面向智能硬件的轻量化BMC重构实践

昆仑太科BMC团队负责人 王亚洲

2026.7

目录

1、xPU管理的瓶颈

xPU管理需求

设备信息管理

- 管理服务器基础信息及XPU卡相关信息

运行状态监控

- 实时监控 XPU 和服务器组件健康与性能

功耗与热管理

- 监控服务器及XPU功耗与散热情况，XPU功耗限制功能

故障检测与预测

- 故障检测与提前预测潜在硬件故障，避免业务中断

负载优化

- 通过识别低利用率设备和优化工作负载分布，确保 AI 集群高效运行

固件升级

- 带外更新XPU卡固件

传统方案的主要问题

协议不兼容

私有协议林立，与Redfish等标准协议脱节，导致不同厂商XPU与BMC难以实现“一次开发、多平台复用”，阻碍生态开放协同。

功能不达标

多数方案功能单一，缺乏全组件资产管理、固件升级及故障诊断等关键能力，无法满足数据中心的精细化管理要求。

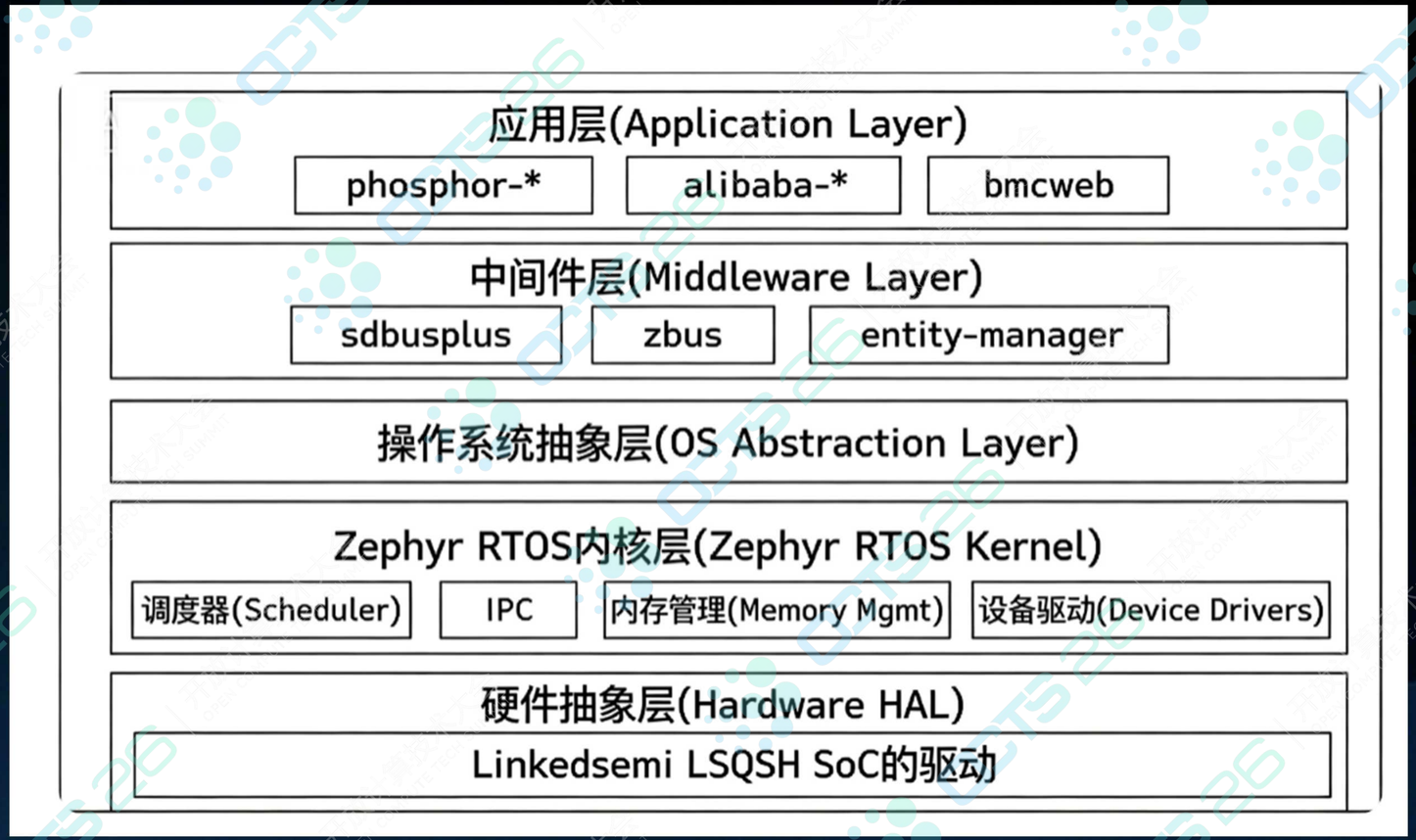
适配效率低

缺乏统一规范导致重复开发，单款适配周期长达数月，产品上市周期长，产业协同成本居高不下，严重拖累运营效率。

2、OpenBMC重构实践

方案一：昆仑轻量级带外管理方案

方案二：zephyr-based OpenBMC



核心技术

OpenBMC 生态严重依赖 Linux 环境下的**sd-bus**库进行 D-Bus 通信，这是系统组件交互的核心基石。要将 OpenBMC 应用无缝迁移到资源受限的 Zephyr RTOS，攻克 sd-bus 的移植难题是必经之路，而这一过程需要直面两大系统在底层架构上的根本性差异。

事件模型

Linux：epoll / socket
Zephyr：k_poll / k_socket

内存模型

Linux：虚拟内存机制
Zephyr：静态/池化内存管理

线程模型

Linux：POSIX pthread 标准
Zephyr：k_thread 内核线程

文件系统

Linux：成熟的 VFS 抽象层
Zephyr：轻量级、极简文件系统

核心依赖

Linux：深度耦合 systemd
Zephyr：无对应组件，需重构

关键优化：D-Bus消息处理延迟优化

核心痛点：高负载下延迟激增

在高并发场景下，D-Bus消息处理延迟从正常的**<5ms**飙升至**>100ms**，引发主机频繁超时，严重影响系统稳定性。

问题根因：多重因素叠加阻塞

经深度剖析，发现事件循环被同步IO阻塞、消息序列化开销过大，且关键服务线程优先级配置不合理，导致任务调度抢占资源不足。

全维度技术优化方案

异步化改造

剥离阻塞操作至独立工作线程，避免主线程事件循环卡顿。

零拷贝传输

大型消息体采用共享内存传递，大幅减少数据拷贝的CPU开销。

优先级调优

提升核心服务线程调度优先级，确保关键消息优先被系统处理。

消息批处理

优化事件分发机制，单次循环批量处理多条消息，提升吞吐效率。

P99延迟降幅：**90%**(150ms → 15ms)

系统吞吐量提升：**300%**(高负载场景下性能飞跃)

关键优化：PSRAM访问瓶颈导致系统卡顿

问题现象：并发访问引发系统异常

多服务并发访问PSRAM时，系统出现周期性卡顿，严重时触发看门狗告警，影响业务稳定性与实时性。

核心根因：资源争用与机制缺失

PSRAM带宽本身有限，多服务无序抢占引发总线严重争用；同时系统缺少DMA支持，大量CPU时间被IO等待占用，导致处理能力下降。

01. 访问串行化管控

引入全局PSRAM访问锁机制，按服务优先级进行排队调度，消除无序并发导致的总线争用冲突。

02. 启用QSPI DMA传输加速

开启DMA通道接管数据搬运，让CPU从繁重的IO操作中解放，专注于核心业务逻辑处理，提升运行效率。

03. 关键服务内存池预分配

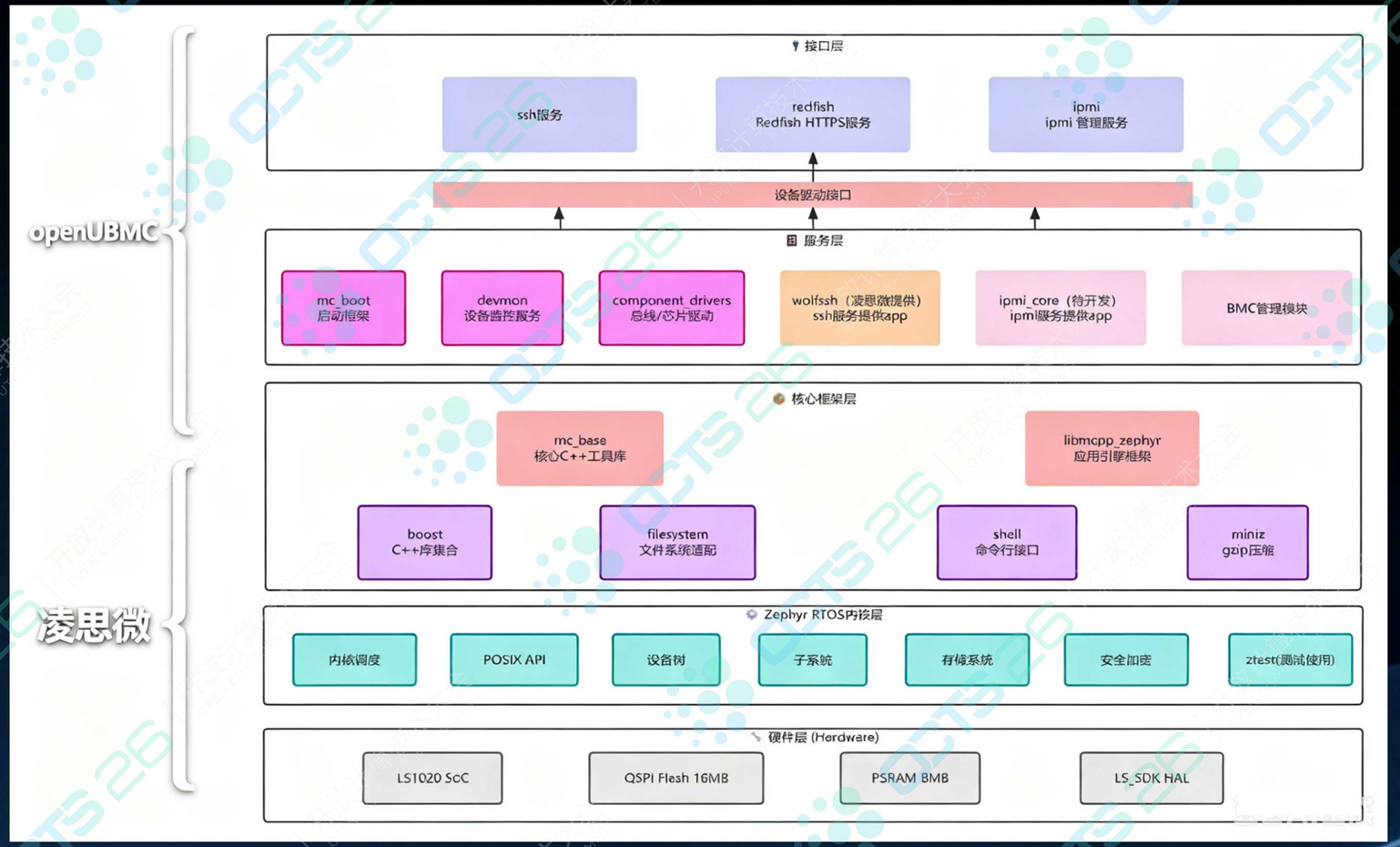
系统初始化时为关键服务预分配固定内存块，避免运行时频繁申请/释放造成的碎片化与额外开销。

04. 日志数据批量异步刷盘

采用环形缓冲区缓存日志与Crashdump数据，积累到阈值后统一批量写入，大幅减少IO操作频次。

3、openUBMC的实践

架构设计


openUBMC
凌思微
接口层
ssh服务
redfish
Redfish HTTPS服务
ipmi
ipmi 管理服务
设备驱动接口
服务层
mc_boot
启动框架
devmon
设备监控服务
component_drivers
总线/芯片驱动
wolfssh（凌思微提供）
ssh服务提供app
ipmi_core（待开发）
ipmi服务提供app
BMC管理模块
核心框架层
mc_base
核心C++工具库
libmcpp_zephyr
应用引擎框架
boost
C++库集合
filesystem
文件系统适配
shell
命令行接口
miniz
gzip压缩
Zephyr RTOS内核层
内核调度
POSIX API
设备树
子系统
存储系统
安全加密
ztest(测试使用)
硬件层 (Hardware)
LS1020 SoC
QSPI Flash 16MB
PSRAM 8MB
LS_SDK HAL

关键技术验证

编译产物与Flash分区布局验证

编译内容：

- 基础能力（网络、littleFS、shell、I2c调试工具等）
- C/C++标准能力（newlib完整库，包含C++异常处理、pthread）
- mcbase+libmcpp 完整内容
- devmon 完整服务
- component_driver I2c和eeprom部分
- redfish、wolfSSL模块、Boost header-only

```
Memory region         Used Size  Region Size  %age Used
          ROM:     3370752 B        12 MB     26.79%
          RAM:      436 KB         1272 KB     34.28%
        PSRAM:         0 B           8 MB      0.00%
        SHMEM:         0 B           4 KB      0.00%
         MBOX:         0 B           4 KB      0.00%
     IDT_LIST:         0 B           2 KB      0.00%
```

flash分区布局（32MB QSPI）

分区	偏移地址	大小	用途说明
Bootloader	0x000000	512KB	引导加载+固件升级支持
XIP代码区	0x080000	~12MB	Zephyr内核+应用就地执行
LittleFS	0xD00000	4MB	/opt挂载点，SR/schema存储
OTA预留	0x1100000	16MB	A/B镜像升级，多版本共存

LittleFS文件系统：与Linux ext4对比

特性维度	LittleFS（Zephyr嵌入式）	ext4（Linux服务器）
断电安全	**COW+日志天然支持**	需journal模式，开销大
最小写入单位	256B块，Flash友好	4KB块，不适合小Flash
动态磨损均衡	**内置支持**，延长Flash寿命	不支持，依赖FTL
适用介质	NOR/NAND Flash/ROM	HDD/SSD/eMMC
文件系统规模	KB~MB级，嵌入式优化	GB~TB级，服务器规模
权限管理	无（嵌入式无需复杂权限）	Unix权限+ACL完整支持
内存占用	**KB级**，适合MCU	MB级，需要充足RAM
目录结构	扁平化设计，快速查找	树形结构，支持海量文件

【技术约束】

- Zephyr RTOS原生不支持ext4文件系统
- ext4依赖Linux内核VFS层，移植到zephyr工作量巨大
- ext4内存占用高（MB级），与880KB SRAM约束不匹配
- ext4适用于HDD/SSD等块设备，对NOR Flash支持不友好

【LittleFS优势】

- Zephyr原生支持：开箱即用，无需额外移植工作
- 嵌入式优化：内存占用KB级，适合MCU场景
- 断电安全：COW（Copy-On-Write）+日志结构，意外断电不损坏数据
- 动态磨损均衡：延长Flash寿命，适合频繁读写场景
- 小块写入：256B块大小，适合小容量NOR Flash

storage_cache机制：解决XIP模式QSPI总线竞争，实现等同于linux的文件系统结构

- 启动时通过memcpy将storage_partition从Flash复制到PSRAM（0x18000000）
- LittleFS直接在PSRAM上运行，零QSPI数据流量，零AHB死锁风险
- PSRAM作为L3缓存层，8MB容量满足文件系统运行需求
- 与openUBMC Linux rootfs目录结构完全一致，验证了编译期预填充机制

devmon服务SYS_INIT自启动验证

devmon在APPLICATION级别按优先级自动初始化，启动后自动从**/opt/bmc/sr**加载SR和schema文件，验证了运行时挂载机制的有效性

核心组件设计

维度	迁移前（原生libmcpp）	迁移后（兼容libmcpp）
IPC机制	D-Bus进程间通信	对象表直调，零IPC序列化
Engine实例	多进程分布式架构	全局单例，统一对象表
方法调用	dbus-send命令行工具	eng调试C API，反射调用
同步原语	std::mutex（pthread依赖）	mc::spinlock（RV32自旋锁）
原子操作	64位native原子操作	RV32 fallback软件实现
定时器	boost::asio::steady_timer	Zephyr k_timer原生定时器

【架构变更：从D-Bus到对象表直调】
迁移前（Linux）：Client → D-Bus Daemon → Engine（3跳，IPC序列化开销）
迁移后（Zephyr）：Client → Engine对象表 → 方法执行（1跳，零IPC）

THANKS

THANKS

THANKS

THANKS