异步 I/O 在数据库引擎中的实现

杨岢瑞

Aug 16, 2026

磁盘和网络的 I/O 延迟常常成为数据库引擎的首要瓶颈。一张随机读的 16 KiB 数据页，在 NVMe SSD 上也要花费几十到上百微秒；如果让工作线程在每次 I/O 时都阻塞在 `pread` 或 `fsync` 上，上下文切换与线程栈占用会迅速耗尽 CPU 与内存资源，导致并发连接数受限。异步 I/O 通过让 I/O 请求与执行流程解耦，使工作线程可以在等待期间继续处理其他请求，从而提高整体吞吐并降低 P99 延迟。
本文聚焦单机 OLTP 引擎，讨论如何在 MySQL InnoDB、PostgreSQL 等系统中引入异步 I/O。读者需要具备操作系统的 I/O 模型和数据库存储结构基础。

1　操作系统层 I/O 模型

同步阻塞 I/O 要求调用线程一直等待内核完成数据搬运；非阻塞 I/O 虽然立即返回，但需要应用程序反复轮询才能得知完成状态。I/O 多路复用通过 `epoll` 或 `kqueue` 把多个文件描述符集中管理，只在事件就绪时才通知用户态，从而把等待时间从线程级降低到事件级。Linux AIO 进一步把 I/O 请求本身也异步化：`libaio` 早期版本通过内核线程池实现，而 `io_uring` 则采用共享的提交队列与完成队列，配合零拷贝与链式操作，极大减少了系统调用与内存拷贝。
跨平台选型时，Windows 的 IOCP 与 macOS 的 `kqueue` 在语义上与 Linux 接近，但接口细节不同。数据库引擎通常会封装一层抽象，以便在不同内核间切换。

2　数据库引擎的 I/O 需求

在 OLTP 引擎中，最频繁的 I/O 类型是数据页的随机读写、WAL/Redo Log 的顺序追加写，以及后台的 DoubleWrite Buffer 与 Change Buffer 刷盘。读请求往往随机且延迟敏感，而写请求多为顺序且吞吐敏感。传统方案依赖页缓存和 `fsync` 保证持久性，但在高并发下，`fsync` 成为全局锁，阻塞后续提交。

3　异步 I/O 实现架构

从 SQL 层到存储介质，I/O 请求会依次经过执行引擎、存储引擎接口、缓冲池和 I/O 调度层。I/O 调度层负责合并相邻扇区、按优先级排序，并把请求提交给 `io_uring` 或 `libaio`。完成事件通过回调或协程恢复机制返回到上层，错误则进入重试队列。

4 关键技术细节

当 Buffer Pool 未命中时，存储引擎会构造一个异步读请求，携带页号、目标内存地址和回调函数。请求被放入 io_uring 的提交队列后，工作线程立即切换到其他任务；待内核完成 DMA，完成队列产生 CQE，回调函数负责校验和检查并把页面插入 LRU 列表。
DoubleWrite 和 WAL 的异步写则需要保证落盘顺序。引擎会把多个日志条目打包成一个 batch，通过 io_uring 的链式操作一次性提交，并在回调里推进 group commit 的 LSN。Checkpoint 线程在后台异步刷脏时，会动态调节水位，避免前台查询线程因 Buffer Pool 满而阻塞。
零拷贝体现在 io_uring 的 SPLICE 与 FIXED_FILE 选项：数据从磁盘 DMA 直达用户缓冲区，无需内核—用户空间的额外拷贝。

5 并发模型

1:1 线程模型在高并发时会遇到线程数爆炸与上下文切换开销。协程模型在用户态保存寄存器与栈，实现轻量切换；Rust 的 async/await 或 C++20 协程都能把 I/O 等待点编译成状态机。PostgreSQL 的 WaitEventSet 则是经典的事件驱动状态机，每个后端进程通过 epoll 监听多个文件描述符，状态在事件到来时迁移。
混合模型把前台查询交给协程，后台刷盘交给专用线程池，既保留了协程的低开销，又避免了刷盘任务抢占用户态调度器。

6 工程实践与踩坑

MySQL 8.0.33 引入 innodb_io_uring 选项，需要内核 5.10 以上并开启 CONFIG_IO_URING。开启后，随机读场景 QPS 提升约 35%，P99 延迟下降 25%。PostgreSQL 的 io_uring 补丁目前仍在社区迭代，兼容性回退逻辑会检查内核版本并降级到 libaio 或 posix_fadvise。
监控指标包括 io_uring 的 inflight 深度、平均等待时间，以及错误码分布。稳定性方面，需要注意内存屏障正确性、文件描述符上限和写放大问题。

7 性能评测与案例

测试环境采用 NVMe SSD、64 核 CPU 和 256 GiB 内存。基准工具为 sysbench 的 oltp_read_write 脚本。在 1024 并发下，异步 I/O 方案的磁盘利用率从 85% 降到 60%，I/O 带宽提升 40%。写密集的 TPC-C 场景中，group commit 的收益更为显著，提交延迟中位数下降 50%。

8 未来演进

计算存储分离后，RDMA 与 SPDK 将把 I/O 路径进一步下沉到用户态；ZNS SSD 与 CXL 内存带来新的 I/O 调度问题；AI 辅助调度可根据历史负载预测最佳合并与优先级策略。
异步 I/O 是现代数据库引擎性能的“新基建”，需要操作系统、硬件与数据库三层协同设计。落地时必须兼顾兼容性、监控与可运维性。

9 附录

9.1 参考资料

1. io_uring 官方文档与内核源码
2. MySQL 8.0.33 Release Notes
3. PostgreSQL aio 补丁讨论邮件列表

9.2 内核参数示例

```
# 允许的最大异步 I/O 请求数
echo 1048576 > /proc/sys/fs/aio-max-nr
# 脏页回写比例
sysctl -w vm.dirty_ratio=10
```

9.3 io_uring 最小读请求示例

```
struct io_uring ring;
io_uring_queue_init(64, &ring, 0);

/* 准备读请求 */
struct io_uring_sqe *sqe = io_uring_get_sqe(&ring);
io_uring_prep_read(sqe, fd, buf, 16384, offset);
sqe->user_data = (uint64_t)my_read_ctx;

/* 提交到内核 */
io_uring_submit(&ring);

/* 等待完成事件 */
struct io_uring_cqe *cqe;
io_uring_wait_cqe(&ring, &cqe);
/* cqe->res 为返回的字节数或错误码 */
io_uring_cqe_seen(&ring, cqe);
```

上述代码首先通过 io_uring_queue_init 创建深度为 64 的队列；io_uring_prep_read 把读请求填入 SQE，指定文件描述符、目标缓冲区与偏移量；io_uring_submit 一次性把 SQE 提交给内核；随后 io_uring_wait_cqe 阻塞直到 CQE 就绪，检查 cqe→res 可知是否成功，最后调用 io_uring_cqe_seen 告知内核该 CQE 已消费。