# 刘罡

(+86) 18707530177 | liugscho@163.com | github.com/Gangliu0036

## 教育背景

**2023.09 ~ 2026.06(预计)** 深圳大学 光电信息工程（硕士）

专业成绩：GPA 3.63 / 4.0

**2019.09 ~ 2023.06** 深圳大学 光电信息科学与工程（本科）

专业成绩：GPA 3.7 / 4.5

语言能力：大学英语六级（CET-6）,普通话二级甲等

获奖情况：研究生学业一等奖学金，第十九届“挑战杯”全国大学生课外学术科技作品竞赛（量子计算赛道）国家二等奖

## 专业技能

- 熟悉 C/C++，熟练使用指针操作及内存管理，掌握 C++面向对象特性（封装/继承/多态），熟悉 STL 常用容器与算法，熟练应用 C++11 特性（智能指针/移动语义等），了解 Go、Python 语言基础语法
- 熟悉常用数据结构（链表/栈/队列/二叉树/哈希表），掌握基础算法（排序/搜索/贪心/动态规划），具备 LeetCode 算法题解决能力
- 熟悉计算机网络体系结构（OSI 七层模型/TCP/IP 四层模型），深入理解 TCP/UDP 协议特性，掌握 TCP 连接管理（三次握手/四次挥手）、流量控制、拥塞控制等机制
- 熟悉 Linux 系统开发环境，掌握网络编程与 I/O 多路复用技术（select/poll/epoll），理解 Reactor 线程池模型实现原理
- 熟练使用 MySQL 数据库，熟悉索引优化（B+树结构）、事务管理（ACID 特性）、存储引擎（InnoDB/MyISAM）、锁机制（表锁/行锁）等核心机制
- 熟悉操作系统核心概念，包括进程通信（管道/消息队列/共享内存）、死锁预防（银行家算法）、内存管理（分页/分段/虚拟内存）等机制
- 具备 Cursor 等智能编程辅助工具使用经验和本地化部署大模型经验。在科研阶段曾基于 llama.cpp 部署本地大模型（支持 4-bit 量化），实现私有化文档问答系统，模型显存占用降低 60%。

## 项目经历

**2025.03 ~ 2025.06** 个人项目 高并发文件服务系统（**C++/libevent**）

项目背景: 设计高性能分布式文件服务系统，支持 PB 级数据存储与 10k+并发连接，解决传统同步日志导致的 I/O 瓶颈问题（从 15ms/op 优化至 2.3ms/op）

核心技术架构:

- 网络层：基于 libevent 实现 Reactor 线程池模型，采用 one loop per thread 架构，通过 epoll ET 模式实现事件驱动（ET 优于 LT 避免重复触发，减少系统调用 30%）
- 异步日志：双缓冲队列+生产者消费者模型实现日志解耦，前端线程写入 A 缓冲，后端线程批量刷新 B 缓冲至磁盘，缓冲交换通过条件变量通知，批量写入将 IOPS 从 6.5k 提升至 18.2k
- 存储优化：实现三级存储架构（Redis 热数据/SSD 温数据/HDD 冷数据），基于 LRU-K 算法（K=2）降低缓存污染，冷数据采用 zlib level 5 压缩节省 37% 存储空间
- 并发控制：元数据哈希表采用细粒度读写锁（shared_mutex）实现无锁读并发，写操作通过独占锁保证线程安全，相比全局锁提升读吞吐 4.2 倍

性能优化实践:

- 零拷贝传输：应用 sendfile 系统调用减少用户态/内核态切换，避免 4 次数据拷贝降低 CPU 占用率 28%
- 内存管理：采用 jemalloc 替换 ptmalloc2，通过 thread cache 和 size class 机制降低内存碎片率 15%，配合对象池模式复用 Buffer 对象
- 日志压缩：实现滚动备份策略（单文件限制 100MB），定期压缩归档历史日志，7 天自动清理降低磁盘占用 82%

技术指标: 2 核 4G 环境下 QPS 达 8.5k（较同步版本提升 42.6%），峰值吞吐 130MB/s，P99 延迟< 5ms，支持 128 线程并发写入日志无数据丢失

**2025.01 ~ 2025.03** 个人项目 分布式 **KV** 存储（**Raft/C++**）

项目背景: 实现强一致性分布式键值数据库，解决 CAP 理论中 CP 场景需求，支持(N/2)-1 节点故障下的高可用性（5 节点集群可容忍 2 节点宕机）

**Raft** 共识算法实现:

- 领导者选举：Follower 心跳超时（150-300ms 随机化避免选票分裂）后转为 Candidate，递增 Term 并行发送 RequestVote RPC，获取多数票（⌈N/2⌉+1）后成为 Leader
- 日志复制：Leader 接收客户端请求后追加本地日志，通过 AppendEntries RPC 并行复制至 Follower，采用流水线批处理（batch size=128）提升吞吐，当多数节点持久化后更新 commitIndex 并 apply 到状态机
- 安全性保证：实现 Log Matching Property（相同 index 和 term 的日志内容相同）、Leader Completeness（已提交日志不会丢失）、State Machine Safety（不同节点相同 index 执行相同命令）

存储引擎设计:

- **SkipListPro** 跳表：实现 4 层索引结构（平衡查询 O(logN)与空间占用），通过 CAS 原子操作实现无锁并发读，写操作采用 mutex 保护关键路径
- 持久化机制：实现 WAL（Write-Ahead Log）保证数据持久性，采用 mmap 映射磁盘文件减少系统调用，定期生成 Snapshot 压缩日志（保留最近 10k 条日志+全量快照）
- 内存优化：跳表节点采用 flexible array member 降低内存碎片，实现 lazy deletion 延迟回收内存，空间利用率达 98%

**RPC** 通信框架:

- 基于 protobuf 序列化（IDL 定义 RequestVote/AppendEntries 消息体），自定义二进制协议（header+body）实现 TCP 流式传输
- 采用 Muduo 网络库实现非阻塞 I/O，通过 epoll+线程池处理并发 RPC，实现连接池复用 TCP 连接降低握手开销

容错与一致性:

- 脑裂处理：通过 Term 机制检测网络分区，旧 Leader 收到更高 Term 拒绝请求并转为 Follower
- 线性一致性：客户端请求携带序列号，Leader 通过(ClientID+SeqNo)去重实现幂等性，成功应用后返回响应
- 测试验证：通过 Jepsen 测试框架注入网络分区/节点崩溃/时钟漂移等故障，验证线性一致性无违反

性能数据: 10 节点集群处理 200k+日志条目，RPC 框架 QPS 达 12k（P99 延迟 1.5ms），支持 10M 级键值对毫秒级查询，自动故障转移时间＜ 500ms

## 自我评价

1. 有较强的信息检索能力，擅长坚决疑难杂症，通过 Google、Github、StackOverflow 等国外论坛/文档解决技术问题.
2. 与时俱进，拥抱 AI，在开发过程中会使用大模型帮助理解难懂错误堆栈和调试思路，提高自身开发效率。
3. 在开发中对待问题具有认真求索的精神，够在短时间内解决问题并理解知识点，严格要求自己遵守编码规范，这使我少写了很多 bug。