隔唤醒，导致脏页在两次运行之间不断累积且得不到处理。在高延迟存储上，由于每次 I/O 都会造成阻塞，每个激活周期内刷新的脏页更少；与此同时，脏页在两次激活之间持续增加，从而加速触发无 I/O 脏页节流。

压力条件 4：单卷 QoS 上限。分布式块存储旨在通过 mClock 等 QoS 机制，为多个并发用户提供稳定的 I/O 性能。与本地存储不同，在本地存储中，单个用户可以充分利用磁盘的性能；分布式存储会强制实施每设备性能保证，防止任何单个卷独占底层物理基础设施。因此，单个虚拟磁盘无法达到其所跨越的物理磁盘池所能提供的聚合吞吐量。数据库在单个卷上运行时，无论该卷标称的 IOPS 或带宽设置如何，都会受到这一 QoS 上限的固有限制。此条件与压力条件 1–3 相互作用：即使内核刷新线程能够更快地处理输入/输出，单卷 QoS 上限仍会限制可实现的吞吐量。

3.3 缓解条件

同样重要的是，找出哪些因素能够缓解这些症状。

缓解条件 1：提高并行度。随着线程数增加，分布式存储的吞吐量会大幅提升。如图 2 所示，EBS 随机写性能从 1 个线程时的 10.9 MB/s 提升至 4 个线程时的 132 MB/s，提升了 12 倍。Ceph RBD 也呈现出显著的扩展性，性能从约 3 MB/s 提升至 101 MB/s，提升了 34 倍。线程数较高时，分布式存储与本地存储之间的性能差距会小得多。并行化通过将每次 I/O 延迟分摊到多个并发操作中，缓解了这一问题。

缓解条件 2：更大的 I/O 大小。将 I/O 大小从 8 KB 增至 512 KB，可显著提升分布式存储的吞吐量。如图 3 所示，EBS 随机写入性能从 8 KB 时的 10.8 MB/s 提升至 512 KB 时的 182 MB/s，提升了 17 倍。Ceph RBD 也有类似提升，从 2.8 MB/s 提升至 67.7 MB/s，提升了 24 倍。I/O 合并通过减少每次操作的延迟开销，缓解了这一性能异常。

缓解条件 3：通过合并 I/O 实现 I/O 合并。合并 I/O 发挥作用时，性能会显著提升。表 1 显示，在分布式存储上，采用缓冲 I/O 的顺序写入性能可以超过直接 I/O。EBS 采用缓冲 I/O 时达到 487 MB/s，而采用直接 I/O 时为 132 MB/s，性能提升了 3.7 倍。平均 I/O 大小为 122 KB，这表明合并 I/O 成功将多个 8 KB 写入合并为更大的操作。值得注意的是，通过这种合并，EBS 带宽超过了其名义上的受 IOPS 限制的吞吐量，这证明合并 I/O 是一种强有力的缓解机制。

缓解条件 4：多卷性能聚合。前面的实验已经展现出单卷上限：如图 2 所示，EBS 随机写入吞吐量在超过 4 个线程后达到 132 MB/s 的饱和上限——这是配置的 16,000 IOPS 所施加的限制；而本地 SSD 的性能仍在持续扩展。这种饱和并非由内核空间瓶颈导致，而是由针对每个设备实施的服务质量约束造成的。由于存储成本主要取决于卷总大小，而非卷的数量，因此分配多个标准 gp3 卷（每个卷为 3,000 IOPS、125 MB/s），即可在成本相同的情况下聚合出相等或更高的吞吐量，从而突破单卷上限。

这些发现表明，病理现象源于内核空间 I/O 机制和每卷服务质量约束。在第 4 节中，我们将诊断具体的根本原因。

4 诊断

根据第 3 节记录的症状，我们诊断出四个共同导致病态级联的根本原因。

4.1 根本原因

根本原因 1：单线程内核刷新器。Linux 内核为每个块设备分配一个内核刷新线程（KFT）。对于延迟为 40–100 微秒的本地存储，这种设计运行良好。然而，对于延迟为 500–2000 微秒的分布式存储，单个刷新线程会成为严重瓶颈。

当数据库工作负载执行大量写入系统调用时，数据会以脏页的形式写入内存。KFT 会刷新这些页面，但由于它是单线程，必须等每次写入完成后才能发起下一次写入。在高延迟存储上，脏页的累积速度会超过刷新速度，最终触发无 I/O 脏页节流，限制或暂停写入系统调用。PostgreSQL 和 MySQL 等数据库会以 8 KB 或 16 KB 页面执行非 WAL I/O，这使得分布式存储上的问题更加严重。

根本原因 2：I/O 队列争用。云分布式块存储提供的 I/O 队列数量受到严重限制——AWS EBS 和 Azure 磁盘存储的每个设备都只有两个。这一限制是

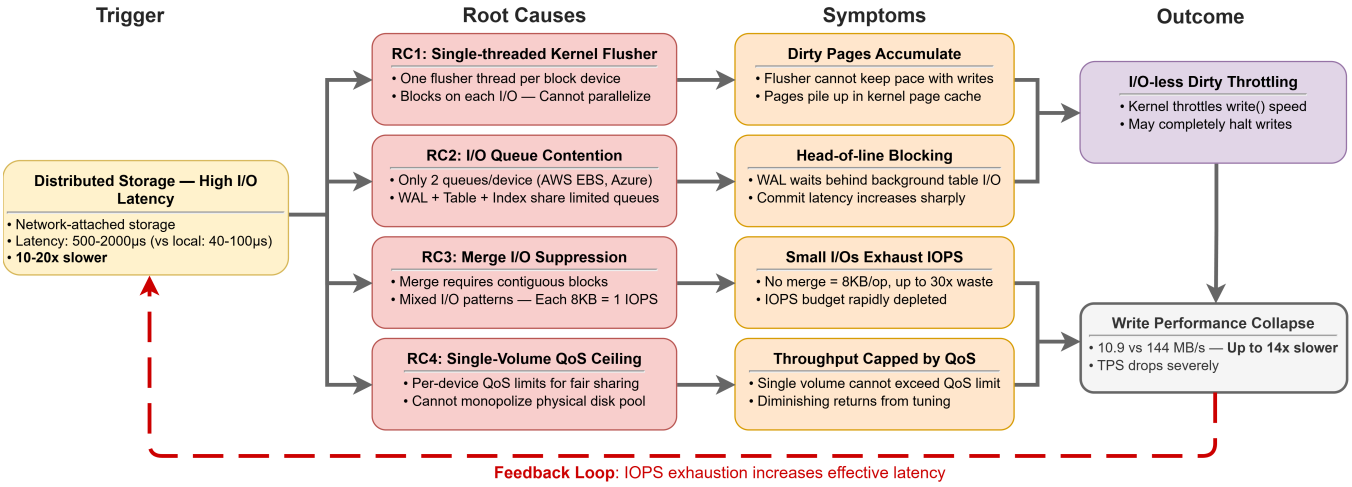


图 6：云数据库的病态级联

一种服务质量设计选择：云存储限制每个设备的队列数量，以确保所有租户都能获得稳定的性能，而不是最大限度地提升任何单个实例的性能。

有限队列处理混合 I/O 命令时，对延迟敏感的预写日志写入会与后台表和索引刷新争抢资源。当对延迟敏感的预写日志操作排在后台表和索引刷新之后时，就会发生队头阻塞，进而增加事务提交延迟。

根本原因 3：合并 I/O 抑制。 如第 2.3.5 节所述，合并 I/O 会将相邻请求整合为更大的操作。然而，合并 I/O 要求数据块在同一 I/O 队列中连续。在单磁盘配置中，预写日志、表和索引的 I/O 模式相互交错，破坏了连续性并抑制合并 I/O。由于无法有效合并，大多数 8 KB 写入各消耗一个 IOPS，迅速耗尽 IOPS 预算。这解释了为何增加 IOPS 分配通常无法解决性能问题——瓶颈在于结构，而非资源不足。

根本原因 4：单卷 QoS 上限。 分布式存储通过实施单设备 QoS 保证，确保租户之间公平共享资源。与用户可充分利用硬件的本地存储不同，单个虚拟磁盘无法独占其跨越的物理磁盘池。这种架构限制造成了吞吐量上限，且该上限独立于卷配置的 IOPS 或带宽。正如第 3.2 节所示，即使使用 8 个线程，Ceph RBD 缓冲 I/O 随机写也仅达到 51.1 MB/s，远低于集群的聚合性能（表 1）。当数据库运行在单个卷上时，即使内核空间瓶颈（RC1–RC3）得到部分缓解，也无法利用更广泛的物理基础设施。这一上限会加剧病态级联：即使提高刷新器吞吐量，单个卷的 QoS 限制仍会限制可实现的 I/O 性能。

4.2 病态级联

这些根本原因形成了一个相互强化的循环：高延迟拖慢单线程刷新器（RC1），导致脏页累积。有限队列中的混合输入/输出（RC2）会抑制合并 I/O（RC3），使大多数 8 KB 写入各消耗一个 IOPS，并进一步增加有效延迟。单卷 QoS 上限（RC4）限制了可实现的最大吞吐量，因此调整内核参数或提高单个磁盘的 IOPS，只能带来有限的改善。这一连锁反应最终会触发无 I/O 脏页节流，使数据库写入性能骤降。先前研究认为，云数据库性能面临的挑战主要源于网络每秒数据包数和带宽限制。Aurora 论文 [60] 设计了存储计算分离架构来解决这些瓶颈，后续的复现研究 [50] 也验证了这一观点。然而，现代虚拟机实例如今可提供 10–100 Gbps 的网络带宽，核心网络设备也能达到 100–400 Gbps，因此网络已远不太可能成为主要瓶颈。我们的分析表明，这种性能问题源于内核空间 I/O 机制和分布式存储 QoS 约束。从这一诊断来看，先前的架构似乎只是间接绕过了这些瓶颈——仅将预写日志从计算节点传输到存储节点——而没有解决根本原因。下一节将介绍 SteelDB，这种架构能够直接解决这些根本原因。

5 处理：STEELDB

本节介绍 SteelDB，这是一种零补丁架构，可直接解决第 4 节中指出的根本原因。我们将零补丁架构定义为：仅通过配置云基础设施来提升数据库性能，而无需对内核或数据库源代码进行任何修改。

5.1 架构概览

SteelDB 将数据库组件分布在多个线上虚拟磁盘上。图 7 展示了该架构的概览。关键观察是，每个虚拟磁盘都有各自的内核刷新线程、I/O 队列和数据局部性域。将组件分布到不同磁盘上，可将这些针对单个设备的限制转化为并行且相互隔离的操作。SteelDB 利用 PostgreSQL 的标准功能——使用表空间将表和索引分配到特定虚拟磁盘，并使用符号链接将预写日志目录放在专用磁盘上。这种方法无需修改 PostgreSQL 或 Linux 内核，同时完全兼容上游版本。

5.2 处理原则

我们根据第 4 节中的诊断总结出四项处理原则。这四项原则均可仅通过标准数据库功能——表空间和符号链接——实现，无需修改数据库或内核代码。

原则 1：并行化刷新操作。将 WAL、表和索引分配到不同的虚拟磁盘，可同时启用多个内核刷新线程，将单线程刷新器瓶颈转变为并行回写。如第 3 节所示，通过多个线程访问分布式存储时，可实现高吞吐量。这使数据库能够利用分布式存储基础设施固有的并行性，正如缓解条件 1 所示。

原则 2：按功能隔离输入/输出队列。将每个数据库组件分配到独立的虚拟磁盘，可为其提供专用的输入/输出队列，从而消除 RC2 中诊断出的混合工作负载拥塞。尤其是，WAL 写入不再排在后台表和索引刷新之后。反过来，表和索引的读取操作也不再受 WAL fsync 等待阻塞。

原则 3：恢复合并 I/O 概率。将数据库组件隔离到专用卷上，可使每个设备的输入/输出队列处理连续块范围，最大限度地增加合并 I/O 机会。如第 3 节所示，合并会将多个 8 KB 写入合并为更大的操作，使 IOPS 消耗降低一个数量级，从而在标准云卷上以更低成本实现更高吞吐量。

原则 4：利用多卷性能聚合。即使未来的内核补丁能够解决 RC1（单线程 KFT），级联效应仍会在 RC4（QoS 上限）处终止，从而证实：只有通过设备层的多卷横向扩展，才能真正释放分布式存储的性能。

借助多个虚拟磁盘，SteelDB 利用了分布式存储的单设备 QoS 保证。由于每个卷都会获得独立的性能配额，聚合吞吐量会随卷数量增加而提升。这一原则利用了分布式存储的设计特性：由于存储成本主要取决于卷总大小，而非卷数量，多卷配置无需额外成本即可实现更高性能。此外，多个卷会横跨存储节点上的更广泛物理磁盘集合，从而访问单个卷无法触达的基础设施容量。

涌现效应：脏页累积预防。四项原则相互强化。原则 1—3 共同解决内核空间瓶颈，而原则 4 则克服基础设施级 QoS 限制。它们共同防止脏页累积——这是无 I/O 脏页节流和写入性能崩溃的触发因素。

5.3 数据放置策略

本节定义了 SteelDB 的数据放置算法。与索引选择和表分区等物理数据库设计问题类似，[43]，在虚拟磁盘上的最佳放置本质上依赖工作负载，不存在通用解决方案。由于大部分性能提升源于架构分离本身，而非特定的表分配，因此我们采用了一种实用的基于评分的启发式方法。

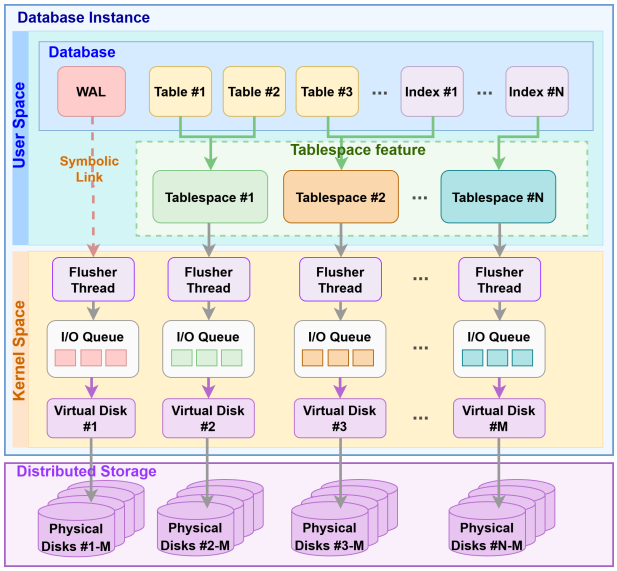


图 7：SteelDB 的架构

步骤 1：确定刷新线程的数量。KFT 的初始数量设为 4。如果虚拟 CPU 核心数为 4 个或更少，则设为 2。KFT 的最大数量限制为虚拟 CPU 核心数的四分之一。此限制可避免线程过多导致性能下降，因为过多的线程会增加上下文切换开销和线程管理成本。在实践中，本文评估的所有配置中，4 个 KFT 已足够。

步骤 2：为 WAL 分配专用资源。为 WAL 分配专用的 I/O 队列和 KFT，并赋予最高优先级。此分配方式可最大限度提高 WAL 写入的合并 I/O 效率，并将预写日志 I/O 延迟降至最低——这两者都会直接影响事务提交时间。此外，初始时会为所有表和索引分配一个 KFT。

步骤 3：放置表和索引。如果 KFT 数量不少于 4，则使用贪心评分算法确定分配方案。令 t_i 表示第 i 个表或索引。运行与预期工作负载相似的基准测试，并使用数据库统计工具 [23, 49] 和 iostat，获取每个 t_i 的写入频率 W_{t_i}、读取频率 R_{t_i} 和数据大小 S_{t_i}。随后计算以下分数：

$$Score(t_i) = \alpha \cdot N(W_{t_i}) + \beta \cdot N(R_{t_i}) + \gamma \cdot N(S_{t_i}), \quad (1)$$

其中，$N(\cdot)$ 将数值归一化为 $[0, 1]$，，权重分别为 $\alpha = 0.5$、$\beta = 0.3$ $\gamma = 0.2$ 和 。按分数对表和索引排序，并按优先级依次分配给 KFT。为确保分布均衡，将检查各 KFT 上分数的变异系数（CV）：如果 CV 超过 0.3 且尚未达到 KFT 数量上限，则增加一个 KFT，并重新计算分配方案，直至满足条件。

步骤 4：虚拟磁盘性能调优。在云环境中，可以根据工作负载特征为每个卷分配输入/输出性能。高性能 I/O 成本高昂，因此必须进行有选择的分配。超低延迟高成本存储（例如 io2）非常适合存放预写日志（WAL），因为它可以降低写入延迟并提升事务吞吐量。相比之下，表和索引的延迟要求较低，因此吞吐量和 IOPS

效率才是首要考量。与其使用单个高性能磁盘，不如使用多个经济实惠的基础层虚拟磁盘（例如 gp3），从而实现兼顾性能与成本的输入/输出优化。

尽管这个分配问题与多维背包问题相似，但与实现 SteelDB 架构及其处理原则相比，最优放置并非那么关键。通过所提出的评分算法手动推导近似解，已足以为实际部署带来显著收益。在工作负载波动的情况下进行动态再平衡和在线数据迁移，仍是值得探索的未来方向。

5.4 利用 DistributedBlockStorage 特性

SteelDB 利用分布式存储的特性，超越单卷性能。

多卷并行。 如原则 4（第 5.2 节）所述，分配多个卷能够实现随卷数量扩展的聚合性能。分布式存储为每个设备提供性能保证；将多个卷分配给同一实例时，这些性能保证会累积，从而实现单个卷无法达到的聚合吞吐量。例如，使用四个 100GB 虚拟磁盘，每个磁盘均配置为标准的 3,000 IOPS 和 125 MB/s 带宽，可实现 12,000 IOPS 和 500 MB/s 的综合性能。通过了解分布式存储的内部机制，可以理解这一策略为何有效。在 Ceph 中，当一个放置组可以占用大量磁盘空间时，一个虚拟卷所跨越的物理磁盘数量可能超过副本数。假设单个卷可以同时访问八块物理磁盘，那么连接 10 个虚拟磁盘便可访问 80 块物理磁盘。

现代云基础设施。 现代云环境通过提供专用于存储的 10–100 Gbps 网络带宽 [3]，并支持每个实例挂载 27–128 个虚拟磁盘 [4]，为这一策略提供了支持。由于存储成本主要取决于卷的总大小，而非卷的数量，因此将数据分散到多个较小的磁盘上不会产生额外成本。

5.5 零补丁优势

SteelDB 的零补丁架构具有多项实际优势：

诊断清晰性与可复现性。 由于性能提升完全源自资源分配，而非代码修改，因此这验证了我们的诊断：内核空间 I/O 机制是导致性能问题的根源。任何 PostgreSQL 部署都可以利用标准功能和公开可用的云基础设施复现这些结果。

生产安全性。 政府机构和金融机构等要求严格维护软件完整性的组织，可以在不影响安全合规的前提下采用 SteelDB。

免除维护。 Aurora 等存算分离数据库需要投入大量工程精力，以维持与上游 PostgreSQL 版本的兼容性。SteelDB 消除了这一负担，使用户能够立即采用新版 PostgreSQL 和安全补丁。

完整的生态系统兼容性。 SteelDB 与 PostgreSQL 的工具、扩展和监控解决方案完全兼容，保留开源生态系统的全部优势。

读扩展能力。 SteelDB 通过流复制继承了 PostgreSQL 原生的读扩展能力。无需任何修改即可部署只读副本；由于读扩展与 SteelDB 的写入路径优化彼此独立，两者协同解决了 OLTP 可扩展性的两个方面。

6 评估

我们通过 TPC-C 基准测试以及与商业云数据库的比较来评估 SteelDB。

6.1 实验设置

我们使用 HammerDB [35]运行 TPC-C [59]工作负载，这是一种广泛采用的 OLTP 基准测试。数据库大小设置为 WH1,000，虚拟用户数设置为 100。基准测试运行时间为 20 分钟，之前进行了 5 分钟的预热。本文采用的评估指标为每分钟新订单数（NOPM）。我们排除 MySQL，因为其基准测试结果受许可证限制 [48]，且 InnoDB 的自适应刷写与 SteelDB 的多磁盘策略相冲突。SteelDB 使用标准 AWS EBS（gp3，3,000 IOPS，125 MB/s）。作为基线，我们纳入了单磁盘配置、使用四块磁盘的 RAID 0、IOPS 为 12,000 且带宽为 500 MB/s 的 gp3 配置，以及混合双磁盘配置。数据库服务器使用 c6in.8xlarge 虚拟机实例（32 个虚拟 CPU、128 GB 内存、25 Gbps EBS 带宽）。客户端实例为 m6i.2xlarge（8 个虚拟 CPU、32 GB 内存）。PostgreSQL 配置参数如图 8 所示。所有内核参数（包括脏页回写设置）均保持默认值。在 Ceph RBD 实验中，数据库服务器和客户端均配置为 32 个虚拟 CPU 和 64 GB 内存。图 5 给出了 Ceph 集群的规格。

SteelDB 中的磁盘分配。 对于 EBS，预写日志被分配到优先级最高的专用磁盘。在四磁盘配置中，库存表和订单_明细表根据其较高的写入频率被分配到不同的磁盘上。对于 Ceph RBD，预写日志存储在 SATA 连接的固态硬盘上，而表数据和索引数据存储在 RBD 上。所有虚拟磁盘的卷大小均统一为 200 GB，以确保性能一致。

6.2 AWS EBS 上的处理效果

表 2 展示了 AWS EBS 上的实验结果。我们围绕第 4 节中确定的四个根本原因展开分析。

总体性能。 四磁盘 SteelDB 配置（P2）达到了 242,107 NOPM，而单磁盘基线（B1）为 69,894 NOPM，性能提升了 3.5 倍。双磁盘 SteelDB 配置（P1）仅使用 6,000 IOPS 和 250 MB/s，却达到了 154,150 NOPM，表明即使是部分处理也能带来可测量的收益。

RC1 验证：单线程刷新器。四磁盘 RAID 0 配置（B2）使用的磁盘数量与聚合规格均与 SteelDB（P2）相同，但由于 RAID 向内核呈现一个逻辑设备，因此只使用一个 KFT。RAID 0 的 NOPM 为 123,981，而 SteelDB 为 242,107，相差 2 倍。这证实，带来性能提升的是刷新器并行化（原则 1），而非单纯增加带宽。表 4 进一步支持了这一发现：

表 2：EBS 上的 TPC-C 基准测试结果

方法	KFTs	IOPS	带宽（MB/s）	NOPM
B1 1块磁盘	1	3,000	125	69,894
B24块磁盘（RAID 0）	1	12,000	500	123,981
B3 1块磁盘（增强版）	1	12,000	500	163,698
B4 2块磁盘（WAL拆分）	2	12,000	500	179,092
P12块磁盘（SteelDB）	2	6,000	250	154,150
P24块磁盘（SteelDB）	4	12,000	500	242,107

SteelDB 的 CPU 用户利用率更高（14.8% 对比 10.4%），这表明数据库进程将更多时间用于有效计算，而非等待输入/输出。

RC2 验证：I/O 队列争用。表 4 显示，SteelDB 的 WAL 磁盘平均 I/O 队列长度为 1.8，而单个增强型磁盘配置（B3）为 14.0。这一 8× 降幅证实，输入/输出队列隔离（原则 2）消除了 RC2 中诊断出的队头阻塞。借助专用队列，WAL 写入能够以极低的排队延迟完成，直接改善事务提交延迟。

RC3 验证：合并 I/O 抑制。在 SteelDB 的专用 WAL 磁盘中，56.2% 的 I/O 操作经过合并，平均 I/O 大小达到 75.5 KB，相当于每次操作有效合并了约十个 8 KB 的 WAL 页面。这验证了原则 3：基于功能的数据放置可恢复合并 I/O 概率。相比之下，单磁盘配置会在共享队列上混合 WAL、表和索引的 I/O 模式，破坏合并 I/O 所需的连续性。

RC4 验证：单卷 QoS 上限。 SteelDB 的四块标准 gp3 磁盘（P2：总计 12,000 IOPS、500 MB/s）优于一块聚合规格相同的增强版 gp3 磁盘（B3：12,000 IOPS、500 MB/s），NOPM 达到 242,107，而后者为 163,698，差距达 1.5 倍。P2 与 B3 之间 1.5× 的差距——尽管聚合 IOPS 和带宽相同——反映了全部四项根本原因的影响。然而，B3 即使在单个卷上达到 12,000 IOPS，仍无法匹敌 P2，这与 RC4 中诊断出的 QoS 上限相符。

协同效应。 B4（双磁盘 WAL 拆分，179,092 NOPM）与 P2（四磁盘 SteelDB，242,107 NOPM）的对比表明，四项原则结合起来，比单独解决任何一项根本原因带来更大的提升。B4 将 WAL 移至专用磁盘，针对 WAL 应用了原则 1 和 2，但没有隔离表和索引的输入/输出。SteelDB 将输入/输出隔离和合并 I/O 恢复扩展到所有数据库组件，相比部分处理额外提升了 35%。

6.3 Ceph RBD 上的处理效果

表 3 表 3 展示了 Ceph RBD 上的结果，证实了相同的趋势。SteelDB RBDx4 达到 384,062 NOPM，而单个 RBD 基线为 181,319 NOPM，性能提升了 2.1 倍。相比之下，尽管使用了四块磁盘，RAID 0 RBDx4 的 NOPM 仅为 300,639。SteelDB 与 RAID 0 之间的差异证实，推动性能提升的是输入/输出队列分离和刷新器并行化，而非单纯的带宽聚合。这些结果在两种独立的存储实现——EBS 和 Ceph RBD——上均成立，说明我们的诊断具有普遍性：根本原因在于内核输入/输出机制与分布式块存储之间的交互，而非任何特定的实现。

表 3：Ceph RBD 上的 TPC-C 基准测试结果

方法	WAL 磁盘	KFT	NOPM
B1 SATA 固态硬盘	SATA 固态硬盘	1	220,109
B2 Ceph RBD x1	SATA 固态硬盘	2	181,319
B3 Ceph RBD x4（RAID 0）	SATA 固态硬盘	2	300,639
P1Ceph RBD x2（SteelDB）	SATA 固态硬盘	3	250,259
P2Ceph RBD x4（SteelDB）	SATA 固态硬盘	5	384,062

表 4：EBS 基准测试中的 CPU 使用率分析

方法	%usr	%iowait	平均 I/O 队列 WAL 磁盘中的大小
B31块磁盘（增强版）	10.4	8.82	14.0
P2 4块磁盘（SteelDB）	14.8	20.1	1.8

```
shared_buffers=4096MB，work_mem=32MB，maintenance_work_mem=1024MB，
checkpoint_timeout=5分钟，checkpoint_completion_target=0.9，wal_init_zero=no，
wal_recycle=no，max_wal_size=8GB，min_wal_size=8GB
```

图 8：PostgreSQL 配置参数

6.4 与云数据库的比较

我们将 SteelDB 与已在商业环境中部署的存算分离数据库进行比较。

与 Amazon Aurora 的比较。 我们将 SteelDB 与 Amazon Aurora 进行了对比评估。对于 PostgreSQL [9], , Aurora 是一种具有代表性的存储与计算分离数据库。Aurora 配置为采用 I/O 优化 [2]存储，该存储专为高输入/输出工作负载设计，且仅按使用时长计费，与输入/输出量无关。由于 TPC-C 的输入/输出密集度高，且包含频繁的小规模写入和提交，因此 I/O 优化是针对该工作负载最有利的 Aurora 配置，可确保比较公平。

我们评估了两种 SteelDB 配置：仅使用 gp3 的配置（23 个卷，每个卷为 3,000 IOPS / 125 MB/s）和混合配置（6 个 io2 卷，每个卷为 1,000 IOPS + 17 gp3 卷）。通过 mdadm 将卷分组为四个 RAID 0 阵列：6 个用于预写日志，8 个用于库存表，6 个用于订单明细表，3 个用于其余表；预写日志的块大小为 8 KB，其余为 32 KB。为匹配提升后的磁盘性能，我们将 max_wal_size 参数提高至 32 GB。

SteelDB、Aurora 和 AlloyDB 的月度成本计算基于截至 2024 [6] [30] 年 11 月东京区域的定价。

表 5 总结了结果。SteelDB 在所有配置下的表现均优于 Aurora。在 r6i.8xlarge 上采用 io2 混合配置时，SteelDB 的性能提升了 3.1 倍，同时成本降低了 58%——性价比提升了 7.3× 倍。即使仅使用标准 gp3 卷，运行在 16 虚拟 CPU 实例上的 SteelDB 也比运行在 32 虚拟 CPU 实例上的 Aurora 快 67%——以约六分之一的成本实现了更高的性能。将 c6in.8xlarge 的 gp3 配置与单磁盘基准配置相比，SteelDB 的性能提升了 9 倍，且无需增加 IOPS 或带宽成本。

值得注意的是，即使在 SteelDB 的性能比 Aurora 高出 3.1 倍的配置下，网络也并非主要瓶颈。这一实验结果证实，在现代云基础设施下，先前研究指出的网络带宽和每秒数据包数限制已不再 [50, 60] 限制性能

表 5：TPC-C 基准测试结果的性能比较：SteelDB 与其他云数据库 ses

数据库实例		磁盘	EBS 带宽 KFT		NOPM 月度成本（美元）	
Aurora	r6i.4xlarge（16 虚拟 CPU，128GB）	I/O 优化	最高 10Gbps	N/A	236,011	2,657.2
SteelDB	r6i.4xlarge（16 虚拟 CPU，128GB）	gp3 x23	最高 10Gbps	4	505,601	887.7
SteelDB	r6i.4xlarge（16 虚拟 CPU，128GB）	io2 x6 +gp3 x17	最高 10Gbps	4	580,256	1,331.7
Aurora	r6i.8xlarge（32 虚拟 CPU，256GB）	I/O 优化	10Gbps	N/A	302,838	5,314.4
SteelDB	r6i.8xlarge（32 虚拟 CPU，256GB）	gp3 x23	10Gbps	4	637,532	1,775.4
SteelDB	r6i.8xlarge（32 虚拟 CPU，256GB）	io2x6 +gp3 x17 10Gbps		4	926,589	2,219.4
SteelDB	c6in.8xlarge（32 虚拟 CPU，64GB）	gp3 x23	25Gbps	4	638,790	1,667.9
SteelDB	c6in.8xlarge（32虚拟 CPU，64GB）	io2x6 +gp3 x17 25Gbps		4	930,640	2,111.9
AlloyDB	GCP 实例（64 虚拟 CPU，512GB）不可变		N/A	N/A	841,035	9,311.3
SteelDB	r6i.16xlarge（64 虚拟 CPU，512GB）	gp3 ×23	20Gbps	4	640,556	3,550.7
SteelDB	r6i.16xlarge（64 虚拟 CPU，512GB）	io2x6 +gp3 x17 20Gbps		4	1,210,515	3,994.7

——这验证了我们的诊断：真正的瓶颈在于内核空间 I/O 机制。

此外，TPC-C 数据库大小（WH1000）约为 100 GB。在仅有 64 GB 内存的 c6in.8xlarge 实例上，由于数据库无法完全装入内存，SteelDB 的性能仍优于 Aurora，这表明 SteelDB 的性能提升源于 I/O 优化，而非内存缓存效应。

与 GCP AlloyDB 的比较。2024年，谷歌 Cloud 与 Accenture 发布了一份官方白皮书 [31]，评估 AlloyDB 与 Amazon Aurora、OCI ATP [47], 和 OCI MySQL 等竞争云数据库的表现。我们将其作为参考，原因如下：(1) 这是 AlloyDB 开发者发布的官方基准测试，确保配置经过最佳调优且有利于 AlloyDB；(2) 它采用了与我们评估相同的基准测试方法——使用 HammerDB、TPC-C 模型和 WH1000 数据库规模；(3) AlloyDB 在其评估中胜过所有竞争数据库，因此构成了最强的可用基线。根据该白皮书，AlloyDB 在一台配备 64 个虚拟 CPU、512 GB 内存的实例上，分别使用 1、16、32、64、128 和 256 个虚拟用户进行评估，并在 256 个用户时达到 841,035 NOPM 的峰值性能。用户数较少时，性能有所下降。相比之下，SteelDB 在相同实例规格下，仅使用 100 个虚拟用户便达到 1,210,515 NOPM。AlloyDB 自身的结果显示，用户数较少时吞吐量会下降，且其峰值需要 256 个用户。尽管如此，SteelDB 仍以低 57% 的成本实现了高出 1.4× 的吞吐量。即使使用规格减半的实例，SteelDB 的表现也超过了 AlloyDB 的峰值：r6i.8xlarge（32 个虚拟 CPU、256 GB、10 Gbps EBS 带宽）达到 926,589 NOPM，c6in.8xlarge（32 个虚拟 CPU、64 GB、25 Gbps EBS 带宽）达到 930,640 NOPM——两者均超过 AlloyDB 在配备 64 个虚拟 CPU、512 GB 内存的实例上取得的 841,035 NOPM。这证实了性能提升源于 I/O 优化，而非硬件扩展。Aurora 和 AlloyDB 分别在最有利的条件下接受评估——I/O 优化存储和开发者调优配置；而 SteelDB 的所有测试均使用默认设置的标准 gp3 卷（ 3,000 IOPS / 125 MB/s ），这是 AWS 上成本最低的等级。尽管条件并不对等，SteelDB 仍胜过两者。将 gp3 扩展至 16,000 IOPS（截至 2026 [5]为 80,000 IOPS）还能进一步提升性能，这表明这些结果只是 SteelDB 潜力的下限。

资源利用率分析。图9展示了在采用 io2+gp3 混合磁盘配置的 r6i.16xlarge 实例上运行 TPC-C 基准测试期间的资源利用情况。磁盘 I/O 面板几乎全是写入流量，这与 WH1000 数据集（约 100 GB）完全驻留在 512 GB 主存中的情况一致，因此不会产生存储层读取 I/O。CPU 利用率主要由 %usr 占据；由于 KFT 和 I/O 队列活动，%system 也占据了不小的比例，这是 SteelDB 多磁盘架构的典型表现。%iowait 始终保持较低水平，这与多磁盘架构的高效写入能力以及数据集完全驻留在内存中的情况相符。请注意，解读分布式存储上的 %iowait 时应谨慎，因为 Linux 不会将所有由网络引起的等待时间都计入此指标 [32]。在 MD 层（RAID 层），聚合 I/O 性能平均达到 110,078 IOPS（峰值 219,897 IOPS）和 1,046 MB/s（峰值 2,872 MB/s）。这些数值超过了 EBS 的标称性能限制，这归功于合并 I/O 的高效运行以及突发积分机制 [51]，二者均符合 AWS 规范。合并后的实际磁盘层 IOPS 平均为 42,366（峰值 89,929）。合并 I/O 速率证实原则 3 有效发挥了作用：WAL 磁盘组的平均合并率为 83.2%（峰值 96.0%），而其他磁盘组的平均合并率为 14.4% 至 25.3%（峰值 87.4%–92.5%）。在整个基准测试期间，脏页水平始终保持较低，且未进行任何内核参数调优，这证实了四项处理原则协同作用，能够在生产规模工作负载中防止脏页累积。

对其他工作负载的适用性。 第 4 节中诊断出的四个根本原因完全存在于内核—存储接口内部，与其上运行的工作负载无关。任何在分布式块存储上发出缓冲 I/O 的数据库，都会受到同样的病态级联影响，无论其访问模式属于事务型、键值型还是分析型。例如，YCSB [21],由简单的键值操作构成，查询处理开销极低，因此其吞吐量很大程度上取决于原始存储 I/O 性能。写密集型工作负载将直接受益于刷新器并行化和合并 I/O 的恢复，而读密集型工作负载则受益于 I/O 队列隔离，因为这可以消除队头阻塞。当数据集可容纳于内存中时，读取请求由缓冲区缓存提供服务——但 Aurora 和 AlloyDB 也同样如此，因此性能差异仍然

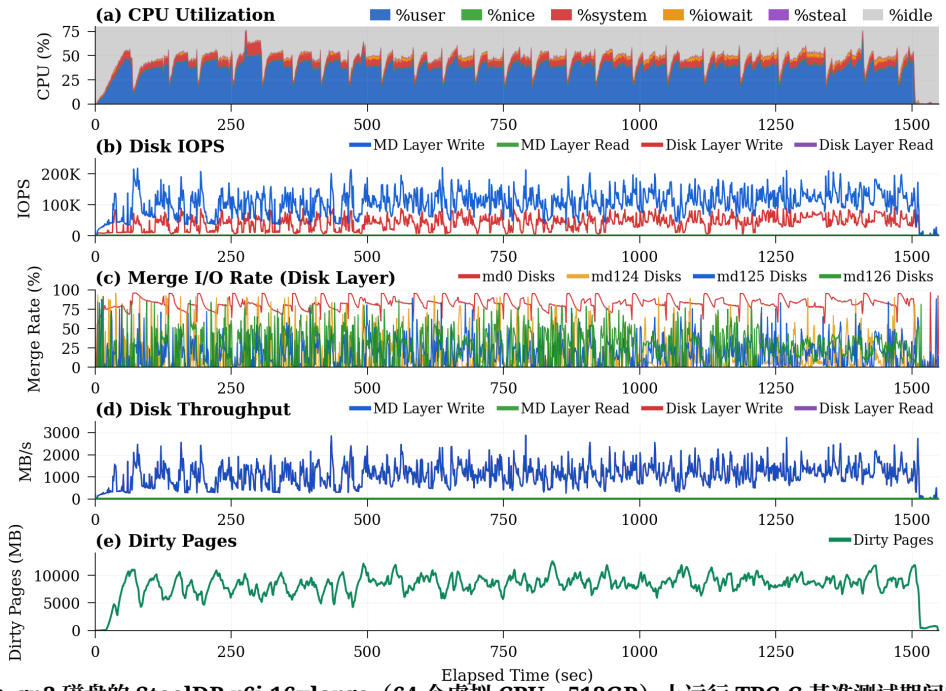


图9：在配备 io2+gp3 磁盘的 SteelDB r6i.16xlarge（64 个虚拟 CPU，512GB）上运行 TPC-C 基准测试期间的资源利用率

表6：PostgreSQL 版本发布延迟比较

数据库	主版本延迟 [天]			次版本延迟 [天]		
	Avg	Max	Min	Avg	Max	Min
Aurora	254	405	90	56	120	22
AlloyDB	426	438	405	153*	不适用**	不适用**
SteelDB	≈0	≈0	≈0	≈0	≈0	≈0

*基于 AlloyDB 15.7 和 14.12 版本 **无法计算；许多次要版本尚未发布

取决于写入路径效率，而 SteelDB 的优化在这方面最为显著。图9中的总 I/O 吞吐量（平均 1,046 MB/s，峰值 2,872 MB/s）证实，无论访问模式如何，该处理都能消除结构性 I/O 上限。对其他基准测试进行定量评估仍有待未来开展。

综合来看，这些结果表明，过去十年来存储计算分离的趋势，实则是为了应对内核空间瓶颈，而这些瓶颈无需专有存储或代码修改即可解决。真正的问题出在内核空间 I/O 机制上；SteelDB 的零补丁处理以极低的成本带来了更出色的性能，同时免去了专有存储和补丁维护所带来的工程负担。

6.5 软件维护成本分析

零补丁架构消除了软件维护开销。存算分离数据库必须针对每个新的 PostgreSQL 版本应用专有补丁。我们分析了 Aurora [10] 和 AlloyDB [28] 的发布计划，以量化这项负担。表 6 总结了结果。PostgreSQL 社区每年发布一个主版本，每季度发布一个次版本。截至 2025年3月3日——距离 PostgreSQL 17 于 2024年9月26日发布已逾五个月——Aurora 和 AlloyDB 均不支持 PostgreSQL 17，而 SteelDB 可在新版本发布后立即部署。Aurora 的主版本延迟平均为 254 天（最长 405 天），次版本延迟平均为 56 天。AlloyDB 的延迟更长：主版本延迟平均为 426 天（最长 438 天），甚至超过了年度发布周期，而且许多次版本始终未能发布。尽管 AWS 和 Google Cloud——全球顶尖的云服务提供商——拥有雄厚的工程资源，这些延迟依然存在，表明基于补丁的维护成本天生高昂。这些延迟减缓了应对安全漏洞的速度，对于受严格合规要求约束的政府机构和金融机构而言，这是一个值得关注的问题。SteelDB 消除了这些成本，使其能够立即部署新的 PostgreSQL 版本，并与上游版本保持二进制级一致性。

6.6 关于可用性的讨论

一种自然的担忧是，增加虚拟磁盘的数量会引入更多故障点，从而降低系统持久性。我们从数学上证明，这一直觉并不正确：卷分区不会影响系统级故障概率。分布式存储中的虚拟磁盘通过基于软件的配额进行管理，而非依据物理磁盘边界进行管理。设块级故障概率为 p，副本数为 r，总容量为 B 个块，划分为 k 个卷。每个卷的故障概率为 $(1-p^r)$。系统级故障概率为：

$$P_{\text{system}} = 1-(1-P_i)^k = 1-(1-p^r)^B, \qquad (2)$$

k 项相互抵消，证明 P_{system} 仅取决于总容量 B，而不取决于分区数量 k。此外，较小的卷可以缩小单次故障的影响范围并缩短恢复时间。实际上，分布式存储具有很高的持久性。AWS gp3 卷的年故障率（AFR）低于 0.2% [15]，，io2 卷则低于 0.001% [5]，显著低于标准机械硬盘通常 1% 的 AFR。

对于需要更高可用性的系统，可以采用一些标准措施，例如跨可用区进行流复制、将预写日志归档到外部存储（如 S3），或通过购买额外的 IOPS 来减少虚拟磁盘数量。这些措施与 SteelDB 正交，可提供与传统配置相当的可用性。

关于运维复杂性，配置并挂载多个卷是一次性操作，可借助 Terraform [36], Pulumi [53], 和 Ansible [55]等工具轻松自动化。如果 SteelDB 作为托管服务提供，用户根本不会察觉到这种复杂性——正如 Aurora 用户不了解其内部的多节点存储架构一样。

7 相关工作

存算分离数据库。Amazon Aurora [60] ，于 2015年发布的 Amazon Aurora 为后续云原生数据库奠定了范本，包括 Azure Hyperscale [11],PolarDB [17], Neon [44], 和 GCP AlloyDB [27]。Aurora 引入了一种分离计算层和存储层的架构：计算节点只向存储节点写入预写日志，数据页则在存储节点重建。Aurora 论文将其写入性能提升归因于避免了网络带宽和每秒数据包数瓶颈。Aurora 的若干效果仍有待商榷，因此 SIGMOD 2024 [50]开展了一项复现研究，独立验证了关键设计选择，并阐明了其性能表现。然而，Aurora 论文和复现研究都完全没有探讨云分布式存储延迟导致的内核刷新线程瓶颈，也没有讨论内核空间中的 I/O 队列争用或合并 I/O 抑制——这些正是本文指出的根本原因。以往研究完全忽视了这些内核空间瓶颈。此前的架构并未识别并解决这些根本原因，而是采用计算节点仅写入预写日志的方式，间接绕过内核空间的异常，却没有意识到其存在。我们的分析表明，归因于网络限制的性能下降，更准确地说是由内核空间 I/O 机制所致。SteelDB 采用通用分布式块存储和零补丁架构，直接解决这些根本原因。由此既能实现成本效益，又能快速进行安全更新——而背负补丁维护负担的专有存储方案无法做到这一点。

分布式事务数据库。像 CockroachDB [57],、Spanner [22],、PolarDB-X [16],、TiDB [37], 、YugabyteDB [67], 和 OceanBase [66] 这样的分布式事务数据库，通过在多个节点间分配事务共识，实现可扩展的性能和高可用性。然而，这些系统在常见 SQL 模式（分布式连接操作、分组）上会产生开销，限制了它们作为通用 OLTP 数据库的应用。在云基础设施上，它们还面临双重延迟问题：节点间基于网络的共识，以及通过网络访问分布式块存储，这两者叠加后会显著增加每操作延迟，降低单节点性能。我们的工作针对的是另一个问题，并采用了根本不同的方法。

共享存储数据库。 Oracle RAC [19]和 PolarDB-MP [65]等共享存储数据库支持多主访问通过共享存储实现，但需要复杂的分布式锁管理以及高性能共享存储，这会在云环境中产生极高的成本。

数据库与内核输入/输出的演进，以及 CloudJump。在 20 世纪 90 年代初，带回写缓存的硬件 RAID 控制器尚未普及，数据库会通过表空间将数据分布到多个磁盘上，以实现输入/输出并行性。随着硬件 RAID 控制器的引入，业界标准做法转向使用单卷 RAID 阵列；这种单卷方式至今仍占主导地位——Amazon RDS [8], 、Google Cloud SQL [29], 和 Azure Database 均不支持多卷架构。

与此同时，Linux 内核也经历了显著演进。2009 年的一项补丁 [63]通过为每块磁盘分配一个 KFT，改进了内核刷新线程的管理；此前，所有磁盘共用一个 KFT。2013 年的一项补丁 [12] 引入了多队列块 I/O（blk-mq）支持，并在内核 4.19 [13], 中成为默认机制，从而实现并行 I/O 处理。近期研究还探讨了缓冲 I/O 优化 [39] ，以及在分布式文件系统中结合缓冲 I/O 和直接 I/O 的方法 [54]。SteelDB 基于这些内核改进 ——每设备 KFT 和多队列 I/O——同时针对云分布式存储的具体特性进行优化。在这些基础机制之上进行策略性数据放置，能够带来简单数据分布无法实现的性能提升。因此，SteelDB 并非恢复面向本地带宽的旧式多磁盘数据放置方案，而是针对仅存在于分布式块存储中的内核空间瓶颈；目前，云托管数据库均不支持这种配置。

CloudJump [20] 通过直接 I/O 补丁加速 MySQL 在云块存储上的运行，但并未解决本文指出的 KFT 瓶颈或 I/O 队列限制。

8 结论

我们对云端 OLTP 数据库中的写入性能退化进行了病态分析。通过对分布式块存储开展受控实验，我们确定了四个根本原因——单线程刷新器的局限、I/O 队列争用、合并 I/O 抑制，以及单卷 QoS 上限。基于这一诊断，我们验证了 SteelDB：一种跨层零补丁编排架构，仅通过策略性数据放置，协调数据库、内核和分布式块存储层，从而解决这些根本原因。这种病态级联源于这些层之间的相互作用——单独检查任何一层时都无法察觉，这也解释了为何以往研究忽略了这些瓶颈。TPC-C 评估表明，SteelDB 的性能最高可提升 9 倍。与 Aurora 相比，SteelDB 的性能提高了 3.1 倍，成本降低了 58%，成本效益提升了 7.3 倍。与 AlloyDB 相比，其吞吐量提高了 1.4 倍，成本降低了 57%。这些结果是在采用标准 gp3 和 io2 卷的最低成本配置下实现的。零补丁架构消除了软件维护开销：Aurora 的主要版本升级平均需要 254 天，而 SteelDB 可立即采用新版本。通过揭示缓冲 I/O 与高延迟分布式存储相互作用所产生的瓶颈，本研究表明，高性能云数据库可以基于通用内核原语构建，无需专有存储或定制补丁。这一视角为云数据库研究开辟了新方向。

参考文献

[1] Abutalib Aghayev、Sage Weil、Michael Kuchnik、Mark Nelson、Gregory R. Ganger 和 George Amvrosiadis。2019年。文件系统不适合作为分布式存储后端：Ceph 十年演进的经验教训。载于第 *27* 届 *ACM* 操作系统原理研讨会论文集（*SOSP '19*）。ACM，美国纽约州纽约市，353–369。[2] 亚马逊云科技。2024年。Amazon Aurora 存储 - Amazon Aurora。https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Overview.StorageReliability.html#aurora-storage-type 访问日期：2026年3月30日。[3] 亚马逊云科技。2024年。Amazon EBS 优化型实例类型。https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-optimized.html 访问日期：2026年3月30日。[4] 亚马逊云科技。2024年。Amazon EC2 实例的 Amazon EBS 卷限制。https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/volume_ limits.html 访问日期：2026年3月30日。[5] 亚马逊云科技。2024年。Amazon EBS 卷类型 - Amazon Elastic Compute Cloud。https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs- volume-types.html 访问日期：2026年3月30日。[6] 亚马逊云科技。2024年。AWS 定价计算器。https://calculator.aws/#/ 访问日期：2026年3月30日。[7] 亚马逊云科技。2024年。云块存储 - Amazon EBS - AWS。https: //aws.amazon.com/ebs/ 访问日期：2026年3月30日。[8] 亚马逊云科技。2024年。托管 SQL 数据库 - Amazon 关系数据库服务（RDS）- AWS。https://aws.amazon.com/rds/ 访问日期：2026年3月30日。[9] 亚马逊云科技。2024年。OLTP 数据库、MySQL 和 PostgreSQL 托管数据库 - Amazon Aurora - AWS。https://aws.amazon.com/rds/aurora/?nc1=h_ ls 访问日期：2026年3月30日。[10] 亚马逊云科技。2024年。Aurora PostgreSQL 发布日历 - Amazon Aurora。https://docs.aws.amazon.com/AmazonRDS/latest/AuroraPostgreSQLReleaseNotes/aurorapostgresql-release-calendar.html 访问日期：2026年3月30日。[11] Panagiotis Antonopoulos、Alex Budovski、Cristian Diaconu、Alejandro Hernan- dez Saenz、Jack Hu、Hanuma Kodavalla、Donald Kossmann、Sandeep Lingam、Umar Farooq Minhas、Naveen Prakash、Vijendra Purohit、Hugh Qu、Chai- tanya Sreenivas Ravella、Krystyna Reisteter、Sheetal Shrotri、Dixin Tang 和 Vikram Wakade。2019年。Socrates：云端的新型 SQL Server。载于*ACM*国际数据管理会议论文集（*SIGMOD '19*）。ACM，美国纽约州纽约市，1743–1756。[12] Jens Axboe。2013 年。blk-mq：新的多队列块 I/O 排队机制 · torvalds/linux@320ae51。https://github.com/torvalds/linux/commit/320ae51feed5c2f13664aa05a76bec198967e04d 访问日期：2026年3月30日。[13] Jens Axboe。2017 年。scsi：默认使用 scsi-mq · tor- valds/linux@5c279bd。https://github.com/torvalds/linux/commit/5c279bd9e40624f4ab6e688671026d6005b066fa 访问日期：2026年3月30日。[14] Jens Axboe。2024年。灵活 I/O 测试器。https://github.com/axboe/fio 访问日期：2026年3月30日。[15] Marc Brooker、Tao Chen 和 Fan Ping。2020年。数百万个微型数据库。载于第 *17* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 20*）。USENIX协会，加利福尼亚州圣克拉拉，463–478。[16] Wei Cao、Feifei Li、Gui Huang、Jianghang Lou、Jianwei Zhao、Dengcheng He、Mengshi Sun、Yingqiang Zhang、Sheng Wang、Xueqiang Wu、Han Liao、Zilin Chen、Xiaojian Fang、Mo Chen、Chenghui Liang、Yanxin Luo、Huanming Wang、Songlei Wang、Zhanfeng Ma、Xinjun Yang、Xiang Peng、Yubin Ruan、Yuhui Wang、Jie Zhou、Jianying Wang、Qingda Hu 和 Junbin Kang。2022年。PolarDB-X：面向云原生应用的弹性分布式关系数据库。载于*IEEE* 数据工程国际会议论文集（*ICDE '22*）。IEEE，马来西亚吉隆坡，2859–2872。[17] Wei Cao、Yingqiang Zhang、Xinjun Yang、Feifei Li、Sheng Wang、Qingda Hu、Xuntao Cheng、Zongzhi Chen、Zhenjun Liu、Jing Fang、Bo Wang、Yuhui Wang、Haiqing Sun、Ze Yang、Zhushi Cheng、Sen Chen、Jian Wu、Wei Hu、Jianwei Zhao、Yusong Gao、Songlu Cai、Yunyang Zhang 和 Jiawang Tong。2021年。PolarDB Serverless：面向解耦式数据中心的云原生数据库。载于*ACM*国际数据管理会议论文集（*SIGMOD '21*）。ACM，美国纽约州纽约市，2477–2489。[18] Joe Celko。2010年。*Joe Celko* 的《*SQL for Smarties*》第四版：高级 *SQL* 编程。Morgan Kaufmann，美国马萨诸塞州伯灵顿。[19] S. Chandrasekaran 和 R. Bamford。2003年。共享缓存——并行数据库的未来。载于第 *19* 届国际数据工程会议论文集（*ICDE '03*）。IEEE，印度班加罗尔，840–850。[20] Zongzhi Chen、Xinjun Yang、Feifei Li、Xuntao Cheng、Qingda Hu、Zheyu Miao、Rongbiao Xie、Xiaofei Wu、Kang Wang、Zhao Song、Haiqing Sun、Zechao Zhuang、Yuming Yang、Jie Xu、Liang Yin、Wenchao Zhou 和 Sheng Wang。2022年。Cloud-Jump：针对云存储优化云数据库。*VLDB*基金会会刊（*PVLDB*）15，12（2022），3432–3444。[21] Brian F. Cooper、Adam Silberstein、Erwin Tam、Raghu Ramakrishnan 和 Russell Sears。2010年。使用 YCSB 对云服务系统进行基准测试。载于第 1 届 ACM 云计算研讨会论文集（SoCC '10）。ACM，美国纽约州纽约市，143–154。[22] James C. Corbett、Jeffrey Dean、Michael Epstein、Andrew Fikes、Christopher Frost、J. J. Furman、Sanjay Ghemawat、Andrey Gubarev、Christopher Heiser、Peter Hochschild、Wilson Hsieh、Sebastian Kanthak、Eugene Kogan、Hongyi Li、Alexander Lloyd、Sergey Melnik、David Mwaura、David Nagle、Sean Quinlan、Rajesh Rao、Lindsay Rolig、Yasushi Saito、Michal Szymaniak、Christopher Taylor、Ruth Wang 和 Dale Woodford。2013 年。Spanner：谷歌的全球分布式数据库。《ACM计算机系统汇刊》（TOCS '13）31，3（2013），1–22。[23] Gilles Darold。2024年3月30日，2026年。pgBadger。https://pgbadger.darold.net/ 访问日期：3月。[24] Siying Dong、Andrew Kryczka、Yanqin Jin 和 Michael Stumm。2021年。RocksDB：面向大规模应用的键值存储的开发重点演进。《ACM存储汇刊》（TOS）17，4（2021），1–32。[25] eBPF基金会。2024年。eBPF——介绍、教程与社区资源。https://ebpf.io 访问日期：2026年3月30日。[26] Sebastien Godard。2024年。iostat(1)——Linux 手册页。https://linux.die.net/man/1/ iostat 访问日期：2026年3月30日。[27] 谷歌。2024年。PostgreSQL版 AlloyDB。https://cloud.google.com/products/ alloydb 访问日期：2026年3月30日。[28] 谷歌。2024年。PostgreSQL版 AlloyDB 发布说明。https://cloud.google.com/ alloydb/docs/release-notes 访问日期：2026年3月30日。[29] 谷歌。2024年3月30日，2026年。Cloud SQL。https://cloud.google.com/sql 访问日期：3月。[30] 谷歌。2024年。Google Cloud 价格计算器。https://cloud.google.com/ products/calculator?hl=en 访问日期：2026年3月30日。[31] 谷歌、埃森哲。2024年。AlloyDB：企业工作负载的一个选择? https://www.accenture.com/content/dam/accenture/final/accenture-com/document-2/Accenture-AlloyDB-For-Enterprise-Workloads.pdf 访问日期：2026年3月30日。[32] Brendan Gregg。2020年。《系统性能：企业与云》（第 2 版）。Addison-Wesley Professional，波士顿，马萨诸塞州。[33] Ajay Gulati、Arif Merchant 和 Peter Varman。2007年。d-clock：异构资源环境中的分布式服务质量。载于 ACM分布式计算原理研讨会论文集（PODC '07）。美国俄勒冈州波特兰，330–331。[34] Ajay Gulati、Arif Merchant 和 Peter J. Varman。2010年。mClock：处理虚拟机监控程序 I/O 调度中的吞吐量变化。载于 USENIX 操作系统设计与实现会议论文集（OSDI '10）。USENIX协会，加拿大不列颠哥伦比亚省温哥华，437–450。[35] HammerDB。2024年。HammerDB。https://www.hammerdb.com/ 访问日期：2026年3月30日。[36] HashiCorp。2024年。HashiCorp 的 Terraform。https://www.terraform.io/ 访问日期：2026年3月30日。[37] Dongxu Huang、Qi Liu、Qiu Cui、Zhuhe Fang、Xiaoyu Ma、Fei Xu、Li Shen、Liu Tang、Yuxing Zhou、Menglong Huang、Wan Wei、Cong Liu、Jian Zhang、Jianjun Li、Xuelian Wu、Lingyu Song、Ruoxi Sun、Shuaipeng Yu、Lei Zhao、Nicholas Cameron、Liquan Pei 和 Xin Tang。2020年。TiDB：一种基于 Raft 的 HTAP数据库。《VLDB基金会会刊》（PVLDB）13，12（2020），3072–3084。[38] IO Visor项目。2024年3月30日，2026年。bcc。https://github.com/iovisor/bcc 访问日期：3月。[39] Jongseok Kim、Chanu Yu 和 Euiseong Seo。2023年。重振缓冲 I/O：优化页面回收和 I/O 限速。载于 2023 年 IEEE 第 41 届国际计算机设计会议论文集（ICCD）。IEEE，美国华盛顿哥伦比亚特区，475–482。[40] Leslie Lamport。1998年。兼职议会。《ACM计算机系统汇刊》16，2（1998），133–169。[41] M.D. McIlroy、E.N. Pinson 和 B.A. Tague。1978 年。UNIX 分时系统：前言。《贝尔系统技术期刊》 57，6 （ 1978 ），1899–1904。第 2 部分。[42] C. Mohan、Don Haderle、Bruce Lindsay、Hamid Pirahesh 和 Peter Schwarz。1992年。ARIES：一种支持细粒度锁定和部分回滚的事务恢复方法，采用预写日志。《ACM事务数据库系统》17，1（1992），94–162。[43] Maryam Mozaffari、Anton Dignös、Johann Gamper 和 Uta Störl。2024年。自调优数据库系统：自动数据库模式设计与调优的系统性文献综述。《ACM计算调查》56，11（2024），277:1–277:37。[44] Neon。2024年3月30日，2026年。Neon无服务器 Postgres。https://neon.tech 访问日期：3月。[45] Diego Ongaro 和 John Ousterhout。2014年。探寻易于理解的共识算法。载于 USENIX年度技术会议论文集（USENIX ATC '14）。USENIX协会，费城，宾夕法尼亚州，305–320。[46] 甲骨文。2024年。MySQL。https://www.mysql.com/ 访问日期：2026年3月30日。[47] 甲骨文。2024年。Oracle自治事务处理。https://www.oracle.com/autonomous-database/autonomous-transaction-processing/ 访问日期：2026年3月30日。[48] 甲骨文。2024年。OTN开发与分发许可条款。https://www.oracle.com/downloads/licenses/instant-client-lic.html 访问日期：2026年3月30日。

[49] OSSC-DB。2024年。pg_statsinfo。https://github.com/ossc-db/pg_statsinfo 访问日期：2026年3月30日。[50] Xi Pang 和 Jianguo Wang。2024年。理解存算分离数据库设计原则对性能的影响。载于 *ACM*国际数据管理会议论文集（*SIGMOD '24*）。ACM，美国纽约州纽约市，180:1–180:26。[51] Hojin Park、Gregory R. Ganger 和 George Amvrosiadis。2020年。以更少资源获得更多 IOPS：利用公有云中的突发型存储。载于 第 *12* 届 *USENIX*云计算热点研讨会（*HotCloud '20*）。USENIX协会，线上。[52] PostgreSQL全球开发组。2024年。PostgreSQL。https://www. postgresql.org/ 访问日期：2026年3月30日。[53] Pulumi。2024年。Pulumi——基础设施即代码。https://www.pulumi.com/ 访问日期：2026年3月30日。[54] Yingjin Qian、Marc-André Vef、Patrick Farrell、Andreas Dilger、Xi Li、Shuichi Ihara、Yinjin Fu、Wei Xue 和 André Brinkmann。2024年。在分布式文件系统中结合缓冲 I/O 与直接 I/O。载于 *USENIX*文件与存储技术会议论文集（*FAST '24*）。USENIX协会，加利福尼亚州圣克拉拉，17–34。[55] 红帽。2024年。红帽 Ansible。https://www.ansible.com/ 访问日期：2026年3月30日。[56] Schumacher Robin 和 Chibber Harshita。2025年。市场份额：*2024*年全球数据库管理系统。技术报告。Gartner公司。文档编号：G00806029。https://www.gartner.com/en/documents/6494271。访问日期：2026年3月30日。[57] Rebecca Taft、Irfan Sharif、Andrei Matei、Nathan VanBenschoten、Jordan Lewis、Tobias Grieger、Kai Niemi、Andy Woods、Anne Birzin、Raphael Poss、Paul Bardea、Amruta Ranade、Ben Darnell、Bram Gruneir、Justin Jaffray、Lucy Zhang 和 Peter Mattis。2020年。CockroachDB：具有弹性的地理分布式 SQL 数据库。载于 *ACM*国际数据管理会议论文集（*SIGMOD '20*）。ACM，美国纽约州纽约市，1493–1509。[58] Linus Torvalds。2025年。torvalds/linux。https://github.com/torvalds/linux 访问日期：2026年3月30日。[59] 事务处理性能委员会。2024年。TPC-C 主页。https: //www.tpc.org/tpcc/ 访问日期：2026年3月30日。[60] Alexandre Verbitski、Anurag Gupta、Debanjan Saha、Murali Brahmadesam、Kamal Gupta、Raman Mittal、Sailesh Krishnamurthy、Sandor Maurice、Tengiz Kharatishvili 和 Xiaofeng Bao。2017年。Amazon Aurora：高吞吐量云原生关系数据库的设计考量。载于*ACM*国际数据管理会议论文集（*SIGMOD '17*）。ACM，美国纽约州纽约市，第1041–1052页。[61] Andy Warfield。2023年。Amazon S3。载于第*21*届*USENIX*文件与存储技术会议（*FAST '23*）。USENIX协会，加利福尼亚州圣克拉拉。地点：加利福尼亚州圣克拉拉。出版方：USENIX协会。[62] Sage A. Weil、Scott A. Brandt、Ethan L. Miller、Darrell D. E. Long 和 Carlos Maltzahn。2006年。Ceph：一种可扩展的高性能分布式文件系统。载于操作系统设计与实现会议论文集（*OSDI '06*）。USENIX协会，美国华盛顿州西雅图，第307–320页。[63] Fengguang Wu。2009 年。writeback：改用每个BDI的线程来刷新数据 · torvalds/linux@03ba378。https://github.com/torvalds/linux/commit/03ba3782e8dcc5b0e1efe440d33084f066e38cae 访问日期：2026年3月30日。[64] Fengguang Wu。2012年。Fengguang, W. 无 IO 脏页限流。https://events.static.linuxfound.org/images/stories/pdf/lcjp 2012_ 30, wu.pdf 访问 2026 日期：3月。[65] Xinjun Yang、Yingqiang Zhang、Hao Chen、Feifei Li、Bo Wang、Jing Fang、Chuan Sun 和 Yuhui Wang。2024年。PolarDB-MP：基于解耦共享内存的多主云原生数据库。载于*ACM*国际数据管理会议论文集（*SIGMOD '24*）。ACM，美国纽约州纽约市，第295–308页。[66] Zhenkun Yang、Chuanhui Yang、Fusheng Han、Mingqiang Zhuang、Bing Yang、Zhifeng Yang、Xiaojun Cheng、Yuzhong Zhao、Wenhui Shi、Huafeng Xi、Huang Yu、Bin Liu、Yi Pan、Boxue Yin、Junquan Chen 和 Quanqing Xu。2022年。OceanBase：每秒处理7.07亿次事务的分布式关系数据库系统。*VLDB*基金会会刊（*PVLDB*）15，12（2022），3385–3397。[67] Yugabyte。2024年。面向现代应用的分布式PostgreSQL。https://www.yugabyte.com/ 访问日期：2026年3月30日。