第二节 背景

随着技术的发展，整体式内核（如 Linux）承担了更多的功能和责任，目前，Linux 的调度程序有一个复杂的算法，它使用平衡树，每一个核有独自的任务列表，同时必须在内核之间进行复杂的负载均衡才能保持良好的利用率。随着问题的复杂化，这种启发式方法（heurist）显得不那么灵活。

Linux 内核依靠启发式方法做出重要决策，启发式通常是复杂、及计算密集、且有时不切实际的解决方案的廉价替代方案。启发式的目标是在最短的时间获得足够好的方案而不是获得最好的方案。内核使用的启发式方案是一种针对平均情况的一刀切的方法，使用相同内核版本的服务器会使用同样的启发式方法。通过机器学习，可以专门针对每个服务器的工作负载做出决策来提高性能。

专用加速器正在激增，但是当前软件和系统对及俗气的支持仅限于用户模式程序，加速器附带用户库和内核驱动程序，其接口和实现是专有的，尽管存在许多加速器虚拟化技术，可以为应用程序提供虚拟 GPU，但内核空间应用程序不能使用现有解决方案。

第三节 动机

将 ML 模型添加到系统内核的经验激励我们设计基础设施。设计的一个关键挑战是收集推理所需的特征数据，这可能需要在不同的抽象层、采用不同锁机制的不同模块中查询内核数据结构（在第五节提出一个 API 来应对这种挑战）。同时加速器（例如 GPU）至关重要，单独 CPU 无法满足性能需求。

不幸的是，以往的加速器堆栈通常不会公开内核空间 API，并且通常依赖于内核旁路设计，以前的内核加速系统不能满足性能要求。将加速器公开给内核空间揭示了 OS 和 LML 设置特有的机遇和挑战。此设置特有的主要挑战包括管理内核和用户空间应用程序之间的加速器争用，减少跨用户内核边界的不必要数据移动，以及使内核子系统能够根据性能和准确性盈利能力在 CPU 和加速器之间进行调制。

Contention and Performance Variability.内核 LM 工作可以与用户空间工作争夺加速器设备的访问权限，且没有明确的机制来管理。同时加速必须分摊数据传输成本才能提高性能，这需要对可能与内核的延迟目标不一致的输入进行批处理。

性能关键型用户需要稳定地访问专用硬件，ML 辅助内核和计算首先用户之间共享 GPU 时将由于争用引起性能病态，严重影响服务质量。

Data movement.从内核空间调用用户空间 API，需要将数据编组并从源上下文复制到用户空间进程中，并在完成后将结果和修改后的缓冲区复制回来。这可能会导致跨用户内核边界的冗余数据传

输和不必要的同步，从而造成严重的性能损失。由于不存在将数据传输到加速器的内核级接口，因此必须先将内核级数据缓冲区复制到用户空间，然后再使用 cudaMemcpy 等 API 将其复制到加速器或从加速器复制。内核机制的智能组合允许自动数据编组，并消除跨用户内核边界数据传输的双重缓冲。

不直接使用加速器的接口是因为频繁更改的内部接口和缺乏公开可用的文档使得对加速器软件堆栈的部分进行逆向工程变得不切实际，驱动程序并没有向内核公开必要的高级 API，而更高级的 API 可以提供更好的 ML 支持。

对于设备直接管理争用的问题，基于硬件的解决方案往往不灵活，复杂且不断发展的争用管理策略更容易用软件表达，ML 需要额外的策略支持来处理可变的性能盈利功能。

对于隔离是否收到影响的问题，所有加速器都支持某种类型的地址空间隔离虽然任何将 OS 内核数据卸载到加速器的方法都可能暴露新的侧信道，但我们将对侧信道缓解的调查留给未来的工作。

第四节 LAKE 系统设计

4.1 系统架构

LAKE 的设计基于 Linux 内核，并支持 NVIDIA GPU 和 CUDA 接口。系统架构包括三个核心组件：lakeLib、lakeShm 和 lakeD。

lakeLib：提供 API 接口，将内核空间的调用转发到用户空间。类似于代理模式，将内核空间和用户空间的通信复杂性隐藏起来，使得内核开发者无需直接处理硬件接口的复杂性。

lakeShm：提供共享内存机制，优化内核与用户空间之间的数据传输。共享内存的设计避免了数据的多次拷贝，显著降低了传输开销。

lakeD：用户空间守护进程，负责执行硬件加速任务，并将结果返回给内核。这种设计类似于微内核架构，将复杂的硬件加速任务从内核空间移到用户空间，从而提高了系统的稳定性和安全性。同时，lakeD 的设计使得硬件加速器的使用更加灵活，可以动态调整资源分配。

4.2 动态调整硬件使用

LAKE 允许根据任务的批量大小和性能需求动态选择 CPU 或 GPU 执行。这种设计确保了硬件资源的最优利用，避免了不必要的性能开销。硬件加速并不总是有利的，尤其是在处理小批量任务时。LAKE 通过自定义执行策略（如基于 eBPF 的策略框架）在 CPU 和 GPU 之间动态切换，确保在不同的工作负载下提供最佳性能。

例如，当批量大小超过某个阈值时，LAKE 会选择 GPU 执行，否则选择 CPU 执行。

4.3 争用管理

LAKE 提供了争用管理机制，用于解决内核与用户空间之间的资源争用问题。这种设计确保了内核和用户空间应用之间的资源分配不会导致性能下降。

争用检测：LAKE 通过监控硬件加速器的利用率来检测争用情况。当用户空间应用需要 GPU 资源时，LAKE 会检测到这种压力，并动态调整内核的 GPU 使用。

动态调整：LAKE 通过限制内核对 GPU 资源的使用来避免争用，确保了用户空间应用的性能需求得到满足，同时避免了内核对 GPU 资源的过度使用。

4.4 高级 API

LAKE 支持现有的机器学习库并提供了高级 API，使得内核空间应用可以轻松使用这些库。

LAKE 通过 lakeLib 模块将高级 API 的调用转发到用户空间，由 lakeD 执行，这样内核空间应用可以直接使用复杂的机器学习模型。LAKE 还提供了自动数据序列化功能，将内核空间的数据转换为用户空间库所需的格式，简化了数据处理流程，提高了系统的灵活性和可维护性。

第五节 内核特征注册表

1 研究核心

LAKE 提出了一种**内核内特征注册表（In-Kernel Feature Registry）**，旨在高效管理机器学习模型的推理流程与特征向量捕获，解决传统内核中 ML 集成面临的性能瓶颈、异步特征采集复杂性以及批量推理优化问题。其核心目标包括：

1. **最小化 ML 功能对内核性能的影响**；
2. **支持跨模块、多线程的异步特征捕获**；
3. **简化批量特征向量的推理流程**。

2 具体方法与实现原理

2.1 API 设计与性能优化

2.1.1 数据结构：

1. 特征向量存储在内存的**循环缓冲区**中，格式为 `<numfeatures, kvpair*, ts_begin, ts_end>`，其中 `kvpair` 使用**无锁哈希表**实现键值对映射，避免锁竞争。
2. ML 模型在启动时从文件系统加载至内存，减少推理时的 I/O 开销。

2.1.2 内核内实现：避免用户态-内核态切换开销，确保特征捕获和推理处于关键路径时的高效性。

2.2 特征模式（Schema）

1. 每个注册表关联一个**模式**，定义特征向量的格式：`<feature_key → (size, entries)>`，其中：

   ① `size` 为特征值占用的字节数（如 4 字节整数）；

   ② `entries` 支持历史数据存储（例如 `entries=N` 表示保留最近 N 次采样的数组）。

2. 通过模式自动处理历史数据，简化开发者对时间序列特征的管理（如 I/O 延迟的历史记录）。

2.3 异步特征捕获

1. **跨模块与多线程支持：**

   ① 提供 `begin_fv_capture()`、`capture_feature()` 和 `commit_feature_capture()` 等 API，允许在代码任意位置插入特征捕获逻辑。

   ② 支持增量更新（`capture_feature_incr()`），避免频繁全局状态修改。

2. **时间戳管理：**特征向量记录 `ts_begin`（开始时间）和 `ts_end`（提交时间），便于按时间窗口检索批次。

2.4 批量推理与资源管理

1. **批次控制**：通过 `get_features()` 按时间戳或全量获取批次，`truncate_features()` 清除已处理数据（保留历史依赖的特征）。
2. **加速器调度**：开发者注册回调函数（`register_classifier()`）执行推理，并通过 `register_policy()` 管理加速器资源分配策略。

3 可行性分析

3.1 性能可行性：

1. 内存驻留模型、无锁数据结构和循环缓冲区设计显著降低延迟，适合实时性要求高的场景（如 I/O 调度）。
2. 异步特征捕获避免同步锁竞争，适配多线程并发环境。

3.2 开发便捷性：

1. 模式化特征定义和历史数据支持减少了手动状态管理的复杂性。
2. 案例研究（I/O 延迟预测）验证了 API 的实际效用：通过在 I/O 提交和完成代码点插入特征捕获逻辑，结合批量推理，实现动态决策（如拒绝高延迟 I/O）。

3.3 扩展性：

1. 支持动态模型更新（通过文件系统加载），适应模型迭代需求。
2. 批量管理 API 允许开发者灵活控制推理频率与资源消耗，平衡性能与准确性。

4 案例验证：I/O 延迟预测

场景：在 RAID 等冗余存储系统中，通过预测 I/O 延迟优化吞吐量。

实现：

I/O 提交时：记录起始时间、增加待处理 I/O 计数，提交特征向量；定期触发批量推理，根据结果决策是否重定向 I/O。

I/O 完成时：计算实际延迟、减少待处理 I/O 计数，更新特征向量。

优势：LAKE 的异步 API 简化了跨代码模块的特征采集，无需开发者手动处理并发控制。

LAKE 的特征注册表通过**内核内高效数据结构**、**异步 API 设计**和**灵活的批量管理机制**，为 ML 模型与内核的深度集成提供了可行方案。其设计在性能、开发复杂度与扩展性之间取得平衡，并通过实际案例验证了其在复杂系统（如存储优化）中的实用性。