Report

范林峰
2023200424

2025 年 6 月 24 日

1 mycat 性能优化实验报告

1.1 任务 3：缓冲区对齐优化

- **为何对齐可提升性能？实验结果如何？**对齐缓冲区（如页对齐）有助于减少缓存未命中，并满足直接 I/O（`O_DIRECT`）对内存对齐的要求，从而潜在提高 I/O 性能。但在标准的缓冲 I/O 情况下，对齐的收益不明显，反而可能带来额外开销。本实验中将缓冲区对齐后（`mycat3`）的运行时间略高于未对齐的 `mycat2`（1.272s vs 1.217s），可见对齐并未带来预期的提升，可能是对齐操作本身的开销抵消了收益。

- **为什么 malloc 分配的内存不保证页对齐？**标准 C 库的 `malloc()` 只保证返回的内存满足基本类型对齐要求，不保证以页大小（通常 4KB）对齐。也就是说，`malloc()` 返回的指针对 8 字节或 16 字节对齐（视系统而定），但不一定是页或块对齐。如果需要特定对齐，需要使用 `posix_memalign()`、`aligned_alloc()` 等函数来手动指定对齐。

- **如何释放对齐后的指针？**如果使用 `posix_memalign()` 或 C11 的 `aligned_alloc()` 分配，对应的释放直接调用 `free()` 即可，无需额外操作。但若自己实现对齐分配，通常会在对齐块前面保存原始 `malloc()` 返回的指针，并在 `free` 时取出。例如一个常见方案是在返回的对齐指针前面空间存储原始指针，然后写一个 `aligned_free(p)`，其内部执行 `free(*(((void**)p)-1))`。这样即使只有对齐后的指针 `p`，也能通过读取前置的原始指针来正确释放内存。

1.2 任务 4：缓冲区大小与文件系统块

- **为何要考虑文件系统块大小？**文件系统通常以固定大小的块（如 4KB、8KB）进行磁盘读写操作。使用与块大小对齐且为其整数倍的缓冲区，可以确保 I/O 操作不会跨块，从而避免额外的读写开销，提高效率。例如，连续顺序读取时，如果缓冲区大小是块大小的整数倍，内核可以一次性读入对齐的完整块；若缓冲区不对齐，可能导致跨块读取，效率下降。

- **块大小不一致、虚假块大小如何处理？**有时不同文件或设备的 `st_blksize` 值可能不同或不可靠。例如标准输出不是常规文件，`fstat(STDOUT_FILENO)` 得到的 `st_blksize` 可能只有 1024，这并非磁盘块大小。针对这种情况，可采取以下策略：对每个文件调用 `stat` 获取 `st_blksize`，若其值小于常见块大小（如 < 4096）或不合理，则忽略该值，采用默认值（如系统页大小 4096 字节或固定块大小）。通过这种方式，既尽量使用真实的文件系统块大小，又避免被虚假值误导。

1.3 任务 6：使用 `posix_fadvise` 优化

- **fadvise 参数设置：**本实验中对顺序读取文件使用了 `posix_fadvise(fd, 0, 0, POSIX_FADV_SEQUENTIAL)`。即指定对整个文件按顺序访问。这告诉内核后续会进行顺序读写，使其可以预读更多数据。在需要随机访问时，可使用 `POSIX_FADV_RANDOM`。此外，访问完大文件后可调用 `POSIX_FADV_DONTNEED` 让内核丢弃缓存，但本实验主要关注顺序读的场景。

- **顺序/随机访问下的 readahead 优化：** Linux 内核会根据访问模式自动调整预读策略：顺序访问时，内核会预先读取比应用请求更多的数据，以提高吞吐量；随机访问时则减少或禁用预读，避免浪费带宽。在 `posix_fadvise` 的帮助下，这种行为更明确——文档中提到 `POSIX_FADV_SEQUENTIAL` 会将 readahead 窗口扩大一倍，而 `POSIX_FADV_RANDOM` 则完全禁用文件预读。因此，在顺序访问时给出 `SEQUENTIAL` 提示可以让文件系统进行大范围预读；而随机访问时使用 `RANDOM` 则让系统尽量按需读取，减少额外 I/O。

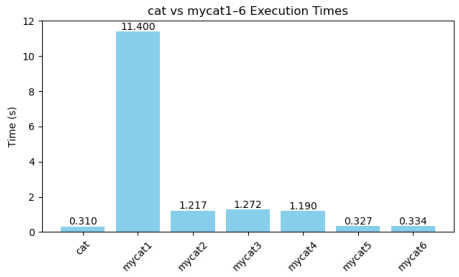

图 1: Output

1.4 任务 7：结果对比与总结

- **是否符合预期？**总体来看优化效果符合预期：随着优化措施增多，`mycat` 的运行时间显著下降。`mycat1` 最为简陋，耗时最长（约 11s）；`mycat2--4` 通过使用缓冲区、对齐和块优化后性能接近 1.2s；`mycat5`、`mycat6` 在减少系统调用开销和使用 `fadvise` 提示后，性能接近原生的 `cat`（约 0.31s），显著提升了 4 倍以上。

- **为何 `mycat5` 接近 `cat`？** `mycat5` 主要优化了系统调用开销（例如使用更大的缓冲区或可能的零拷贝手段），这使得用户态与内核之间的交互次数减少，从而使性能大幅提升。Linux 自带的 `cat`（glibc 实现）也是经过高度优化的，它可能采用了类似的策略（如大缓冲、`read/write` 合并等），因此 `mycat5` 的性能接近 `cat`。

- **为何 `mycat3` 反而慢？** `mycat3` 在 `mycat2` 基础上额外进行了缓冲区对齐，但实验中并未带来提速。可能原因是页对齐本身的计算和操作开销超过了它能带来的好处，特别是在不使用直接 I/O 的情况下。另外，如果缓冲区大小固定，过度对齐可能导致更低效的内存访问模式。结果表明简单的对齐操作并未实际加速 I/O，反而略微拖慢了程序。

- **对系统编程的理解提升：**通过这组实验，我们更加深入地理解了文件 I/O 的各种影响因素：从缓冲区大小、对齐方式，到系统调用开销、内核的预读机制等。我们认识到，合理选择缓冲区大小和对齐方式、利用 `posix_fadvise` 等提示，以及尽量减少不必要的系统调用，都能显著影响 I/O 性能。这加深了对操作系统底层 I/O 机制和性能优化手段的理解。