操作系统设计创新的深度分析及未来展望

In-depth Analysis of Innovations in Operating System Design and Future Prospects

学　　校　浙江大学

学　　院　软件学院

专　　业　人工智能

姓　　名　韩金兴

指导教师　赵新奎

2025 年 10 月 20 日

引言

传统操作系统体系结构（如 Unix/Linux）的核心理念和实现架构已经延续了数十年。然而，近年来计算硬件和应用环境发生了巨大的变化。当代数据中心服务器规模空前增大，常见多处理器/多核系统拥有数百核心、海量内存和存储；同时，云计算的普及带来了大规模集群，要求操作系统管理分布在整个数据中心的资。新的计算范式也不断涌现：一方面，各类异构硬件（如 GPU、TPU、智能网卡、FPGA 等）已成为大型部署中的常见元素，为性能优化带来机会也提出资源管理的新挑战；另一方面，新型应用（特别是大规模机器学习、物联网和大数据分析）对操作系统提出了更高的性能和弹性需求。与此同时，无服务器（Serverless）等新编程模型兴起，用户希望按需使用资源、快速扩缩容，这要求操作系统具备高弹性和快速调度能力。此外，操作系统自身的复杂性在累积，Linux 等通用 OS 经过长期功能增强变得庞大且难以快速适应新需求；诸如系统可观测性和数据溯源（Provenance）等特性在传统 OS 中支持不足。现有操作系统在资源利用效率、扩展性、异构支持、容错能力、安全隔离和可观测性等方面逐渐暴露出局限。这些趋势促使学术界和工业界重新思考操作系统的设计思路，以更好地适应现代计算环境尤其是 AI 时代的需求。

在此背景下，不少创新性操作系统架构被提出。本文将对 LegoOS、DBOS 和 FlexOS 这三项在 OS 领域具代表性的研究成果进行深入分析。一方面，它们分别代表了当前操作系统研究中三个重要方向：硬件架构层面的资源解耦与分布式 OS（LegoOS），软件架构层面的数据中心操作系统重构（DBOS），以及安全机制层面的 OS 模块化与专用化（FlexOS）。另一方面，这三种理念对于支持未来 AI 系统都具有潜在影响：AI 工作负载往往运行在大型分布式环境，利用 GPU 等异构硬件，并强调弹性伸缩和可靠性，因而值得探讨这些新 OS 架构能否更好地满足 AI 时代需求。本文将首先分别介绍 LegoOS、DBOS 和 FlexOS 的技术核心和设计理念，分析其优势、局限及对操作系统发展的贡献。在比较分析部分，将对比三者在体系结构、性能影响、安全机制等方面的差异与共性。随后在未来发展展望部分，结合 AI 系统的发展趋势，探讨操作系统未来可能的演进方向，包括异构资源调度、系统弹性、可观测性增强、安全隔离以及数据库型 OS 模型等，并提出可能的研究路径。

第一章 三大新型操作系统

本章将分别综述 LegoOS、DBOS 和 FlexOS 三篇论文的主要内容，聚焦于每个系统的技术核心和架构设计，阐释其设计理念，并总结其优缺点以及对操作系统领域的创新贡献。

1.1 LegoOS：面向硬件资源解耦的分散式操作系统

LegoOS 提出了一种“资源解耦”背景下的新型操作系统架构，旨在突破传统以单机服务器为单位的 OS 设计。在大型数据中心中，常规做法是每台物理服务器运行一个整体内核管理本机 CPU、内存和 I/O 设备。这种单机单内核模式导致资源利用不均衡、扩展性受限，并且硬件升级或故障往往影响整机。LegoOS 针对这一问题，引入了“分离内核（Splitkernel）”的架构理念。Splitkernel 的基本思想是：当硬件资源按功能类型拆分部署时，操作系统也应随之拆分。具体而言，LegoOS 将传统操作系统的功能模块解耦成多个松耦合的管理单元（监视器，monitor），分别驻留于不同类型的硬件组件上，各自负责本地资源的管理。

LegoOS 将操作系统大体划分为三类监视器：进程监视器（运行在具有处理器的节点上）、内存监视器（运行在专用内存节点上）和存储监视器（运行在存储设备节点上），分别管理对应硬件组件的资源。这些监视器类似“乐高积木”一样按需组合，故名 LegoOS。它们通过高速网络互联，只在需要访问非本地资源时才进行通信，其余时间各自独立运行，这种设计减少了跨组件的耦合与共享状态。LegoOS 不维护跨组件的硬件缓存一致性，所有监视器仅通过显式消息传递协作，而不依赖于传统共享内存或缓存一致协议。这一点充分利用了数据中心高速网络（如 RDMA）的性能优势，避免了在分布式环境中强制保持硬件一致性所带来的开销。

在 LegoOS 中，用户视角下的计算实例被抽象为虚拟节点（vNode），相当于一台由 LegoOS 提供的虚拟服务器。一个 vNode 的资源可以来自多个不同的硬件组件（处理器来自某计算节点、主存来自远程内存节点等），同时一个物理组件上的资源也可划分给多个 vNode 共享。这种抽象使应用无需感知底层资源分离的存在，仍像在单一服务器上运行。为了缓解处理器与远程内存分离带来的性能损失，LegoOS 在计算节点上设计了一层扩展 DRAM 缓存（ExCache）：即利用本地少量 DRAM（如 4 GB）充当处理器最后一级缓存之下的软缓存，用于存放热

点数据。进程监视器负责管理这一缓存并执行传统 OS 的进程调度和系统调用接口支持，而内存监视器则统一管理全局的虚拟地址空间和物理内存分配。LegoOS 采用了两级分布式虚拟内存管理机制：处理器端快速缓存常用地址映射，内存端集中分配和维护地址空间，既保证了访问性能又兼顾了全局内存负载均衡。此外，LegoOS 通过维护远程内存数据副本，实现了内存节点故障时的快速恢复，以提高系统可靠性。

LegoOS 的架构打破了服务器为单位的资源孤岛，实现资源池化与动态组合。各类资源可根据负载需要灵活调配，内存监视器可以将空闲内存分配给任何需要的 vNode，从而显著提高整体内存利用率，缓解单机内存不足或 CPU 空闲的问题。其次，硬件组件的弹性扩展和独立演进成为可能：新增内存节点或存储节点可无缝纳入系统，而不必受制于现有服务器配置；某个组件类型的硬件升级也无需更换整机。同时，LegoOS 将故障域缩小到组件级别，提高了容错性：单个内存节点故障只影响其管理的那部分内存，由于计算与存储分离，其它部分可以继续运行或迅速接管，避免“一机宕机，全机停机”的情况。此外，LegoOS 为支持资源解耦进行了专门优化，如引入 ExCache 和两级内存管理，使得远程资源访问的性能损耗降至较低水平。论文的评测表明，在应用工作集的一部分（25%）作为缓存驻留计算节点的情况下，LegoOS 运行典型基准程序的性能仅比传统单机 Linux 慢约 1.3–1.7 倍。相较于同等内存受限情况下使用本地 SSD 或网络交换的 Linux，LegoOS 的性能反而高出 0.8–3.2 倍。这说明通过架构和机制创新，LegoOS 在性能上接近传统 OS 的同时，换来了更好的资源利用和可靠性。LegoOS 也清晰地分离了 OS 各子系统的职责，进程、内存、存储监视器之间共享状态极少且接口简洁。这种模块化设计降低了 OS 的复杂度和耦合度，各子系统的实现可以相对独立，甚至针对不同硬件进行定制，体现了一定的可定制性和异构兼容能力。

尽管具备上述优点，LegoOS 也面临一些局限和挑战。网络延迟开销是资源解耦架构不可避免的问题。处理器对内存的访问需要经过网络通信，即使有 ExCache 缓冲仍难以达到本地内存的延迟和带宽。对于内存密集型或需要频繁同步的工作负载，LegoOS 性能可能显著下降。此外，LegoOS 放弃了跨组件的硬件缓存一致性，这虽然简化了设计但也意味着需要由软件保证数据一致性或避免共

享内存模型，这对应用透明度提出了要求。当前 LegoOS 通过将共享数据放在存储服务来规避一致性问题，但这可能并不适合所有应用场景。第三，LegoOS 依赖高速互连网络才能发挥优势，在一般网络条件下性能会受影响。因此实际部署 LegoOS 需要数据中心具备相应网络基础设施支持。这种全新的 OS 架构也缺乏与现有操作系统生态的兼容性细节。尽管论文实现了 Linux ABI 兼容层，允许未修改的 Linux 应用运行在 LegoOS 提供的 vNode 上，但底层架构的改变仍可能带来隐蔽的问题和调试困难。例如，传统应用假定内存访问统一共享，而在 LegoOS 上变成远程内存调用，这对延迟敏感的软件（如数据库系统自身）可能需要特殊优化适配。从开发维护角度看，LegoOS 引入了多个分布式协调点，需要精心设计协议以确保一致性和效率。这些都增加了实现的复杂性和验证难度。

LegoOS 作为首个针对硬件资源解耦架构设计的操作系统，提出了全新的 Splitkernel 操作系统模型，突破了长期以来“单机单内核”的范式，将 OS 功能按硬件功能垂直拆分到不同节点。这一理念为未来数据中心级操作系统提供了新的思路：即操作系统不再局限于管理单台机器，而是直接管理一组分布式硬件资源。LegoOS 也验证了分离 CPU 和内存这种极端架构的可行性，提出了软硬结合的解决方案来保持性能。这为后续硬件架构发展提供了先导经验。在资源利用和容错方面的效果也证明了资源解耦在操作系统层面的价值：通过改善资源打包和降低系统平均故障影响时间，其实验结果显示显著提升了整体效率和可靠性。此外，LegoOS 的实现向后兼容 Linux 接口，让应用无需修改即可运行，这表明新 OS 架构并非高不可攀，可以与现有生态衔接，这对于推动学术理念向实际系统转化具有借鉴意义。作为硬件解耦时代操作系统的一次大胆探索，LegoOS 开创了“分布式多组件操作系统”这一全新类别，其思想很可能影响后续的数据中心 OS 和云基础设施管理系统的设计。

1.2 DBOS：数据库为中心的分布式操作系统

DBOS（Database-Oriented Operating System）由麻省理工、斯坦福等机构的研究者提出，其核心思想是将分布式关系数据库作为操作系统的内核。传统操作系统在单节点上管理资源，而集群调度、分布式存储、网络通信等功能往往由另一个层次的软件实现，两者缺乏统一协调。DBOS 认为，在大规模集群和云环境下，操作系统应当提升抽象层次，以数据管理的方式来统一管理整个系统状态。

由此，DBOS 主张“一切系统状态皆表”（Everything is a table）。

操作系统中涉及到的各种状态信息（进程、任务队列、文件目录、权限、通信消息等等）都被存储在分布式数据库的表格中进行管理。操作系统提供的各项服务则被实现为对这些数据库表的标准 SQL 事务操作。例如，在 DBOS 中发送进程间消息，相当于向消息表执行一条 INSERT 语句，而接收消息则对应对表执行 SELECT 加 DELETE 查询；再如，文件系统的元数据操作被转换为对数据库执行增删改查操作，调度器为调度任务可以查询和更新任务表等。通过这种方式，原本由内核代码管理的复杂逻辑被统一映射为数据库事务，利用 DBMS 的高性能事务引擎来保证原子性、一致性、隔离性和持久性。

DBOS 的系统架构被描述为“四层堆栈”：底层第 1 层是一个微内核或最小 OS，仅负责基本硬件抽象（中断、设备驱动、底层字节移动等）；第 2 层是一个高性能分布式内存数据库（当前实现使用 VoltDB，一种主存事务型数据库）；第 3 层是“OS 服务”实现层，编写为 SQL 存储过程或查询，用于实现调度、IPC、文件系统等操作系统服务逻辑；第 4 层是用户应用本身。这种架构下，数据库管理系统成为 OS 的核心：所有 OS 状态由数据库统一存储和维护，OS 服务以事务形式运行在数据库中。

DBOS 的理念源自这样一种认识：在超大规模系统中，管理操作系统状态本质上是一个“大数据管理”问题，应该交由数据库系统来处理。数据库擅长处理海量数据的存储、查询和一致性维护，将 OS 状态转变为关系数据后，可以方便地对系统运行状况进行分析和查询（即内置可观测性和分析能力)，这正是目前传统 OS 所缺乏的。此外，数据库天然支持高可用（通过复制、日志等机制）和事务一致性，DBOS 将这些能力“一次性”地在数据库层实现，OS 的各项服务不需各自再重复解决容错与并发问题。这有望大幅降低系统复杂度，因为众多 OS 模块可以共用数据库提供的可靠机制，而不必各自实现锁管理、恢复等功能[。

DBOS 采取循序渐进的原型实现策略。“DBOS-straw”（稻草人阶段）使用现有 Linux 作为第 1 层，在其上部署 VoltDB 作为第 2 层，并手工实现了一部分第 3 层功能以验证概念。他们首先把任务调度、文件系统和进程间通信（IPC）三项 OS 服务移植到 DBOS 框架下作为样例。研究者为此设计了若干关系表，例如调度方面使用 Task 表记录任务及其状态，由 SQL 触发器/查询决定任务分配，证明

了用 SQL 可以实现如 FIFO、ShortestJobFirst 等不同策略且性能不俗。文件系统方面，他们将文件元数据和数据块索引存入数据库表，实现了基本的打开、读写和目录操作，并与经典文件系统 ext4 性能对比。IPC 方面，通过消息表来收发信息，提供与 gRPC 等通信框架的对比评估。

实验结果表明，数据库驱动的 OS 服务性能可以接近甚至部分超越传统实现：如简单 FIFO 调度下，DBOS 每秒可调度约 75 万任务，中位调度延迟在 200 微秒量级，优于某些现有调度器；文件写性能与 Linux 相当，读性能略低但可接受；IPC 延迟相比 TCP 稍高，但仍在毫秒内，且可利用数据库特性实现更强可靠性保障（消息不丢失且按序等）。这些初步结果有效回应了外界对性能的质疑：尽管增加了数据库这一层，DBOS 的性能并未不可接受地下降，反而简化了大量代码。据论文统计，一个 DBOS 原型实现的文件系统仅用了数十行 SQL，相比之下传统 Linux 的 ext4 实现有数万行 C 代码。由此可见，借助成熟 DBMS，OS 服务的实现代码量和复杂度都大幅降低。

DBOS 的设计极大地提升了系统全局的可观测性和可管理性。由于所有 OS 状态都在数据库表中，管理员或调度器可以方便地对这些表执行查询，实时了解系统资源使用、任务队列长度、文件分布等。这意味着 OS 本身就提供了统一的状态监视和分析接口（通过 SQL），无需再借助额外的监控工具。相比传统 OS 需要依赖 log、各种统计接口，DBOS 使运维与分析变成数据库查询问题，能够提供比现有系统更强的分析能力。例如，DBOS 可以很容易地执行复杂查询来发现资源瓶颈或异常模式，为自适应优化提供支持。其次，DBOS 通过数据库的事务机制实现了各 OS 服务强一致性和高可用。多个节点共享的系统状态（如分布式文件系统元数据、一组服务器的调度队列）在 DBOS 中天然保持一致，因为所有更新都是 ACID 事务的一部分。一旦数据库启用复制等机制，OS 服务也自动具备了容错和故障恢复能力。比如，DBOS 的消息表可以复制容灾，保证即使节点故障消息也不丢失且可立即接管。这种统一的容错策略避免了传统分布式 OS 中各子系统分别设计恢复协议的麻烦。再次，简化代码和减少系统复杂度是 DBOS 一大卖点。借助高度优化的数据库引擎，开发者在实现 OS 功能时可以更多地关注策略，而将并发控制、持久化这些细节留给数据库去处理。这不仅减少了 bug 出现的可能（因为使用了经过验证的 DBMS 代码），也让新功能开发更加敏捷。

DBOS 架构天然适合大规模和云环境。传统 OS 对多节点协调支持不足，需要依赖额外的软件层；而在 DBOS 中，整个集群的资源就是由一个全局分布式数据库来管理，集群即计算机。这为未来统一的集群操作系统提供了实现路径。特别是在 Serverless 情境下，DBOS 可扮演关键角色：它能根据负载在零到成千上万节点间扩展任务，同时利用数据库记录每一步执行和状态变化，实现按需计费和时间穿梭调试。从安全角度看，DBOS 使细粒度访问控制和审计变得容易：数据库已支持用户/权限管理，可以限定不同应用对系统表的访问，且所有操作都有日志记录，这比传统内核加应用日志的机制更加统一。

尽管概念诱人，DBOS 也存在一些明显的挑战。性能开销与实时性问题：将 OS 调用转换为数据库事务，难免引入额外的延迟。对于某些要求极低延迟的操作（如内存管理中的页表查询，设备中断响应等），通过 SQL 访问数据库可能无法满足实时性要求。虽然研究表明 DBOS 性能“可竞争”，但在微观上它始终比直接在内核中操作多了至少一次事务提交代价。解决这一问题需要高速的内存数据库以及可能的本地缓存机制。其二，底层依赖：DBOS 高度依赖于所选数据库系统的性能和可靠性。一旦数据库自身出现 bug 或崩溃，整个操作系统功能都会受到影响，可谓将鸡蛋放在同一篮子里。当然，设计者也认识到这一点，未来计划定制专用的微内核和存储引擎以提高稳健性。其三，通用性：将 OS 服务抽象为关系模型是否对所有资源管理问都合适？有些 OS 状态可能不易或不高效地表述为关系表。DBOS 目前关注的是较高层次的服务，对于底层操作（比如进程切换）仍需传统 OS 配合，这就造成了双层架构：微内核+数据库。如果微内核与数据库的边界处理不好，可能出现新的复杂性。其四，迁移成本：应用程序需要运行在 DBOS 上，必须使用 DBOS 提供的系统服务接口。目前 DBOS 通过提供兼容 POSIX 文件系统和常见 IPC 接口，努力降低迁移门槛。但实质上，要充分利用 DBOS 优势，应用可能需要更紧密地和数据库交互，比如直接使用 SQL 通知机制等，这对编程模型也是改变。其五，扩展性瓶颈：尽管是分布式数据库，但全局单一的 DB 可能会成为某些工作负载的瓶颈。特别是在超大规模分布式环境，不同区域或分区之间是否需要多数据库协同，以及如何减少跨区域通信，都是需要研究的问题。最后，开发难度也是考虑因素：把操作系统开发变成数据库开发，需要操作系统和数据库两方面专家的合作。目前 DBOS 团队集合了数据库

和系统领域的知名研究者，这种跨界融合本身也具有一定挑战性。

DBOS 重新定义了操作系统的边界，提出“操作系统即数据库”的革命性观点，将 OS 看作管理大规模状态的工具而非仅管理硬件。这种视角转变为今后设计数据中心操作平台提供了新方向。DBOS 也证明了数据库技术在操作系统中的可用性，展示了通过数据库实现调度、IPC、文件系统等的可行方案。这为后续系统研究者提供了宝贵经验：或许可以更多借鉴成熟的数据管理技术来构建操作系统的子系统。在简化 OS 实现——用几十行 SQL 代码实现了传统 OS 几千行代码的功能，这强烈说明高层抽象和领域特定工具可以极大提高软件开发效率，也为操作系统教育和实现提供了新思路。DBOS 将事务、可观测、可恢复等高级特性自然引入 OS，丰富了操作系统应有的功能集合。DBOS 内置“时间旅行”能力，可将系统恢复到过去某一时刻状态，以方便调试和容灾。这种功能在传统 OS 中很难实现，但通过数据库的 MVCC 快照就变得水到渠成。DBOS 已在产学两界引发广泛关注，也促成了产业化探索（如 DBOS Inc.的成立和 DBOS Cloud 平台推出）。这表明其理念具有现实吸引力，可能成为云时代 OS 演进的一条重要路线。DBOS 的出现代表了面向数据和事务的操作系统的崭新尝试，在学术上填补了 OS 与数据库交叉领域的空白，对未来云原生操作系统的设计具有开创意义。

1.3 FlexOS：隔离机制灵活可定制的模块化操作系统

FlexOS 关注的是操作系统的安全隔离和定制优化问题。传统操作系统在设计之初通常固化了一套安全与隔离策略—Linux 等主流 OS 采用内核/用户两级隔离；而一些微内核系统则采取更细粒度的服务隔离，每个服务在独立地址空间通过 IPC 交互。无论哪种，通常在设计时就确定，部署后难以更改。然而，现代应用的需求多种多样，有的追求极致性能容忍较弱隔离，有的则对安全可靠性要求极高希望更强隔离。此外，新硬件不断涌现出新型隔离机制，如 Intel 的 MPK（内存保护键）、ARM 的 TrustZone、安全虚拟化扩展等。一个固定的 OS 架构难以同时适应所有这些场景：过于保守则性能损失，过于开放则安全不足。FlexOS 的理念正是为了解决这一两难，它提出让操作系统的隔离和保护策略也变得灵活可定制。FlexOS 基于 Library OS（LibOS）模型实现，一个应用进程携带一个专用的操作系统实例（通常作为其用户态库）。不同于一般 LibOS 固定运行在单一地址空间，FlexOS 将 LibOS 内部进一步拆分为多个细粒度组件（如调度器、文件

系统、网络栈等模块）。关键是：这些组件可以根据需要选择性地彼此隔离。FlexOS 提供了一套配置语言或编译时标注，允许开发者在构建操作系统时指定哪些组件应隔离于不同的隔间，以及采用哪种隔离机制。可用的隔离方式范围很广，从在同一地址空间内但软件检查内存访问的轻隔离，到使用硬件的页表隔离（不同地址空间）甚至不同特权级/虚拟机等强隔离，都由开发者选择。同时，FlexOS 支持设定组件间的数据共享策略（共享内存、消息传递、副本同步等）和启用附加的软件硬化措施（如内存安全检查）。这一切配置在编译时完成，系统会自动生成对应隔离策略的 OS 版本。

FlexOS 的架构使其覆盖了操作系统设计空间上从单体内核、高性能 unikernel 到多服务隔离微内核的多个点。通过配置，FlexOS 可以生成："所有组件都在同一地址空间、无特权隔离"的构型，类似 unikernel，追求极致性能；也可以生成："每个组件在独立进程，由硬件 MMU 隔离"的构型，类似微内核，强调安全可靠；或者介于两者之间的种种混合（部分组件合并、部分隔离）。为了指导用户在海量组合中做出选择，FlexOS 还提供了自动化探索工具。该工具基于应用的特性和用户指定的性能/安全权重，智能地搜索配置空间，推荐一组满足要求的最优配置。论文中，他们针对 Redis、Nginx、SQLite 等实际应用评估了 FlexOS 生成的大量 OS 配置，总计探索了 80 种不同隔离组合。结果表明，不同配置在安全和性能上的差异显著：完全隔离往往性能下降，而无隔离则安全性低。FlexOS 的探索工具能在众多配置中筛选出满足给定性能预算下最安全的几个，例如对 Redis 应用在允许一定性能损失前提下选出 5 种最安全配置。同时，实验显示在配置相同时（即隔离策略等价的情况下），FlexOS 运行应用的性能与甚至优于对比的现有系统。这说明 FlexOS 框架本身开销很小，灵活性并未牺牲效率。

FlexOS 的主要优点在于安全与性能的可调权衡，传统 OS 很难同时满足不同应用对隔离的要求，而 FlexOS 通过编译期专用化使得按需取舍成为可能。对可信度高且追求速度的组件，可以配置为在同一隔间直接调用，避免昂贵的上下文切换和拷贝；对不信任的第三方模块，则可隔离在受限环境中，即使牺牲些性能也保障系统其余部分安全。这样一来，操作系统为应用量身定制，赋予开发者根据应用场景选择安全-性能平衡点的能力，这是过去只有系统设计者才能决定的事情。FlexOS 基于 LibOS，使得每个应用的 OS 实例彼此独立，不同应用可以有

不同的隔离配置，在多租户环境下实现差异化的安全策略。这比“一刀切”的系统级策略灵活得多。FlexOS 兼容并利用多种新硬件隔离特性，支持比如 Intel MPK 用于进程内隔离、使用虚拟化扩展实现强隔离等。当有新机制出现时，理论上只需在 FlexOS 中添加相应选项即可应用于需要的组件，而不必推翻整个 OS 重构。这样，FlexOS 提供了一个试验平台来快速评估新隔离技术在 OS 中的应用。由于组件模块化，FlexOS 得以精简内核信任计算基（TCB）：通过隔离不可信模块，系统的 TCB 可以缩小到仅包含关键组件，从而提高整体安全可信度。实验结果展示了 FlexOS 的性能竞争力，在 Equivalent 配置下性能不输给 Linux 等系统。甚至某些情况下，由于避免了 Linux 中冗余的隔离边界或历史包袱，FlexOS 表现更优。这说明灵活隔离并没有牺牲效率，相反通过专用化可能提升性能。

FlexOS 也存在一些局限性，由于目前是在 LibOS 上下文中运作，这意味着适用范围主要是单进程/单应用场景，需要在虚拟机或容器中运行一个应用+其 LibOS 实例。它并不直接取代传统通用 OS 在多进程、多用户环境下的作用，而更像是一种面向云端应用的专用 OS 框架。因此在一般桌面或服务器多任务场景中，FlexOS 的思路需要演化才能应用。配置空间复杂性也可能让一般开发者望而生畏。虽然提供了探索工具，但有效地刻画应用对安全和性能的需求仍非易事，而且自动探索也可能面临组合爆炸。如何确保找到的配置确实满足要求、以及如何证明其安全性，都是难题。与此同时，调试和验证难度增加：每种不同隔离组合实质上生成了一个不同的 OS，传统调试工具可能不直接适用，而且必须验证不同配置下隔离策略正确无误。这需要借助形式化方法或大量测试，在实际部署前保证可靠性。组件划分粒度的选择影响深远：过粗会牺牲灵活性，过细则可能导致性能损耗过大。FlexOS 作者需要决定哪些模块可独立隔离，这或许不适合所有可能的 OS 服务划分方式。引入多种隔离机制也意味着在系统实现中要处理更多底层细节：比如使用 MPK 则需特定架构支持和编译器配合，使用 VT-x 则要运行在虚拟机管理程序上等等。

FlexOS 的贡献在于其提出的“可定制隔离”这一新的 OS 设计原则，打破了操作系统安全机制一成不变的传统，推动 OS 朝着“按应用定制”方向发展。这一思想契合了当前软件定义和专用优化的潮流，提供了在安全-性能维度上动态权衡的实用手段。FlexOS 拓展并延续了 LibOS/unikernel 思想，将其从纯粹追求

性能拓展到兼顾安全。以前的 unikernel 主要通过减少隔离来提速，而 FlexOS 证明了也可以在需要时增加隔离来提安全，相当于统一了 unikernel 和微内核两种流派的优点，丰富了 OS 架构多样性。FlexOS 在工程上实现了自动化的 OS 配置生成。通过配置语言与工具支持，它把过去需要修改代码才能改变 OS 结构的事情变成了配置选项，大大降低了试验不同 OS 设计的门槛。这对操作系统研究和教育也很有价值——人们可以用 FlexOS 较方便地生成各种结构的 OS 进行对比实验，FlexOS 的研究凸显了新硬件安全功能在 OS 中的应用前景。它综合利用了 MPK、Intel VT-x 等机制，展示了如何在操作系统内部应用它们提供新的隔离方式。这将鼓励硬件和 OS 协同设计，以提供更多可灵活组合的安全手段。综合而言，FlexOS 开创了操作系统安全机制专用化的新方向，让 OS 能够像配置软件那样灵活调整自身架构以适应应用需求，这是对传统 OS 架构僵化模式的一大突破。

第二章 比较分析

以上分别介绍了 LegoOS、DBOS 和 FlexOS 的设计与特点，本章我将从架构定位、性能、安全等方面对三者进行对比分析，以把握它们在操作系统创新谱系中的异同。

三项研究关注的层面不同，LegoOS 主要面向数据中心硬件架构的变革，聚焦于硬件资源如何在多个节点间组合管理的问题；DBOS 关注的是系统软件栈的重构，试图统一集群层次的 OS 服务和数据管理；FlexOS 则立足于单机操作系统内部，探索 OS 模块划分与隔离策略的弹性调整。可以看出，LegoOS 和 DBOS 都是在分布式/集群背景下提出的，而 FlexOS 回答的是“特定应用的操作系统应该如何定制”。因此三者并不相互排斥，反而可能形成互补：**可以想象在一个 LegoOS 管理的解耦硬件集群上，运行着多个 DBOS 风格的服务框架进行全局资源调度，而每个具体应用服务的实例由 FlexOS 提供定制隔离——三者结合形成未来云操作系统的全貌。**

架构实现方式上，LegoOS 采用的是内核重构的方法——重新编写了一个分布式操作系统内核，向上提供 Linux 兼容接口。它更接近传统 OS 的实现方式，但架构分布式。DBOS 则是 OS 与数据库融合，很大程度上依赖现成的数据库系统提供功能，其 OS 服务逻辑用高级查询语言实现。所以 DBOS 不像一个从零开发的内核，更像一个运行在数据库之上的中间层；事实上，他们短期内仍使用

Linux 内核来跑数据库。FlexOS 可以认为是对存在的操作系统代码进行模块化改造并参数化：它基于 UniKraft 等 LibOS 项目，将代码分割成组件并加入隔离配置选项。因此 FlexOS 不是全新内核，而是对已有 OS 代码库的框架改进。这导致了三者开发难度和成熟度不同：LegoOS 需要实现大量底层细节，工程量大但控制力强；DBOS 借力 VoltDB，快速原型但需要调整数据库来满足 OS 需求；FlexOS 建立在已有组件之上，易于验证概念，但要涵盖所有 Linux 功能仍需长期努力。各自的实现复杂度和可靠性目前都停留在研究原型阶段，没有一个达到像 Linux 那样通用成熟，但 DBOS 由于采用成熟 DBMS，在其范围内可能可靠性更高一些，而 LegoOS 和 FlexOS 则都有大量自研代码需要验证。

在性能方面，三者所关注的指标不同，很难直接比较。LegoOS 牺牲了一定的访问延迟换取资源弹性，论文报告其平均性能损失在 30%－70%范围。DBOS 强调其性能与传统方案“相当”，调度、文件等基本操作可达到同一量级，但在微观基准上有数倍的差距。FlexOS 追求的是在相同隔离级别下不比 Linux 慢，同时提供更佳的安全；从实验看，它在某些配置下甚至略胜 Linux，这是因其剥除了冗余层。如果对比扩展性，LegoOS 和 DBOS 显然针对大规模扩展，LegoOS 通过支持更多资源节点，DBOS 通过分布式 DB 扩展到多机。FlexOS 本身不涉及多机，但可配合 LegoOS/DBOS 使用。

资源开销方面，LegoOS 需要多节点各运行一部分 OS 监视器，整体来看可能增加了一些内存/CPU 开销，但这一点论文未详细量化。DBOS 需要一套 VoltDB 节点，数据库本身消耗内存 CPU 不少，不过考虑到它取代了其他服务（调度器、FS 等），整体开销未必更高。FlexOS 的开销主要来自隔离边界，如多进程隔离会消耗更多内存和上下文切换时间，MPK 隔离增加一些检查指令等。这些开销在 FlexOS 中可以按需调整，没有统一值。当着眼于时延敏感场景（如实时系统），LegoOS 跨网络存取内存的长尾时延可能是最大问题，而 DBOS 需保证数据库事务的尾延迟，FlexOS 则需要确保关键路径上隔离机制的延迟符合要求。所以各自适用的工作负载也不同：LegoOS 适用于吞吐导向、容错优先的场景（如大规模数据处理），DBOS 适用于需要全局协调和分析的场景（如云平台资源管理），FlexOS 适用于对安全要求高且愿意为之优化的单服务场景。

在安全方面，三者均提供了有益改进，但侧重不同。FlexOS 自不必说，其核

心就是灵活实现隔离。传统 OS 一般只有一套安全机制，而 FlexOS 能组合多种隔离手段（进程边界、语言安全、硬件密钥等），使得安全策略可以针对不同模块定制，这是对安全性的巨大增强。相比之下，LegoOS 和 DBOS 没有直接强调安全隔离。LegoOS 由于将各功能分散在不同节点上，反而增加了隔离边界。但 LegoOS 本身没有在软件层细分隔离——每类监视器内部还是单体结构。如果某监视器被攻陷，可能影响同类资源。不过因为 LegoOS 的 vNode 机制，不同应用基本在软件上隔离，只是共享硬件组件。所以可以说 LegoOS 强化了故障和安全隔离的粒度到部件级别，但不涉及进程内隔离新机制。DBOS 在安全上的贡献主要体现在数据完整性和审计方面：所有状态变更都有记录，可追溯来源，符合数据治理需求。此外 DBOS 统一了访问控制，可以更简单地实行安全策略。但是，由于 DBOS 大量功能在用户态实现（SQL 层），传统内核提供的安全（内存隔离等）并未增强。且引入如此多组可能扩大攻击面。不过，DBOS 可以方便地备份和恢复状态，这对故障恢复和攻击后分析是有帮助。

第三章 未来发展展望

面向 AI 应用的迅猛发展和计算环境的演进，操作系统需要顺应并支持若干新的趋势和需求。本章结合本文讨论的 LegoOS、DBOS、FlexOS 的理念，探讨在 AI 时代的背景，操作系统未来可能的发展方向。

操作系统本身的复杂策略越来越多，如何优化调度、预取、缓存大小等可能超出人工调参能力。AI 技术可以用于 OS 自适应优化。早期已有使用强化学习优化 IO 调度或内存管理的工作，而有了 DBOS 提供的丰富运行数据，这种学习将更有效。可以想象未来 OS 配有“OS 大脑”模块，持续分析系统数据，自主调整参数甚至重构某些代码路径（类似 JIT 编译优化）。这将使操作系统更智能地适应各种应用，包括不断变化的 AI 工作负载。

同时，随着生成式 AI 的发展，将来甚至可能出现用自然语言操纵操作系统的场景，如管理员通过对话让 OS 调整资源或检查问题。底层实现需要 OS 提供高层抽象接口，这与 DBOS 的可查询 OS 状态正相吻合。在这种人机接口上，AI 或成为 OS 的一个子系统，为用户请求匹配正确的底层操作。

OS 需要在多个物理隔离的机器之间协调计算而不泄露数据，这涉及安全隔离、网络调度的新组合。也许可以借鉴 LegoOS 的分布式架构，让不同机构的机

器扮演不同资源角色（如一方提供数据内存，一方提供模型计算 CPU），OS 作为中介确保彼此数据不直接暴露但又完成训练，这类跨域协同的操作系统服务将在未来产生需求。

综上，AI 的蓬勃发展既对操作系统提出诸多挑战，又为 OS 领域注入新活力和思路。LegoOS、DBOS、FlexOS 等工作的创新为我们展现了可能的路径：通过解耦与重构提升弹性和扩展，通过数据化与智能化增强管理和优化，通过灵活隔离保障安全和性能。未来操作系统很可能是这些理念的综合体，一个弹性、智能、安全的数据中心操作系统，成为 AI 时代计算基础的坚实支撑。

参考文献

[1] Shan Y, Huang Y, Chen Y, et al. LegoOS: A disseminated, distributed OS for hardware resource disaggregation[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 69-87.

[2] Li Q, Kraft P, Kaffes K, et al. A Progress Report on DBOS: A Database-oriented Operating System[C]//CIDR. 2022.

[3] Lefeuvre H, Bădoiu V A, Jung A, et al. FlexOS: towards flexible OS isolation[C]//Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 467-482.

[4] Baumann A, Barham P, Dagand P E, et al. The multikernel: a new OS architecture for scalable multicore systems[C]//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 2009: 29-44.

[5] Huang Z, Lie D, Tan G, et al. Using safety properties to generate vulnerability patches[C]//2019 IEEE symposium on security and privacy (SP). IEEE, 2019: 539-554.、

[6] Sartakov V A, Vilanova L, Pietzuch P. Cubicleos: A library os with software componentisation for practical isolation[C]//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 546-558.

[7] Kuenzer S, Bădoiu V A, Lefeuvre H, et al. Unikraft: fast, specialized unikernels the easy way[C]//Proceedings of the Sixteenth European Conference on Computer Systems. 2021: 376-394.

[8] Coppock P H, Zhang B, Solomon E H, et al. LithOS: An operating system for efficient machine learning on GPUs[C]//Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. 2025: 1-17.

[9] Gu J, Wu X, Li W, et al. Harmonizing performance and isolation in microkernels with efficient intra-kernel isolation and communication[C]//2020 USENIX Annual Technical Conference (USENIX ATC 20). 2020: 401-417.

[10] Sung M, Olivier P, Lankes S, et al. Intra-unikernel isolation with intel memory protection keys[C]//Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2020: 143-156.

[11] Heiser G, Elphinstone K, Vochteloo J, et al. The mungi single-address-space operating system[J]. Software: Practice and Experience, 1998, 28(9): 901-928.

[12] Vahldiek-Oberwagner A, Elnikety E, Duarte N O, et al. {ERIM}: Secure, efficient in-process isolation with protection keys ({{{{{MPK}}}}})[C]//28th USENIX Security Symposium (USENIX Security 19). 2019: 1221-1238.