KFNative™ Lattice Fabric 互联算法设计文档

撰写日期：2026-09-13 作者： 周舟

第 1 章 拓扑发现与 Fabric 视图

1.1 Fabric 抽象模型

Lattice Fabric 把物理 PCIe 拓扑抽象为**节点 + 链路**图模型：

- **节点**：GPU 配对单元（GPU + 其配对 SSD 组）、纯 SSD 池节点、主机节点；
- **链路**：PCIe 数据链路（Gen5 x16/x8/x4），带带宽属性（x16 单方向 64 GB/s，协议背景）；
- **管理面**：独立于数据面的带外通道（I2C/SMBus，见原理图管理模块），承载拓扑发现与健康信息。

控制面与数据面分离：**集中式 Fabric Manager** 维护全局视图并下发路由/配对决策，数据面只做按表转发——与 CXL Fabric Manager 的管理思想一致（协议背景），保证转发路径简单可预测。

1.2 拓扑发现协议（设计推算）

发现流程：

1. **链路层枚举**：利用 PCIe 枚举能力（设备/功能树）或板卡固件上报，得到 Switch 端口-端点对应关系；
2. **属性交换**：每节点经管理面广播自身能力（类型、lane 数、SSD 容量、健康状态、可用带宽）；
3. **视图汇聚**：Fabric Manager 汇聚各节点上报，构建全局 $G(V, E)$，节点权重 = 容量/带宽，边权重 = 可用带宽/跳数；
4. **增量更新**：热插拔/故障事件经管理面增量刷新视图，避免全量重枚举。

发现的可靠性：发现消息带序号与重传；对"节点已离线但链路仍 UP"的半失效场景，用周期心跳（设计推算：1 s 心跳、3 次丢失判离线）+ 数据面错误计数双重确认。

1.3 数据结构（C 骨架）

```
struct kfn_node {
    u32 node_id;
    u8  type;                   /* GPU_PAIR / SSD_POOL / HOST */
    u64 cap_bytes;              /* 存储容量 */
    u32 bw_avail;               /* 当前可用带宽（GB/s） */
    u32 health;                 /* 健康分 0-100 */
    u32 peer_gpu;               /* 配对 GPU 节点（若有） */
};

struct kfn_link {
    u32 a, b;                   /* 端节点 */
    u32 bw_total;               /* 链路标称带宽（GB/s） */
    u32 bw_used;
    u32 err_cnt;                /* 数据面错误计数 */
    u8  state;                  /* UP / DEGRADED / DOWN */
};

struct kfn_topology {
    struct kfn_node *nodes;
    struct kfn_link *links;
    u32 nr_nodes, nr_links;
    u64 version;                /* 视图版本，增量更新依据 */
};
```

相邻关系表（**Neighbor Table**）：每 GPU 节点维护其"相邻 SSD/相邻 GPU 配对单元"集合，供 Ring 构造与 KV 交换使用（架构文档四要素之"相邻 SSD 交换"）。

> 本章结论：Fabric 以图模型抽象 PCIe 物理拓扑，采用"带外管理面集中发现 + 数据面按表转发"架构；发现协议含枚举、属性交换、视图汇聚与心跳故障判定，输出全局拓扑与相邻关系表，作为路由与 Ring 构造的输入。

第 2 章 路由/转发算法与 Ring 扩展

2.1 转发模型与路由算法

转发模型：源路由（Source Routing）+ 按表转发。Fabric Manager 计算端到端路径，把路径编码写入发送端描述符（或 Switch 转发表），中间 Switch 不做逐跳决策——路径确定、无拥塞协商，适合 PCIe 这种低延迟确定性网络。

路径计算（设计推算）：

5. 带宽约束最短路：用 Dijkstra/SPFA 计算"跳数最少且满足带宽需求"的路径，边权 = 1 / 可用带宽；

6. **本地性优先**：配对单元内 GPU↔SSD 恒为直接路径（Switch 单跳），不参与全局路由；
7. **权重修正**：链路错误计数升高或健康分下降时，边权惩罚性上调，自动绕开退化链路；
8. **多路径**：对大数据流（权重加载）支持等价多路径分摊（每条路径按可用带宽比例分配段大小，设计推算）。

相邻 SSD 交换：KV_SYNC/训练中转只走相邻节点直连路径，保证跳数 = 1，对应架构"相邻交换"主张。

2.2 环形拓扑构造（Ring Extension）

Ring Extension 是 KFNative 训练的关键扩展（白皮书第 3 章，产品主张 64X fewer GPUs）：

构造算法（设计推算）：

```
输入：参与训练的 GPU 配对单元集合 U，目标环大小 K
输出：环形序列表 ring[]，每节点 ring_prev / ring_next

1. 若 |U| <= K：直接按物理相邻性排序成环（尽量复用已有链路）
2. 否则按"物理相邻 + 链路带宽"分块：块内先成小环，再跨块连接成大环
3. 每节点记录：
     ring_next = 下游（数据发送目标）
     ring_prev = 上游（数据接收来源）
4. 环上每节点同时持有其配对 SSD 作为梯度/中间结果暂存位
```

环的选择依据：

- 环拓扑 lane 利用率高、布线简单（硬件方案 2.3 节）；
- 环上通信天然相邻、可流水化，匹配 All/All Ring Reduce（第 4 章）；
- 对故障容忍：环 + 旁路链路（可选）可在断链时重构，见第 3 章。

2.3 Ring 与配对协同的数据流

- 训练中每 GPU 的梯度写本节点配对 SSD（KV 引擎 PUT）；
- 归约按环流动：`send(grad) → ring_next`，`recv(grad) ← ring_prev`；
- 环上暂存位（配对 SSD）承担中间结果累积，实现"计算-通信-存储"三者重叠（架构文档 3.2 节）。

2.4 路由表下发与一致性（设计推算）

- Fabric Manager 把路径/环表封装为管理消息经带外下发；数据面 Switch 表项版本号与拓扑版本绑定；
- 更新采用"两阶段"：先下发新表 → 确认后切换（避免转发中路径断裂）；
- 不一致检测：数据面完成状态含路径版本，若与发送端不符则上报重路由。

> 本章结论：路由采用"集中计算 + 源路由 + 本地性优先 + 多路径分摊"，确定性低延迟；Ring 扩展按物理相邻性与带宽分块构造环形拓扑，与配对 SSD 暂存协同实现训练数据流；表项更新带版本与两阶段切换保证一致性。

第 3 章 负载均衡与故障切换

3.1 负载均衡策略

负载均衡在三个层面进行（设计推算）：

A. 队列/通道级（板卡内）：固件队列仲裁器按命令优先级与 SSD 队列深度分配：KV 热点 > 权重流 > 后台重建；同一 SSD 的多队列按 NVMe 多队列语义分摊。

B. 路径级（Fabric 内）：

- 等价多路径（ECMP）按流哈希分摊到多条等价链路；

- 大数据流按可用带宽比例分段（第 2 章多路径）；

- 周期采样每链路 `bw_used`，超过阈值（如 80%）触发路径重算或流迁移（设计推算阈值，待实测）。

C. 节点/配对级（容量与热点）：

- Fabric Manager 监控每 SSD 池容量水位与 KV 访问热度；

- 热点 KV 页上浮 DDR5 热层（架构文档 3.4 节），冷数据下沉；

- 容量失衡时调度器迁移配对关系（硬件 lane 可重配），实现 GPU 池/SSD 池动态组合。

目标：在保证"配对本地性"前提下最大化池化利用率——先本地、再跨节点，避免为均衡而破坏数据局部性（与架构 1.2 节四要素一致）。

3.2 故障检测

故障类型	检测手段	判定（设计推算）
链路退化	数据面错误计数（CRC/重传）、AER	错误率超阈值或健康分下降 → DEGRADED
节点离线	带外心跳	3 次心跳丢失 → DOWN
SSD 失效	NVMe 错误上报 + 健康监控	固件上报不可恢复错误 → 摘除
Switch 端口故障	端口状态 + 管理面查询	端口 DOWN → 重路由

故障信息统一上报 Fabric Manager 与 BMC；对不可恢复故障触发"数据面重路由 + 服务层纠删码重建"（架构文档 4.2 节）。

3.3 故障切换流程（设计推算）

检测到链路/节点故障
→ 1. 冻结受影响流（发送端暂停，数据面表项版本不变）
→ 2. 重算路径：绕开故障（本地重路由优先，跳数约束）
→ 3. 两阶段下发新表（先下发后切换）
→ 4. 未完成 DMA 重发（PRP/SGL 描述符可重放）
→ 5. 若为节点失效：触发纠删码重建，重建流预留 ≤10% 带宽
→ 6. 心跳恢复/拓扑版本更新，恢复自动检测

Ring 断链恢复：环上断链时，用旁路链路或"拆环成链"重构环（Ring → 链式 AllReduce 仍可工作，性能下降但不断服）；恢复后重新构造完整环。

3.4 切换时间目标（设计推算）

场景	目标恢复时间	说明
单链路退化（有替代路径）	亚秒级	重路由 + 重发
单节点离线（有纠删码副本）	秒级	重建期间服务降级
Switch 整体故障	分钟级	涉及配对迁移与拓扑重构

本章结论：负载均衡分队列/路径/节点三级，以"本地性优先、池化兜底"为原则；故障检测依赖数据面错误、带外心跳与固件上报；切换走"冻结-重算-两阶段下发-重发-重建"流程，Ring 断链可降级为链式继续服务；恢复时间目标为设计推算，需故障注入实测校核。

第 4 章 All-to-All 归约与时延模型

4.1 All/All Ring Reduce 算法

训练集合通信采用 All/All Ring Reduce（对应白皮书 Ring Extension：所有 GPU 全量归约，产品主张 64X fewer GPUs）：

算法描述（设 P 个 GPU 配对节点组成环，数据切为 P 段）：

```
阶段 1（Reduce-Scatter，P-1 步）：
  for step = 1 .. P-1:
  每节点把第 ((rank - step) mod P) 段发送给 ring_next
  同时从 ring_prev 接收第 ((rank - step - 1) mod P) 段并本地累加
  每步传输量 = 总数据 / P

阶段 2（All-Gather，P-1 步）：
  for step = P-1 .. 1:
  每节点把已归约的第 ((rank - step) mod P) 段发送给 ring_next
  同时从 ring_prev 接收完整段，实现全网广播
```

- **通信量**：每节点发送 `2 × (P-1) × (总数据/P) ≈ 2 × 总数据 × (1 - 1/P)`，随 P 增大接近 2× 总数据——与 Ring AllReduce 经典结论一致（协议算法背景）；
- **流水化**：段传输与本地归约重叠，通信延迟可隐藏；
- **存储协同**：每段临时结果落在本节点配对 SSD（KV 引擎 SYNC 中转），实现"计算-通信-存储"重叠（架构 3.2 节）。

4.2 归约正确性与数值

- 浮点累加顺序：Ring 算法保证各节点归约顺序一致（按环序），避免不可重现结果；
- 可选 Kahan 补偿（设计推算：fp16 梯度场景建议启用）控制累加误差；
- 归约操作支持 SUM/AVG/MIN/MAX 及自定义（按训练框架需求扩展）。

4.3 时延模型

单次数据传输时延（数据面全在 Extended PCIe 内，微秒级路径，架构文档 3.3 节）：

```
T = T_pcie_fixed + T_link + T_switch + T_ssd
  T_pcie_fixed : PCIe 事务固定开销（TLP 处理、完成等待）≈ 1–2 μs（协议量级，设计推算）
  T_link       : 传输时间 = L / B（L 段长，B 链路带宽）
  T_switch     : Switch 转发 ≈ 0.5–1 μs/跳（设计推算）
  T_ssd        : SSD 访问 ≈ 数十 μs（读）+ 写入合并（NVMe 背景，白皮书 5.1.5）
```

对比传统路径（架构文档 3.3 节）：

路径	延迟量级	原因
传统：GPU→PCIe→内存→网卡→网络→…→SSD	毫秒级	网络栈 + 多次拷贝 + 交换机
晶格：GPU↔Switch↔SSD（Extended PCIe）	微秒级	路径缩短 + 协议简化（设计推算）

推理流水线时延示例（GPT-3 175B，白皮书 2.4 节示例，设计推算）：

项	值
单层权重	约 3.6 GB
加载时延（Gen6 x16 约 100 GB/s）	约 35 ms
单层前向计算	约 180 ms
流水线单层周期	max(加载, 计算) ≈ 180 ms（加载被隐藏）

Ring AllReduce 总时延（设计推算）：

```
T_allreduce ≈ 2 × (P−1) × (D/P / B_link + T_fixed) + 归约计算时间
```

其中 D 为总梯度数据量、P 为节点数、B_link 为环链路带宽。可见带宽越大、环径越短，归约越快；配对 SSD 暂存使归约无需等待独立内存拷贝。

4.4 带宽与有效带宽

- **链路带宽**：Gen5 x16 单方向 64 GB/s，Gen6 x16 约 128 GB/s（协议背景）；

- **有效带宽主张**（设计推算）：在 KV-Cache/权重流式场景，通过"路径缩短（免网络栈）+ 加载计算重叠 + 环上存储协同"，GPU 感知有效带宽较传统路径放大数十倍（白皮书第 2 章"数十倍有效带宽"主张的定量支撑方向）；
- **验证方法**：建议以实测打点（per-DMA 时延/吞吐 + 流水线空泡率）建立基准，与模型预测对比校核。

本章结论：All/All Ring Reduce 采用标准 Ring 两阶段（Reduce-Scatter + All-Gather），通信量近 2×D 且可流水化；时延模型把传输拆为 PCIe 固定开销 + 链路传输 + Switch 转发 + SSD 访问，量化说明"微秒级路径"与"加载被计算隐藏"的收益；所有数值为设计推算，需实测校核。