Ossim：面向集群规模全栈评估的操作系统驱动实时仿真

Yiliang Wan[1], Jonas Kaufmann[2], Yihan Yang[1], Chaoyi Ruan[1],Haifeng Sun[1*], Antoine Kaufmann[2], Jialin Li[1]

[1]新加坡国立大学 [2]马克斯·普朗克软件系统研究所

arXiv:2611.29173v1 [cs.DC] 22 Nov 2026

摘要

集群规模全栈仿真对于在部署前评估分布式软件栈和新兴硬件组件至关重要。这类仿真必须兼顾两点：对未经修改的生产环境软件栈实现全栈保真度，以及达到迭代配置探索所需的仿真性能。然而，现有方法都无法同时满足这两点。我们提出 Ossim，一种构建于 Linux 虚拟化栈之上的操作系统级集群规模全栈仿真方法。Ossim 由四个子系统组成：面向仿真的调度、实时内存层次结构管理、仿真感知的进程间通信，以及多主机协调。这些机制共同协调共享仿真时间下的真实组件和建模组件，同时控制共置的实时虚拟机之间的干扰。这些机制指向仿真原生操作系统支持，使仿真控制与编排成为操作系统的核心职责。

1 引言

评估分布式系统需要理解其各层之间的交互：应用程序和运行时、操作系统、处理器和内存层次结构、设备以及网络。这些交互决定了部署配置能否提升应用程序性能，以及新兴硬件能力能否带来预期收益。集群规模全栈仿真可在目标规模的硬件 [1, 2, 9, 15, 25, 48, 49] 尚不可用或生产部署尚不切实际之前，支持此类评估。它还有助于为云平台 [40, 51]和大数据框架 [4, 5]等生产工作负载选择部署配置。然而，为了支持迭代探索，模拟器既必须保留生产环境软件栈的行为，也必须达到切实可用的执行速度。

满足这些要求带来了两项挑战。第一项是硬件特性覆盖。生产软件可能依赖特定的处理器指令和特性、架构机制，或网卡硬件卸载等设备功能。要忠实地评估此类软件，模拟器必须实现并精确建模所需的硬件特性或功能。现有模拟器并不一定支持所需的特性，即使它们对系统其他方面提供了详细的模型，禁用不受支持的特性或替换为简化的实现，可能会改变工作负载。

第二个挑战是详细主机仿真的成本。全系统模拟器能够执行未经修改的操作系统，而 SimBricks [32]和 SplitSim [33] 等模块化框架则将主机、设备和网络模拟器组合成端到端系统。然而，细致的 CPU 和内存系统仿真可能占据大部分执行时间，并限制整个组合系统的进度。要准确建模主机性能，不仅需要捕捉指令执行，还要捕捉缓存未命中、一致性流量、内存竞争，以及这些因素与计算的重叠情况。在软件中解析这些交互代价高昂。简化这些因素虽能提高仿真速度，却可能扭曲应用程序执行速率，进而影响其他组件观测到的负载和时序。

实时仿真提供了一种互补的方法，可同时应对这两项挑战。实时仿真直接在物理硬件上执行可用组件。对于实时执行接口所提供的功能，它会复用底层硬件，而无需模拟器实现这些功能。对于处理器和内存，原生执行让物理硬件直接完成指令执行、缓存查找、一致性维护和内存访问，从而避免相应的细粒度仿真工作。当可用硬件能够代表所研究的主机时，这种方法既能保留其原生行为，又能将显式模型留给与目标系统不同的组件或属性。

然而，仅有原生执行并不等同于仿真。虚拟机的进度取决于主机调度和资源可用性，而建模组件则按照自身的时序模型推进。不受控制的时间基准差异会改变客户机计时器、通信时序和分布式协调。同样，使用物理内存虽然保留了主机的内存行为，却不会自动复现不同的目标延迟。因此，组合真实组件和建模组件需要对执行进度、资源性能以及

交互变得可见时的仿真时间。

关键挑战在于，使虚拟化主机和异构建模组件能够在协调的仿真控制下实现可组合性。因此，我们聚焦于一种可组合的基底，其中虚拟机和建模组件共享执行与通信抽象，实时资源

*通讯作者。

可以纳入性能调整，并将这些机制扩展到物理主机。这些控制必须协同工作：改变虚拟 CPU 的有效执行速率，必须与其客户机可见时钟、相对于其他组件的调度以及其外部交互保持一致。

我们的关键洞见是，支持硬件虚拟化的操作系统 [41, 50, 53] 是这些控制机制的天然集成点。主机操作系统涵盖应用程序和客户机内核的执行，使未经修改的软件栈能够参与仿真时间。硬件虚拟化保留了原生执行能力，并可访问已开放的处理器特性；与此同时，主机能够控制虚拟 CPU 调度、客户机可见时钟，以及为客户机内存提供支持的资源。虚拟设备接口则为建模通信和外部组件提供了额外的集成点。协调这些机制，使我们能够在控制完整虚拟化主机的进度与交互的同时，复用原生硬件行为。

我们介绍 Ossim，这是一种面向集群规模全栈实时仿真的操作系统原生底层平台。Ossim 建立在先前实时模拟系统的基础之上 [21, 30, 57], ，并遵循两项设计原则。首先，它将仿真控制集成到操作系统机制中，在保持高效原生执行的同时，强制推进仿真时间、调整资源性能，并传递定时事件。其次，它定义了通用抽象和接口，将通用仿真控制与组件特定实现及性能模型分离。这种分离使真实组件和建模组件、主机支持的资源以及通信模型能够灵活组合，而无需重新设计底层控制机制。我们的设计由四个主要子系统构成：

• 面向仿真的调度 通过统一的虚拟时间抽象来表示实时虚拟 CPU 和建模组件。偏差有界调度协调它们的进度，而协同调度则支持需要协调执行的实时组件。

• 实时内存层次结构管理 保留原生内存访问和一致性，管理其后备资源，并利用硬件计数器测量结果，根据配置的内存延迟差异调整虚拟时间推进。

• 模拟感知通信 提供通用接口，将实时客户机与内置模型或外部模拟器连接起来，并控制消息何时对接收方可见。

• 多主机协调 将进度跟踪和定时通信扩展到多台机器，使仿真能够使用单台主机之外的硬件资源。

我们在 Linux 虚拟化栈中实现了 Ossim，并使用未经修改的分布式服务、数据库和分析框架对其进行评估。在报告的七项端到端指标中，Ossim 得出的结果与相应的物理系统测量相差不超过 9%，而墙钟时间仅增加 2.6-24× 倍。在单独的仿真性能实验中，Ossim 的速度降低了 5.5-25× 倍；相比之下，基于 gem5 TimingSimple 的模块化配置速度降低了 665-1426× 倍，基于 O3 的配置则降低了 3381-9329× 倍。这些结果表明，通过操作系统支持协调原生执行、实时资源与建模交互，无需对整个主机进行细致的软件模拟，也能让全栈模拟切实可行。

2 背景与动机

2.1 集群规模全栈仿真

集群规模全栈仿真能够在目标系统达到相应规模或具备实际部署条件之前，评估分布式软件栈和硬件配置。此类评估必须捕捉各个组件的功能及其跨层交互。例如，将网络处理卸载到 DPU，不仅会改变数据包处理延迟，还会影响分布式数据库可用的 CPU 资源。因此，要评估其端到端收益，就必须捕捉生产软件栈以及处理器、内存、设备和网络的行为。实用的模拟器必须保留这些相关行为，同时运行速度足够快，以支持迭代探索。因此，集群级全栈模拟器必须满足两项要求：

R1. 全栈保真度。 我们将保真度分为两个互补方面：全栈覆盖度和准确性。

• 全栈覆盖度 涉及仿真所表示的功能和交互。在软件方面，这要求在不修改源代码的情况下运行整个软件栈；在硬件方面，覆盖度要求包含该软件所使用的处理器指令、体系结构机制和设备功能。用流量生成器替代端主机，或禁用不受支持的硬件特性，都可能改变被评估的软件路径和跨层行为。

• 准确性 关注所表示的系统在多大程度上忠实于目标系统的行为。执行速率、资源竞争和通信时序必须足够贴近目标系统，才能得出具有代表性的端到端结果。例如，若提供卸载接口，却未体现其对主机 CPU 需求和数据包处理时间的影响，结果就会不准确。

R2. 仿真性能。 模拟器必须足够快速地完成端到端实验，以便迭代探索配置。用户会评估候选硬件、网络或软件配置，检查其行为，并通过重复实验对其进行改进。在集群规模下，每次实验都会在每台模拟服务器上执行完整的生产环境软件栈，因此整个仿真所需的墙钟时间变得至关重要：如果目标执行环境中的一分钟

需要数天才能完成仿真，那么探索多种配置将变得不切实际。

2.2 相关模拟方法

离散事件仿真（DES）。DES 将目标系统建模为一系列按仿真时间处理的事件。gem5 [18] 和 Simics [24]等主机模拟器会执行完整的软件栈，并对处理器、内存系统和设备进行建模；ns-3 [45] 和 OMNeT++ [55] 等网络仿真器则对协议和拓扑结构进行建模。这些工具能够在其支持的配置下提供细致的行为模拟。然而，单独使用特定层的模拟器无法捕捉整个分布式系统中的交互。例如，网络仿真通常使用流量生成器或简化的主机，而非执行生产软件栈。

离散事件仿真还带来了两项实际挑战。首先，功能覆盖范围取决于模拟器的实现，而实现可能会省略某些硬件指令或机制。其次，细致的时序模型执行成本高昂。例如，在 ns-3 中仿真一个包含 1,024 台服务器的胖树 DCTCP 工作负载运行一秒，可能需要十多个小时 [6]。

模块化组合。模块化框架 [32, 33]通过通用接口和仿真时间同步，将独立开发的组件仿真器连接起来。它们能够将主机、设备和网络模型组合成端到端系统，同时保留各自的实现。

组合并不能解决这些组件本身的局限。所需的硬件功能仍必须由模拟器提供，而整体速度取决于最慢的组件仿真器，这通常是端主机仿真。模块化本身并不会自动消除详细 CPU 和内存系统建模的开销。

实时仿真。实时仿真直接在硬件上执行物理上可用的组件。这样，实时仿真便能复用硬件特性，摆脱对软件实现的依赖。处理器原生执行也使硬件能够直接执行指令、进行缓存查找、维护一致性并访问内存；这无需细粒度的主机仿真，即可保留处理器的原生行为。因此，仅需为不可用的组件建立显式模型。

然而，实时组件按照物理调度和资源可用性执行，而建模组件则依据其时序模型推进。此外，可用硬件的资源性能可能与目标硬件不同。为确保仿真准确，实时仿真必须控制执行进度、考虑资源差异，并协调交互何时变得可见。

先前的系统为这一方法奠定了重要基础。基于虚拟机的系统 [21, 57] 展示了如何在受控时间内执行未经修改的软件栈，但其模拟控制仅针对特定场景进行了定制。基于进程和容器的系统和容器的系统 [29, 30, 43]为原生应用程序提供了虚拟时间控制，但将客户操作系统及其与硬件的交互排除在模拟栈之外。其他系统 [36, 43] 探索了加速工作负载的实时执行以及硬件/软件评估，但目标仅限于专用软件栈。

2.3 迈向通用实时仿真基底

模块化组合和实时执行要求一种通用的实时仿真底座，它复用硬件以提供原生功能并实现高效执行，同时提供通用接口来组合异构组件。关键挑战在于，底座必须以统一方式协调实时组件与异构建模组件之间的执行和交互，同时保留原生执行在兼容性和性能方面的优势。

实现这一底座需要具备四种能力。

- *C1*：仿真时间执行控制 必须根据仿真时间而非墙钟时间协调实时执行与建模执行，并以统一方式表示不同执行类型的仿真进度。
- *C2*：实时资源控制 必须防止非预期的主机干扰，并在主机支持的资源所表现出的性能与目标系统不一致时进行调整。
- *C3*：定时跨组件交互 必须提供通用通信接口，并控制组件间交互在何时（仿真时间）变为可见。
- *C4*：多主机协调必须在物理主机之间扩展模拟时间推进和定时通信，同时不改变向仿真暴露的抽象。

这些能力彼此依赖。调整实时处理器的执行速率以考虑目标内存延迟，也必须影响其客户机可见时间、相对于其他组件的进度，以及外部交互的时序。同样，即使待处理消息跨越物理主机边界，也必须与接收方的进度保持协调。因此，核心挑战在于整合执行、资源和交互控制，同时使这些控制所共有的机制独立于组件特定实现和性能模型。

3 Ossim 概述

3.1 操作系统作为通用实时仿真基底

我们的关键观察是，主机操作系统是 §2.3 中所指出的四种能力的天然集成点。它为异构的真实组件和建模组件的执行、物理资源及通信提供统一控制，从而能够提供独立于特定组件的通用仿真抽象。

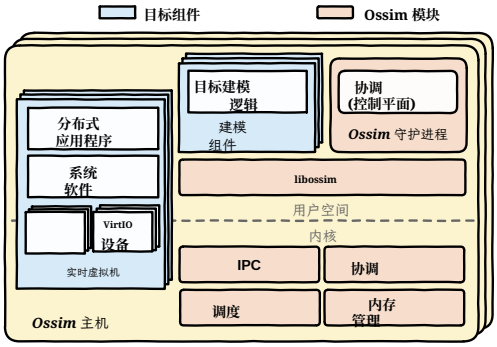


图 1。 Ossim 的总体架构。

模拟时间执行控制。 真实组件和建模组件必须依据统一的模拟进度推进。实时虚拟机在虚拟机监控程序上执行，并受主机操作系统控制；而建模组件通常作为用户空间进程运行，由内核调度。因此，操作系统为协调它们的执行和时间提供了一个统一的控制点。

实时资源控制。 实时组件直接使用 CPU、内存、缓存和设备等物理资源。由于操作系统负责管理这些资源的分配、隔离和访问，因此它自然成为控制主机干扰以及仿真所呈现资源行为的关键位置。

定时跨组件交互。 组件之间的交互必须按照仿真时间变得可见。尽管组件可能使用不同的通信机制，但其交互依赖于操作系统管理的通信和虚拟接口。因此，主机操作系统为控制异构组件之间的交互时序提供了一个共同位置。

多主机协调。 在集群规模下，跨物理主机必须保持相同的执行和交互语义。由于这些语义由每台主机的操作系统强制执行，因此要将其扩展到多台机器，自然需要参与其中的操作系统实例之间进行协调。

这些观察共同表明，主机操作系统可作为通用实时仿真底座的基础：它提供贯穿异构组件、资源、交互和物理主机的通用控制点。

3.2 Ossim 架构

图 1 展示了 Ossim 的架构。我们在 Linux 虚拟化技术栈中实现上述操作系统原生底层平台 [41, 53]。在该技术栈中，硬件虚拟化 [3, 22] 保留了原生执行能力和对所暴露处理器特性的访问，而主机操作系统则控制虚拟 CPU 调度、客户机可见时钟以及为客户机内存提供支持的资源。虚拟设备接口还将实时虚拟机与建模组件连接起来。在此基础上，

表 1. 集群规模全栈仿真的模拟方法比较。

方法 type	系统	Sim. 性能	端主机全栈	集群规模
DES	gem5/Simics	✗	✓	✗
	ns-3/OMNeT++	✓	✗	✓
模块化组合	SimBricks/ SplitSim	*	✓	✓
实时仿真	Phantora	✓	†	✓
	NEX	✓	✗	✗
	Ossim	✓	✓	✓

* De赖于组件仿真器。† 运行未经修改的机器学习应用，且 无需操作系统。

Ossim 将仿真控制集成到操作系统中，同时将通用机制与组件专属实现分离。

Ossim 由四个子系统组成。面向仿真的调度 通过统一的虚拟时间抽象表示实时组件和建模组件，并通过基于域的调度协调它们的进度（§4）。实时内存层次结构管理 保留原生内存访问和一致性，管理其后备资源，并针对目标内存性能差异调整虚拟时间推进（§5）。仿真感知型进程间通信 提供通用接口，用于连接实时客户机与异构网络结构实现，同时控制消息何时对接收方可见（§6）。最后，多主机协调 将上述机制扩展到多台物理机器，使仿真能够使用单台主机之外的资源（§7）。

这些子系统共同构成通用的实时仿真底座，在尽可能保留原生执行的同时，将执行、资源和交互纳入统一的仿真控制之下。表 1 将 Ossim 与现有方法进行了比较。

4 面向仿真的调度

Ossim 分两步在共享的仿真时间下协调实时执行和建模执行（C1）。首先，它将每个参与协调的执行线程表示为一个虚拟任务，并使用统一的虚拟时间（虚拟时间）跟踪其仿真进度（§4.1）。其次，它将虚拟任务分组到域 中，在限制其虚拟时间偏差的同时，兼顾暂时不活动状态和不同的同步要求（§4.2）。

4.1 统一虚拟时间核算

虚拟任务抽象。 虚拟任务是一种用户空间线程，其执行参与仿真时间协调。Ossim 利用这一抽象，将虚拟时间核算与调度逻辑解耦。每个虚拟任务都会向调度器公开一个虚拟时间，而特定于执行方式的核算后端则决定该虚拟时间如何推进。这种分离使调度器能够以统一方式协调异构执行。

Ossim 支持两种核算后端：

基于时钟推导的虚拟时间。 实时虚拟任务（例如 KVM 虚拟 CPU 线程）直接在硬件上执行仿真组件，并根据其硬件执行时间推导虚拟时间。当目标组件的运行速度与仿真主机不同时，可通过配置乘数来缩放这种映射。对于实时虚拟 CPU，Ossim 通过调整 KVM 的半虚拟时钟（pvclock）[54] 来实现基于时钟的核算。原版 KVM 即使在虚拟 CPU 被抢占时，也会根据主机墙钟时间推进客户机时钟，并将被抢占的时段报告为窃取时间；而 Ossim 会在客户机入口处调整时间戳计数器（TSC）[22] 偏移，使时钟仅在客户机执行期间推进。虚拟 CPU 运行时，Ossim 根据相同的时钟状态推算其虚拟时间。客户机退出时，Ossim 会将已执行的客户机时间计入虚拟任务的虚拟时间。因此，客户机时钟与调度器观察到的仿真进度一致。

这种基于时钟的原则同样适用于作为实时组件使用的物理设备。Ossim 不对其内部执行过程进行建模，而是测量其实际执行时间，并将其转换为模拟时间推进或延迟。例如，在实时 DPU 案例研究中，我们测量每个数据包在物理 DPU 上的处理时间，并安排其在相应的虚拟时间延迟后完成（§8.4）。

模型驱动的虚拟时间。 诸如 QEMU 设备模型线程或 ns-3 事件循环线程这样的建模虚拟任务，会根据其建模逻辑推进虚拟时间。它们通过与 Ossim 内核共享的每任务运行页 报告累计的虚拟时间推进。在常见路径上，内核会异步处理待处理的虚拟时间推进，避免每次报告都执行一次系统调用。当虚拟任务必须确保某次推进已被处理后才能继续时，也可以改用同步 ioctl。主机侧的执行本身不会推进建模虚拟任务的虚拟时间。Ossim 使用看门狗识别停滞的虚拟任务，并对停止推进的故障组件发出警告。

4.2 基于域的调度

共同的虚拟时间使异构虚拟任务的进度具有可比性，但协调这些进度还带来了若干额外挑战。首先，可运行的虚拟任务应在可能时独立推进，以利用并行性，同时将它们之间的虚拟时间差异限制在一定界限内。其次，当前无法取得进度的虚拟任务不应对其他任务造成不必要的限制。第三，有些同步策略无法通过常规调度器

仅靠钩子。最后，不同虚拟任务之间的同步要求可能有所不同。Ossim 通过域来应对这些挑战。域抽象出一组必须保持同步的虚拟任务。图 2 使用一个包含两个实时虚拟机和一个 ns-3 建模组件的示例仿真说明了这些机制。

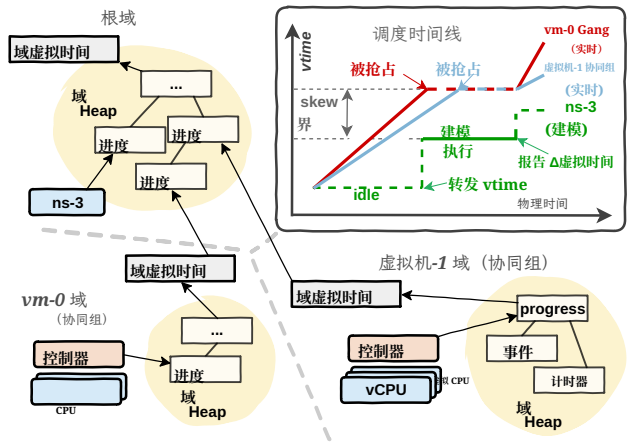


图 2。 一个包含两个实时虚拟机和一个建模的 ns-3 组件的示例仿真中的基于域的调度。域层次结构和约束协调它们的进度，而时间线展示了跨实时执行和建模执行的受偏斜界限约束的调度。

偏斜有界调度。 域通过约束其成员可推进的距离来协调这些成员。Ossim 将每个约束表示为一个约束节点，其中携带一个仿真时间上限。域将这些节点维护在一个最小堆中，称为域堆。最小值定义了域虚拟时间。

每个域 D 都有一个偏斜界限 $bound_D$，用于限制其成员可独立推进的幅度。它类似于保守并行离散事件仿真 [6, 16, 32], 中的前瞻，为独立的模拟时间推进提供了一个窗口。PDES 根据未来事件时间戳的下界（例如通信延迟）推导前瞻，而 Ossim 则在整个域内统一采用配置的界限。如 §8.3 所示，将偏斜界限保持在域内最小交互延迟以下，对于保持建模交互的时序保真度至关重要。

偏斜界限体现了准确性与性能之间的权衡：更大的界限允许更大的独立推进幅度和更高的并行性，而更小的界限则会减少由 vtime 偏斜导致的时序误差，但代价是需要更多同步。同一个堆还可以容纳除成员推进之外的其他约束。例如，第 §6 节稍后将使用这一机制来强制实现事件可见性。

处理非活动虚拟任务。 当虚拟任务在某个未来事件发生前无法取得进展时，例如 vCPU 已暂停或设备正在等待输入/输出，它就会变为非活动 状态。随后，Ossim 会将其进度节点从堆中移除，同时保留其虚拟时间。对于定时重新激活，事件节点 会记录目标虚拟时间；到达该时间后，Ossim 会将虚拟任务推进至该时间，并恢复其进度节点。对于非定时重新激活，Ossim 会先将其推进至当前域虚拟时间，然后再重新插入。

域控制器。为了支持更加灵活的调度策略，Ossim 允许每个域附加一个域控制器，即一个执行域级协调的内核线程，其功能超出普通调度器钩子的范围。控制器可以监控正在运行的成员，更新其调度状态，并在必要时中断它们。例如，实时 vCPU 域使用控制器来强制执行其偏斜界限，而无需依赖会干扰客户机执行的周期性调度器时钟节拍。控制器会推算正在运行的虚拟 CPU 的虚拟时间，更新其进度节点，并抢占任何变得不符合运行条件的虚拟 CPU。

异构同步。同一域中的不同虚拟任务可能需要不同的偏斜界限。若将最严格的要求应用于一个扁平域，就会不必要地限制所有成员，因此 Ossim 支持嵌套域。每个子域都执行自己的同步策略，并以子域虚拟时间为代表，作为单个成员参与其父域。

紧密耦合的实时虚拟任务可能需要接近零的 vtime 偏差。单纯采用较小的偏斜界限可能效率低下，因为即使只是稍微领先，也可能反复导致某个成员失去执行资格。因此，Ossim 提供了一个协同组，这是一种特殊的域，其控制器会将组内成员协同调度到不同的物理 CPU 上 [13]。并发执行能使它们的进度保持紧密同步，而协同组在其父域看来则是一个协调一致的成员。

SMP 实时虚拟机是协同组的主要应用场景。其虚拟 CPU 共享机器时钟和客户机内存，因此执行进度不一致可能导致客户机可见的时序不一致，以及异常的共享内存行为。如图 2 所示，每个实时虚拟机都在更大的仿真域中构成一个协同组。同一机制也可用于协调其他紧密耦合的组件，例如共享解聚内存的虚拟机。

5 实时内存层次结构管理

Ossim 在保持内存功能实时运行的同时，对其性能进行建模（C2）。它逐步采用三种机制，弥合主机与目标机内存行为之间的差距。首先，单元 将每个虚拟机隔离在受控的主机资源基底中（§5.1）。其次，基于配置文件的虚拟时间缩放 将主机上的内存停顿暴露映射到目标机时序（§5.2）。最后，内存区域归因 将停顿归因于各内存区域，并根据其目标机特征分别进行缩放（§5.3）。

5.1 基于单元的主机干扰控制

一个单元是 Ossim 为实时虚拟机提供主机资源隔离的单位。一个单元绑定一台实时虚拟机所需的主机资源。它为虚拟机分配专用 CPU 以及主机 NUMA 节点上的内存容量，同时限制其可用的共享缓存容量和内存带宽。当 QEMU 挂载虚拟机时，Ossim 会将其客户机 RAM 绑定到该节点，填充映射并使其常驻内存。

Ossim 通过 cpuset 风格的 CPU 和 NUMA 放置、[52], 内存容量准入，以及基于 resctrl 的缓存和带宽分配来实现这些控制。它使用英特尔缓存分配技术（CAT） [22] 进行缓存容量分配，并使用内存带宽分配（MBA） [23]限制内存带宽。CPU/NUMA 放置控制虚拟机在哪里执行以及其内存驻留在哪里，而 CAT 和 MBA 则限制它所消耗的共享资源。这些机制共同防止同机共置工作负载无意中改变实时虚拟机所观测到的内存行为。

5.2 基于配置文件的虚拟时间缩放

单元可以消除非预期的主机干扰，但无法消除主机与目标机内存之间系统性的性能差异。例如，宿主机 DRAM 可能比分离式内存更快。Ossim 根据每个虚拟 CPU 的执行过程中暴露于内存延迟的部分，对其虚拟时间进行缩放，以补偿这一差异。

考虑一个计量窗口，其中虚拟 CPU 执行时间的 f 部分暴露于内存停顿，剩余的 $1-f$ 部分为不受影响的执行时间。给定目标机与宿主机内存延迟比 r，Ossim 计算虚拟时间缩放因子 $s = (1-f) + fr = 1 + f(r-1)$。因此，只有暴露于内存延迟的执行时间会被缩放；其余执行时间保持不变。

Ossim 使用性能监控单元（PMU）计数器估算 f。在英特尔处理器上，它跟踪 CYCLE_ACTIVITY.STALLS_L3_MISS，该计数器统计需求加载未命中末级缓存 [22]时执行发生停顿的周期数。这一指标衡量暴露给执行过程的内存延迟，同时自然排除了被内存级并行性隐藏的延迟。该计数器附加到每个虚拟 CPU 上，并在客户机执行期间进行采样。Ossim 使用指数加权移动平均平滑所得的规模，并将其计入该虚拟 CPU 的虚拟时间乘数。

5.3 内存区域归因后端

当客户机访问具有不同目标端到主机端延迟比的内存区域时，单一的聚合停顿比例是不够的。我们以一种新兴配置为例，其中包含客户机本地 DRAM 和一个分离式内存区域 [9] 。Ossim 定义了一个独立于硬件的归因接口，其后端向通用虚拟时间缩放器报告各区域停顿占比。Ossim 目前提供三种此类后端。

NUMA源后端。在多 NUMA 的 Intel 主机上，Ossim 会将虚拟机本地内存放置在虚拟 CPU 所属的 NUMA 节点上，并将解聚内存区域放置在远程 NUMA 节点上。能够区分本地和远程 DRAM 访问的 PMU事件 [22] 使 Ossim 得以估算各区域的停顿占比。由于远程区域本身已产生额外的

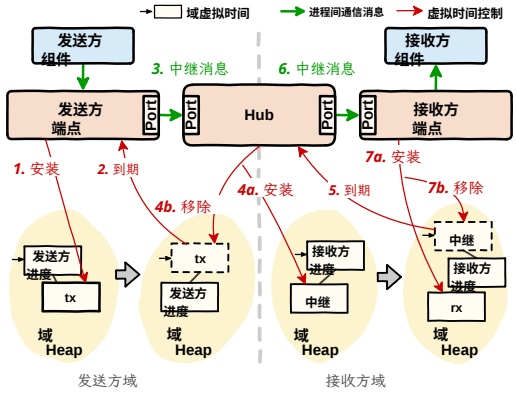


图 3. 消息路径与可见时间强制执行。

主机 NUMA 延迟，Ossim 在将这些停顿映射到目标延迟时会将这部分开销计算在内。此后端需要合适的 NUMA 拓扑，并支持一种解聚内存类别。

客户机辅助后端。 当 NUMA 放置无法区分内存区域时，Ossim 会通过客户机代理（§7.1）在客户机内部进行 PEBS 归因，此时采样的虚拟地址可用于识别被访问的区域。客户机会根据延迟对样本加权 [61]，并将所得停顿份额报告给主机，由主机应用相应的目标端到主机端比率。

物理地址后端。 在 AMD 主机上，IBS Op 会对内存访问延迟进行采样，并在可用时同时采样被访问的物理地址 [3]。Ossim 将采样地址映射到已注册内存区域，并直接计算其延迟加权停顿份额，无需 NUMA 放置或客户机侧归因。

6 仿真感知型进程间通信

Ossim 提供仿真感知型进程间通信，使跨组件通信遵循仿真时间，而非主机时间（C3）。下面介绍其三个基本组件。

6.1 IPC 抽象

如图 3 所示，Ossim 使用枢纽 和端点 来构成组件和通信网络结构。

枢纽。 枢纽对以太网等通信网络结构进行建模，在相连的端点之间路由消息，并捕捉消息穿越该网络结构所产生的性能影响。枢纽处理 Ossim 消息的不同变体；这些变体基于不同网络结构协议共用的一种抽象。

端点。 端点抽象了组件面向枢纽的通信接口，在特定接口的输入/输出与不同形式的 Ossim 消息之间进行转换。此外，它还可以对经过网卡等接口时产生的性能影响进行建模。端点通过不同组件已公开的原生 API 接入，以便与某种通信网络结构集成。对于以太网接口，例如，实时虚拟机会公开 QEMU vhost-user 设备作为 API，而 ns-3 等建模组件则会公开文件描述符网络设备。

端口。 枢纽和端点都具有端口，端口两两连接以交换 Ossim 消息。端口仅提供中继接口，不执行路由或性能建模，因此不会受所用 Ossim 消息变体的影响。

6.2 强制执行可见性时间交付

为了在仿真时间中同步通信，Ossim 为每条消息附加可见性虚拟时间，指明通信路径中的下一个节点何时可以观察到该消息。Ossim 使用第 4 节中的域虚拟时间堆来强制执行这些时间。

每个端点都关联到其所属组件的域。当端点本地操作（如网卡发送或接收）产生建模延迟时，端点会缓冲消息，并在其可见性虚拟时间处插入一个事件节点。图 3 中，发送端点首先安装发送 节点 (1)。当该节点到期时 (2)，端点将消息中继到枢纽 (3)。

每个中继都会先在目的域中建立下一个虚拟时间约束，然后再建立源域中的约束，以确保在途消息始终在某个域中受到虚拟时间约束，从而防止仿真进度超越可见性虚拟时间。因此，枢纽在对消息进行路由并应用建模网络结构的延迟后，会在接收方域中相应的可见性虚拟时间处安装一个事件节点，然后 释放发送方侧的约束（4a–4b）。

6.3 建模通信网络结构

通信网络结构在建模复杂度上差异很大：有些只需对链路行为进行参数化建模，而另一些则需要可编程转发或详细的外部模拟器。枢纽抽象让 Ossim 无需更改端点和端口接口，就能支持这些选择。Ossim 支持三种网络结构实现：内核内置模型、可编程 eBPF 模型和外部建模组件。

内置模型。 内置枢纽完全在内核中实现转发和计时，并提供可配置的性能参数。对于以太网，Ossim 提供点对点链路和交换机。它们适用于行为固定而性能参数可变的常见网络结构。

可编程模型。 对于定制网络结构，Ossim 使用逐消息 eBPF 钩子处理枢纽转发和计时，同时让消息路径保留在内核中。该钩子会选择目标端口并更新可见性虚拟时间，而 eBPF 映射则存储拓扑、路由和竞争状态。Ossim 为此提供了 eth_ebpf。

建模织物。 更复杂的建模织物可以作为外部建模组件运行，并拥有自身的状态和事件循环。

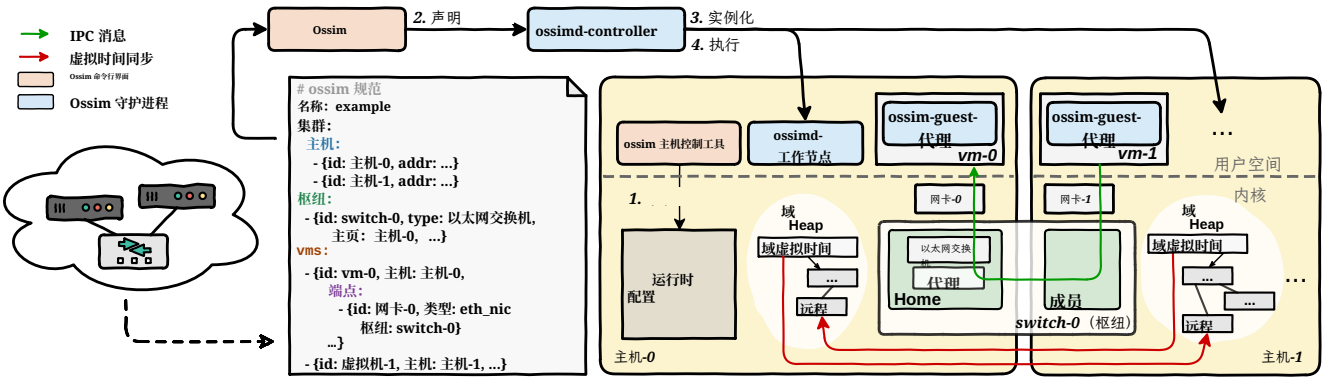


图 4。 在 Ossim 集群上运行一次仿真。控制平面负责将仿真从准备阶段推进到执行阶段，之后工作内核直接交换前沿更新和进程间通信消息。

它们通过相同的端点—中心接口进行连接，而 Ossim 会在界限处强制执行按可见性计时的交付。例如，ns-3 拓扑可以作为建模组件运行，通过点对点链路枢纽连接到实时虚拟机端点，同时保留 ns-3 的内部调度器和网络模型。

7 多主机协调

Ossim 通过将主机协调成一个 Ossim 集群，突破单台物理主机的规模限制（C4）。Ossim 将多主机协调分为两部分：内核数据平面在执行过程中跨越主机边界维护仿真语义（§7.1），而用户空间控制平面负责配置集群并管理仿真的生命周期（§7.2）。

7.1 数据平面

数据平面将 Ossim 的本地语义扩展到主机边界之外。尽管内存隔离仍局限于主机本地，但 Ossim 的调度和进程间通信保证必须借助额外机制，才能跨越参与主机而得到保留。Ossim 通过让主机内核经由主机间通道直接交换虚拟时间进度和定时进程间通信消息，来维持这些保证。

跨主机虚拟时间同步。跨主机虚拟时间同步可在各主机间保持有界偏差执行。每台主机都将其根域的虚拟时间发布给对等主机；对等主机则在其本地域堆中使用一个远程约束节点 表示所接收的值。该节点依据 §4.2 中相同的有界偏差分派规则约束域进度。当根域虚拟时间取得足够进展时，主机会发送更新。为应对极少发生的更新丢失情况，主机会定期重传其最新的虚拟时间。

跨主机进程间通信。 跨主机进程间通信可在不改变接口或时序语义的情况下，将逻辑集线器扩展到多个主机。无论对等端位于何处，端点都连接到同一个集线器；同时，集线器引擎仍会像第 6 节中那样确定路由和消息可见时间。因此，分布式对端点和集线器引擎而言都是透明的。

跨主机集线器通过代理端点表示远程端点。Ossim 使用一个位于<style id='3'>主实例的逻辑集线器，以及位于远程主机上的<style id='5'>成员 实例来实现这一点。在主实例中，每个远程端点都由一个代理端点表示，使集线器引擎能够像处理本地端点一样处理它。随后，Ossim 使用可靠的内核 TCP 将消息及其可见性元数据传输到相应的成员实例。这在保留建模的丢失语义的同时，使同一分布机制能够与不同的集线器引擎组合，而无需为其建模逻辑创建分布式变体。

7.2 控制平面

控制平面将集群级仿真描述转换为正在运行的实验。每台主机都运行一个工作节点守护进程（ossimd-worker），负责其本地仿真资源；此外，还运行一个控制器守护进程（ossimd-controller），负责协调各工作节点之间的操作。每台 Ossim 主机都作为工作节点运行，并负责其本地仿真资源；控制器则协调各工作节点之间的操作。客户端通过控制器访问 Ossim 集群。如图 4 所示，一次仿真依次经过准备、声明、实例化和执行阶段。

准备。准备会在每台主机上建立已知的测量状态。主机参与仿真之前，ossim 主机控制工具会对其进行管理，配置并验证与测量相关的设置，例如 CPU 电源管理、空闲状态和预留 CPU。对于依赖机器的参数，ossim 主机控制工具会使用随附的主机分析器推导出适当的值。

声明。 声明同时指定目标仿真及其在物理集群上的映射。通过 ossim 提交的 YAML 规范首先描述逻辑模拟：其组件、配置、IPC 集线器和连接。随后，它通过指定参与的工作节点以及组件放置和资源分配，将这些对象映射到物理资源。ossimd 控制器将该规范解析为集群范围的

计划，并在将其发送给工作节点之前验证其中的引用、放置和资源共享约束。

实例化。 实例化会创建经验证的仿真规范所需的资源。控制器负责协调集群范围内的操作，而每个工作节点则创建并拥有分配给自己的组件进程、内核对象及其他后备资源。只有在每个工作节点都报告本地资源已就绪后，准备/提交屏障才会完成实例化。

执行。 执行会运行已实例化的系统，并控制其组件内部的工作负载。为避免在客户机启动和实验设置期间产生同步开销，Ossim 可以先在不进行虚拟时间同步的情况下运行组件，并在进入目标感兴趣区域前切换到同步执行。可选的客户机内 ossim-guest-agent 通过 vsock 通信，并支持命令执行和结果收集等功能。它还会传送 §5 中所述的客户机辅助内存归因报告。对于未运行 ossim-guest-agent 的实时虚拟机，Ossim 提供控制台接口，作为交互式客户机控制的备用方式。

8 评估

我们评估 Ossim 能否兼顾全栈保真度与实用的仿真性能。我们首先测量端到端准确性和速度（§8.2），随后分别分析相关机制及其准确性–性能权衡（§8.3）。最后，通过两个案例研究考察 Ossim 与不同网络模型及实时硬件的组合（§8.4）。

8.1 实现与实验设置

实现。 Ossim 为 Linux 增加了 42 K 行代码，为 QEMU 增加了 3.0 K 行，为 ns-3 增加了 2.3 K 行，并在 libossim 中增加了 13.7 K 行，用于控制平面和操作员支持；此外，还包括约 15 K 行构建、操作员和工作负载工具代码。编码代理协助了实现和调试。

测试平台与对比设置。 实验使用三台相同的主机，每台配备双路英特尔至强金牌 6326 处理器（两个 NUMA 节点上共 64 个 CPU）和一块 Mellanox ConnectX-5 网卡，并通过具有 10Gb/s 链路的交换机连接。我们将 Ossim 与 物理环境（物理环境基准）、原版 *KVM*（未经修改的 KVM/QEMU，使用 Linux 网桥网络，并在需要时使用 netem 进行延迟或带宽整形），以及*gem5* 组合（SimBricks [32]，使用 gem5 端主机）进行比较。由硬件支持的设置使用相同的主机、Ubuntu 24.04 LTS 和 Linux 6.18；原版 KVM 使用 QEMU 8.2。目标机器默认配置为 4 个 CPU 和 16 GB DRAM。

Ossim 的主机分析器只需进行一次预实验运行，即可校准机器相关参数（§7.2）。所有 Ossim 实验都会复用此配置文件，该文件独立于工作负载及其目标结果。

表 2. 端到端保真度和仿真性能。 *N*为服务器实例的数量。

工作负载	*N*	指标	物理	原版 KVM（错误）	Ossim（错误）	墙钟时间 / 虚拟时间（秒）
memcached	2	p50（微秒）	66	127（93.4%）	66（0.0%）	155 / 60
Hotel（K8s）	2	p50（µs）	76	107（40.2%）	81（6.1%）	163 / 60
Hotel（分布式 K8s）	3	p50（µs）	61	95（54.9%）	58（5.4%）	171 / 60
TPC-C（MySQL）	2	p50（毫秒）	7.74	13.01（68.1%）	7.63（1.5%）	623 / 60
		tpmC	38,706	24,118（37.7%）	39,466（2.0%）	
Redis 主备	3	p50（µs）	84	106（25.3%）	87（3.2%）	167 / 60
TPC-DS 99（Spark）	3	运行时（s）	5.09	20.07（294.4%）	4.63（9.0%）	162 /6.72

8.2 端到端保真度与性能

Ossim 可运行未经修改的分布式软件栈，其端到端指标与物理测量结果的偏差在 9% 以内。表 2 涵盖 memcached、基于 Kubernetes 的单主机或多主机酒店预订、MySQL TPC-C、Redis 主备，以及包含三个实例的 Spark TPC-DS 工作负载。在报告的七项指标中，Ossim 的误差中位数为 3.2%，而原版 KVM 的偏差为 25–294%。因此，仅有软件兼容性并不能维持目标性能；仿真还必须控制执行、资源和通信。

这种保真度仍然切实可行：Ossim 每仿真一秒需要 2.6–24 个主机秒，而在 24 小时超时限制内，没有对应的 gem5 端到端实验能够完成。

8.3 准确性与性能微基准测试

为解释这些端到端结果，我们先单独考察调度、内存和通信控制（图 5），然后研究仿真成本和偏差界敏感性（图 6）。在可行时，我们将准确性与匹配的实际硬件进行比较；否则，则与配置的模型比较。由于 gem5 CPU 模型未经针对我们的处理器校准，因此我们仅将其用于仿真性能比较。

调度（§4）。Ossim 的仿真感知调度使客户机执行和唤醒时序在宿主机 CPU 争用时仍接近实际硬件表现。我们运行 sysbench CPU、线程和互斥锁测试、1ms 周期性定时器，以及双线程 futex 乒乓测试，同时让后台任务占用每个虚拟 CPU 所属物理核心 0%、10% 或 20% 的资源。

图 5(a)–(c) 展示了对客户机执行的影响。在没有后台负载时，原版 KVM 和 Ossim 的表现都接近物理机。随着负载增加，原版 KVM 的吞吐量会降低，或运行时间会延长。而 Ossim 的表现仍接近其隔离运行时的结果，因为客户机虚拟时间随计量执行时间推进，而非随宿主机墙上时间推进。

对于唤醒时序，图 5(d)–(e)显示，即使没有后台负载，这种差异也依然存在。在原版KVM下，最大定时器唤醒延迟为10.4× 物理，而在Ossim下仅为1.1× 。后台负载会进一步将原版 KVM 的定时器延迟提高到 56× 物理，并增加 futex 第 99 百分位延迟；而 Ossim 的协调调度使两项探测结果都接近物理值。

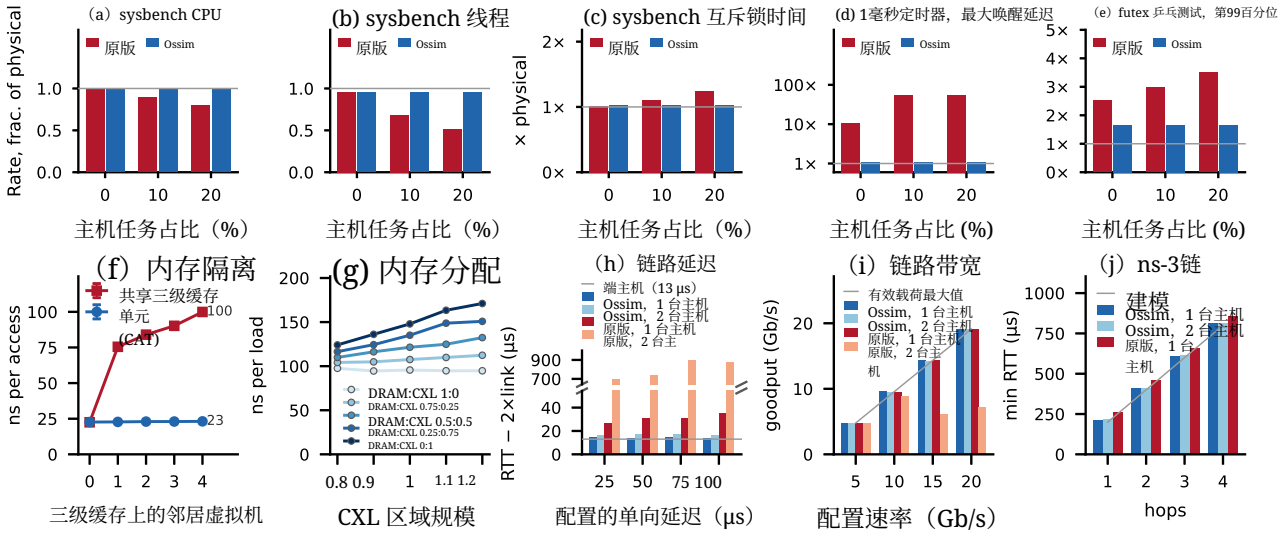


图 5. 机制层面的准确性。(a)–(e)评估不同主机 CPU 后台负载下的调度；(f)–(g)评估内存干扰隔离和各区域的性能缩放；(h)–(j)评估单主机内及跨主机的建模通信。(a)–(e)、(i)和(j)中的细灰线表示最优情况，(h)中的细灰线表示测得的端主机延迟。

内存管理（§5）。除了执行时序之外，Ossim 还同时控制内存干扰和区域特定性能。我们首先在一个虚拟机中使用依赖加载指针追逐测试隔离性，同时让最多四个相邻虚拟机在同一套接字的其他核心上遍历末级缓存（图 5(f)）。在没有隔离的情况下，延迟从 22.5 ns 上升到 100 ns；通过强制实施单元，延迟保持在 22.7–23.1 ns 之间，相较隔离情况下的结果偏差不超过 3%。

隔离可稳定实时内存底层，而各区域的缩放则会调整其建模性能（图 5(g)）。内存密集型追逐混合了本地 DRAM 和 CXL 访问，同时仅将 CXL 区域的缩放系数从 0.8 调整至 1.2。所有 DRAM 工作负载均保持稳定，所有 CXL 工作负载均随缩放系数变化，而混合工作负载则遵循按其 CXL 访问占比加权的目标值。这验证了 Ossim 的区域特定停顿归因和时序调整。

进程间通信与多主机协调（§6、§7）。除了控制每个实时虚拟机，Ossim 还必须保留组件之间的建模通信，包括跨主机通信。对于图 5(h)–(i)，原版 KVM 使用带有 netem 的 Linux 网桥，且不运行后台负载。

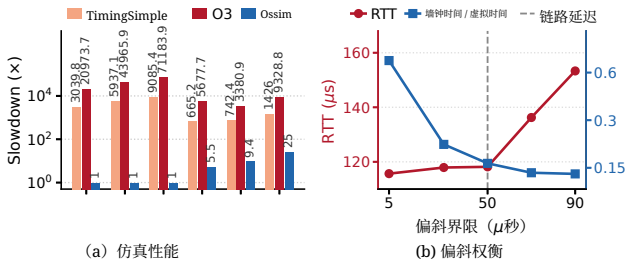
图 5(h)首先评估建模延迟。我们将单向链路延迟从 25 调整到 100 μs，并报告往返时间 − 2 ×链路延迟，该值应保持在实测的 13 μs 端主机贡献附近。Ossim 在单台主机上仅产生 14–15 μs 的额外延迟，在跨主机时为 16–17 μs；而原版 KVM 分别产生 27–35 μs 和 693–892 μs 的额外延迟。这一差距源于 netem 会在物理路径上叠加建模延迟，而 Ossim 则在仿真时间内控制消息可见性。

这种与物理路径的解耦，也使 Ossim 能够重现超出物理 10Gb/s 的建模带宽链路（图 5(i)）。两种配置在单台主机上都能达到配置的带宽，但跨主机时，原版 KVM 会受到物理链路的限制。相比之下，对于建模的 15 和 20Gb/s 链路，Ossim 分别实现了 14.20 和 19.10 Gb/s，与相应的 14.35 和 19.13 Gb/s 有效载荷上限相差不到 1.1%。

最后，图5(j)显示，同样的定时控制也适用于外部 ns-3 网络模型。对于每跳增加 100 μs 单向延迟的链路，两种系统都能复现预期的每跳 200 μs 往返时间增量。不过，Ossim 的延迟仅比模型高出 12–14 μs，而采用 ns-3 TAP集成的原版 KVM 高出 57–60 μs。将 Ossim 的虚拟机分布在两台主机上，并在其中一台上运行 ns-3，只会额外增加约 1 μs，同时保持相同的斜率。我们的原版 KVM 基线没有对应的双主机配置，因为其 ns-3 TAP集成仅在一台主机内连接实时客户机和 ns-3。

仿真性能。此前的实验评估了准确性，而图6(a)考察了真实终端主机执行相较于 gem5 组合所带来的仿真性能优势。对于计算密集型sysbench，将对称多处理宽度从一个 CPU 增至三个 CPU，会增加 gem5 使用 TimingSimple 和 O3模型时的开销。相比之下，Ossim 的开销几乎保持不变，因为新增的虚拟 CPU 可直接在新增的物理核心上并行运行。

因此，在分布式gem5实验中，我们为每个仿真主机仅配置一个 CPU，这种设置有利于 gem5 组合。即使采用这种配置，Ossim 每仿真一秒仍需 5.5–25 墙钟秒，而采用 SimpleTiming CPU 的 gem5 组合需要 665–1426 秒，采用 O3 则需要 3381–9329 秒。在相应的工作负载下，Ossim 的速度比 TimingSimple 快 57–121× 倍，比 O3 快 360–1032× 倍。这表明，实时执行消除了



（a）仿真性能 (b) 偏斜权衡

图6. (a) 仿真性能与 gem5 组件组合的关系。(b) RTT 和仿真成本（每仿真秒的墙钟秒数）随偏斜界限的变化；虚线表示建模的单向延迟。

在典型工作负载下，原本会成为全栈模块化组合主要开销的端主机仿真成本。

偏斜界限权衡。 实时执行无需进行详细的 CPU 仿真，但其协调开销仍取决于组件同步的紧密程度。图6(b)通过将界限从 5 扫描至 90 μs，考察两个位于不同协同组中的虚拟机的准确性–性能权衡。这两个虚拟机通过单向延迟为 50 μs 的链路交换 ping 数据包。将界限从 5 增至 50 μs，可使 RTT 保持在 118 μs 左右，同时将成本从每仿真秒 0.715 主机秒降至 0.160 主机秒（4.5×）。然而，超过建模的单向延迟后，RTT 会先增加 18，随后再增加 35 μs，而成本仅略有降低。这些结果表明，所选界限不应大于与实验相关的最小建模单向延迟。

8.4 案例研究

在考察保真度和性能之后，我们通过两个案例研究展示 Ossim 的灵活性和通用性。首先，我们将相同的实时虚拟机与不同的底层网络实现组合起来，然后添加一个物理 DPU 作为实时组件。

组合胖树网络。 第一个案例研究在不改变虚拟机接口的情况下更改网络模型。我们将 16 个单 vCPU 虚拟机连接到一个每跳延迟为 k=4 秒的胖树网络，使用 eBPF 集线器或外部 ns-3 组件（§6.3）。eBPF 集线器在可编程集线器中实现拓扑、路由和计时，而 ns-3 则将网络作为外部建模组件提供，从而可以利用建模主机添加额外的背景流量。如表 3(a) 所示，两种网络实现都复现了拓扑层级之间预期的 40 μ秒往返时间增量。

组合物理 DPU。 不同于胖树案例对建模网络结构进行变化，第二个案例将实时执行扩展到物理 BlueField-2 数据处理器，以实现数据包处理卸载。Ossim 提供了一个内核内后端，通过两个专用虚拟功能（VF）将物理 DPU 连接到虚拟机的网卡端点。数据包通过其中一个 VF 提交，并通过另一个 VF 接收由 DPU 重写后的数据包

表 3。 案例研究。（a）胖树中虚拟机对之间的往返时间。（b）通过网关虚拟机 CPU 和 DPU 路径进行网络地址转换的往返时间。

(a) 胖树往返时间（μs）

虚拟机对	eBPF	ns-3
同一边缘	59.7	57.8
同一 Pod	99.4	98.1
跨 Pod	139.8	137.5

(b) 网络地址转换往返时间（μs）

工作节点	CPU	DPU 软件	DPU 专用集成电路
0	70.04	72.71	54.89
4	149.16	71.43	53.95
8	163.12	72.02	54.29

。对于来自相反方向的流量，VF 的角色互换。该后端测量每个数据包的 DPU 处理时间，并据此安排其在虚拟时间中的完成时间。

借助这一集成方案，我们比较了网关虚拟机 Linux 协议栈、DPU 软件和 DPU 专用集成电路中的网络地址转换。三个实时虚拟机——客户端、网关和服务器——通过以太网交换集线器连接；DPU 路径绕过网关的网络协议栈。为比较对网关负载的敏感性，我们在匹配规则前放置 1,024 条不匹配的网络地址转换规则，并用在网关内部运行的工作节点对其施加压力；预热后，我们针对每种配置测量 100 次 UDP 回显。随着负载增加，CPU 路径的往返时间从 70.04 上升至 163.12 μs，而 DPU 软件和专用集成电路路径则分别保持在约 72 和 54 μs（表 3(b)）。

9 相关工作与讨论

仿真加速。 除了实时仿真，先前的工作还通过系统和运行时优化 [6, 11, 17, 33, 35, 37], 学习型性能模型 [14, 31, 46, 59, 62], 以及 GPU 和 FPGA 等专用硬件 [7, 20, 26, 28, 34, 42]来加速仿真。

分布式软件栈的仿真。 其他工作针对特定的集群工作负载，尤其是 GPU 训练 [19,43, 56],，通过同时仿真分布式训练策略和底层硬件行为来实现这一目标。相比之下，Ossim 专注于未经修改的全栈分布式软件，以及其在集群规模下与底层硬件的交互。

为客户机提供资源隔离。 分区管理程序 [8,27, 38, 44] 为每个客户机静态分配固定的主机资源，并通过严格禁止共享缓存和内存硬件来实现隔离。相比之下，Ossim 会随着仿真客户机的加入和离开动态分配主机资源，从而在受控共享条件下保持仿真准确性。

讨论。 单元超额配置需要对实时存储层次结构进行时间共享。即使调度器以时间片方式调度虚拟 CPU，新进入的实时虚拟机也可能继承前一台虚拟机的缓存和预取器状态。要保持保真度，就必须重新调节这些状态。时间性缓存重新调节不在本文研究范围内，Ossim 不支持安全的单元超额配置。除了基于 KVM 的虚拟机外，集成其他虚拟化技术 [10, 12, 39, 47, 58, 60, 63]还可将 Ossim 扩展到 GPU 等其他实时硬件。

10 结论

Ossim 采用操作系统级方法，通过将面向仿真的机制集成到 Linux 虚拟化栈中，对未经修改的分布式软件栈进行集群规模全栈仿真。Ossim 让生产软件保持实时运行，同时由操作系统机制控制仿真时间、内存、通信及其在各主机间的分布。我们认为，Ossim 是迈向仿真原生操作系统支持的初步尝试，这代表着一种比集群规模全栈仿真更广阔的发展方向。

参考文献

[1] 2025。Ultra Ethernet 规范 v1.0.1。https://ultraethernet.org/ uec-1-0-spec 访问日期：2026 年 5 月 18 日。[2] 2026。Ultra Accelerator Link 规范。https://ualinkconsortium.org/specification/。访问日期：2026 年 5 月 18 日。[3] Advanced Micro Devices, Inc. 2026。*AMD64* 架构程序员手册，第 *2* 卷：系统编程。出版物 24593，修订版 3.42；访问日期：2026 年 8 月 24 日。[4] Apache 软件基金会。2026。Apache Hadoop：用于分布式存储和处理的开源框架。https://hadoop. apache.org/ 访问日期：2026 年 5 月 18 日。[5] Apache 软件基金会。2026。Apache Spark：面向大规模数据处理的统一分析引擎。https://spark.apache.org/ 访问日期：2026 年 5 月 18 日。[6] Songyuan Bai、Hao Zheng、Chen Tian、Xiaoliang Wang、Chang Liu、Xin Jin、Fu Xiao、Qiao Xiang、Wanchun Dou 和 Guihai Chen。2024。Unison：高并行效率且对用户透明的网络仿真内核。在第十九届欧洲计算机系统会议论文集（*EuroSys '24*）中。[7] Ben Romdhanne Bilel 和 Nikaein Navid。2012。Cunetsim：用于大规模移动网络的基于 GPU 的仿真测试平台。在*2012* 年通信与信息技术国际会议（*ICCIT*）中。IEEE，374–378。[8] T. L. Borden、J. P. Hennessy 和 J. W. Rymarczyk。1989。单个处理器复合体上的多个操作系统。*IBM Systems Journal* 28, 1 1989 104 123 10 1147 281 0104（），–。doi：。/sj。。[9] 2025。CXL 联盟。。Compute Express Link（CXL）规范第 4.0 版。https://www.computeexpresslink.org/download-the- specification。访问日期：2026 年 5 月 19 日。[10] Niklas Eiling、Jonas Baude、Stefan Lankes 和 Antonello Monti。2022。Cricket：支持检查点/重启的 CUDA 应用程序分布式执行虚拟化层。*Concurrency and Computation: Practice and Experience* 34，14（2022），e6474。arXiv：https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6474 doi：10. 1002/cpe.6474[11] Fares Elsabbagh、Shabnam Sheikhha、Victor A Ying、Quan M Nguyen、Joel S Emer 和 Daniel Sanchez。2023。通过软硬件协同设计加速 RTL 仿真。在第 *56* 届 *IEEE/ACM* 国际微体系结构研讨会论文集中。153–166。[12] Ruwen Fan、Tingxu Ren、Minhui Xie、Shiwei Gao、Jiwu Shu 和 Youyou Lu。2025。GPREEMPT：通用且高效的 GPU 抢占式调度。在*2025* 年 *USENIX* 年度技术大会（*USENIX ATC 25*）中。USENIX 协会，马萨诸塞州波士顿，263–272。[13] Dror G Feitelson 和 Larry Rudolph。1992。细粒度同步的协同调度性能优势。*Journal of Parallel and distributed Computing* 16，4（1992），306–318。[14] Miquel Ferriol-Galmés、Jordi Paillisse、José Suárez-Varela、Krzysztof Rusek、Shihan Xiao、Xiang Shi、Xiangle Cheng、Pere Barlet-Ros 和 Albert Cabellos-Aparicio。2023。RouteNet-Fermi：使用图神经网络进行网络建模。*IEEE/ACMtransactionsonnetworking* 31，6，2023，3080，3095（），–。[15] Daniel Firestone、Andrew Putnam、Sambhrama Mundkur、Derek Chiou、Alireza Dabagh、Mike Andrewartha、Hari Angepat、Vivek Bhanu、Adrian Caulfield、Eric Chung 等。2018。Azure 加速网络：公有云中的智能网卡。发表于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*）。51–66。[16] Richard M Fujimoto。1990。并行离散事件仿真。*ACM* 通讯33，10（1990），30–53。[17] Kaihui Gao、Li Chen、Dan Li、Vincent Liu、Xizheng Wang、Ran Zhang 和 Lu Lu。2023。Dons：通过自动并行化实现快速、经济的离散事件网络仿真。发表于*ACM SIGCOMM 2023* 会议论文集。167–181。[18] gem5 项目。2026。gem5 仿真器项目。https://www.gem5.org/。访问日期：2026-05-19。[19] Fei Gui、Kaihui Gao、Li Chen、Dan Li、Vincent Liu、Ran Zhang、Hong-bing Yang 和 Dian Xiong。2025。通过多实验并行仿真加速大语言模型训练系统的设计空间探索。发表于第 *22* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 25*）。473–488。[20] Zizheng Guo、Yanqing Zhang、Runsheng Wang、Yibo Lin 和 Haoxing Ren。2025。GEM：GPU 加速的仿真器启发式 RTL 仿真。发表于*2025* 年第 *62* 届 *ACM/IEEE* 设计自动化会议（*DAC*）。IEEE，1–7。[21] Diwaker Gupta 和 Kashi V. Vishwanath。2008。DieCast：使用精确的比例模型测试分布式系统。发表于第 *5* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI '08*）。USENIX 协会，美国加利福尼亚州旧金山。[22] 英特尔公司。2026。*Intel(R) 64* 和 *IA-32* 架构软件开发人员手册。访问日期：2026-05-19。[23] 英特尔公司。2026。内存带宽分配简介。https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-memory-bandwidth-allocation.html。访问日期：2026-05-20。[24] Intel Simics。2026。Simics 全系统仿真器。https://www.intel.com/content/www/us/en/developer/articles/tool/simics-simulator.html。访问日期：2026-05-19。[25] Norman P Jouppi、Cliff Young、Nishant Patil、David Patterson、Gaurav Agrawal、Raminder Bajwa、Sarah Bates、Suresh Bhatia、Nan Boden、Al Borchers 等。2017。张量处理单元在数据中心内的性能分析。发表于第 *44* 届年度国际计算机体系结构研讨会论文集。1–12。[26] Sagar Karandikar、Howard Mao、Donggyu Kim、David Biancolin、Alon Amid、Dayeol Lee、Nathan Pemberton、Emmanuel Amaro、Colin Schmidt、Aditya Chopra 等。2018。FireSim：公有云中的 FPGA 加速周期精确扩展式系统仿真。发表于*2018* 年 *ACM/IEEE* 第 *45* 届年度国际计算机体系结构研讨会（*ISCA*）。IEEE，29–42。[27] Eric Keller、Jakub Szefer、Jennifer Rexford 和 Ruby B. Lee。2010。NoHype：无虚拟化的虚拟化云基础设施。发表于第 *37* 届年度国际计算机体系结构研讨会论文集（*ISCA '10*）。350–361。doi:10.1145/1815961.1816010[28] Sajy Khashab、Hariharan Sezhiyan、Rani Abboud、Alex Normatov、Stefan Kaestle、Eliav Bar-Ilan、Mohammad Nassar、Omer Shabtai、Wei Bai、Matty Kadosh 等。2025。NSX：人工智能服务器上的大规模网络仿真。发表于第 *2* 届人工智能计算网络研讨会论文集。19–25。[29] Jereme Lamps、Vladimir Adam、David M. Nicol 和 Matthew Caesar。2015。在 Linux 多处理器上结合仿真与网络模拟器。发表于第 *3* 届 *ACM SIGSIM* 高级离散仿真原理会议论文集 (英国伦敦) *(SIGSIM PADS '15)*。计算机协会，纽约，美国，113–124。doi:10.1145/2769458.2769481[30] Jereme Lamps、David M. Nicol 和 Matthew Caesar。2014。Time-Keeper：用于 Linux 的轻量级虚拟时间系统。发表于论文集

第 *2* 届 *ACM SIGSIM* 高级离散仿真原理会议论文集(美国科罗拉多州丹佛) *(SIGSIM PADS '14)*。计算机协会，纽约，美国，179–186。doi:10.1145/2601381.2601395[31] Chenning Li、Arash Nasr-Esfahany、Kevin Zhao、Kimia Noorbakhsh、Prateesh Goyal、Mohammad Alizadeh 和 Thomas E Anderson。2024。m3：使用机器学习进行准确的流级性能估算。发表于*ACM SIGCOMM 2024* 会议论文集。813–827。[32] Hejing Li、Jialin Li 和 Antoine Kaufmann。2022。SimBricks：通过模块化仿真进行端到端网络系统评估。发表于*ACM SIGCOMM 2022* 会议论文集。380–396。[33] Hejing Li、Marvin Meiers、Jialin Li 和 Antoine Kaufmann。2025。SplitSim：迈向面向系统研究的实用大规模全系统仿真。*ACM* 网络期刊 3，CoNEXT4（2025），1–19。doi:10.1145/3768999[34] Qinyong Li、Zhiwei Zhao、Geyong Min、Zi Wang 和 Luwei Fu。2026。GeDES：GPU 驱动的离散事件网络模拟器。发表于第 *21* 届欧洲计算机系统会议论文集。468–483。[35] Tiantian Lin、Cheng Qiu、Xiaohang Wang、Ling Wang、Zhulin Zheng、Yingtao Jiang、Amit Kumar Singh、Jieming Yin、Sihai Qiu、Xiaodong Li 等。2025。LEGOSim：面向多芯粒异构集成的统一并行仿真框架。发表于第 *58* 届 *IEEE/ACM* 国际微体系结构研讨会论文集。1347–1362。[36] Jiacheng Ma、Jonas Kaufmann、Emilien Guandalino、Rishabh Iyer、Bourgeat Thomas 和 George Candea。2025。加速硬件—软件栈的快速端到端性能仿真。发表于*ACM SIGOPS* 第 *31* 届操作系统原理研讨会论文集（*SOSP '25*）。[37] Tie Ma、Long Luo、Hongfang Yu、Xi Chen、Jingzhao Xie、Chongxi Ma、Yunhan Xie、Gang Sun、Tianxi Wei、Li Chen 等。2024。Klonet：易于使用且可扩展的计算机网络教育平台。发表于第 *21* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 24*）。2025–2046。[38] José Martins、Adriano Tavares、Marco Solieri、Marko Bertogna 和 Sandro Pinto。2020。Bao：面向现代多核嵌入式系统的轻量级静态分区虚拟机监控器。发表于与 *HiPEAC* 联合举办的下一代实时嵌入式系统研讨会（*NG-RES*）。3:1–3:14。doi:10.4230/OASIcs.NG-RES.2020.3[39] 英伟达公司。2026。NVIDIA 虚拟 GPU（vGPU）软件。https://docs.nvidia.com/vgpu/。访问日期：2026-05-19。[40] OpenInfra 基金会。2026。全球部署最广泛的开源云软件。https://www.openstack.org 访问日期：2026-05-19。[41] QEMU 项目。2026。QEMU：快速处理器模拟器。https://www.qemu.org/。访问日期：2026-05-18。[42] Yicheng Qian、Ran Shu、Rui Ma、Yang Wang、Derek Chiou、Nadeen Gebara、Luca Piccolboni、Miriam Leeser 和 Yongqiang Xiong。2025。Miniature：基于 FPGA 的快速人工智能超级计算机网络仿真。发表于第 *9* 届亚太网络研讨会论文集。114–120。[43] Jianxing Qin 、 Jingrong Chen、Xinhao Kong、Yongji Wu、Tianjun Yuan、Liang Luo、Zhaodong Wang、Ying Zhang、Tingjun Chen、Alvin R Lebeck 等。2026。Phantora：最大限度复用代码的仿真式机器学习系统性能估算。发表于第 *23* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 26*）。1809–1825。[44] Ralf Ramsauer、Jan Kiszka、Daniel Lohmann 和 Wolfgang Mauerer。2017。瞧，妈妈，没有虚拟机退出！（几乎）。发表于第 *13* 届嵌入式实时应用操作系统平台研讨会论文集（*OSPERT*）。https://arxiv.org/abs/1705.06932。[45] George F. Riley 和 Thomas R. Henderson。2010。ns-3 网络模拟器。发表于网络仿真的建模与工具。施普林格，15–34。doi:10.1007/978-3-642-12331-3_2[46] Krzysztof Rusek、José Suárez-Varela、Paul Almasan、Pere Barlet-Ros 和 Albert Cabellos-Aparicio。2020。RouteNet：利用图神经网络进行软件定义网络建模与优化。*IEEE* 通信领域精选期刊 38, 10 (2020), 2260–2270.[47] Weihang Shen、Mingcong Han、Jialong Liu、Rong Chen 和 Haibo Chen。2025 年。{XSched}：面向多种 {XPU} 的抢占式调度。发表于第 *19* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 25*）。671–692.[48] Jovan Stojkovic、Abraham Farrell、Zhangxiaowen Gong、Christopher J Hughes 和 Josep Torrellas。2026 年。AccelFlow：为微服务编排封装内的细粒度加速器集群。发表于*2026* 年 *IEEE* 高性能计算机体系结构国际研讨会（*HPCA*）。IEEE，1–17.[49] Yan Sun、Yifan Yuan、Zeduo Yu、Reese Kuper、Chihun Song、Jinghan Huang、Houxiang Ji、Siddharth Agarwal、Jiaqi Lou、Ipoom Jeong 等。2023 年。通过真正支持 CXL 的系统和设备揭开 CXL 内存的神秘面纱。发表于第 *56* 届 *IEEE/ACM* 国际微体系结构研讨会论文集。105–121。[50] libvirt 项目。2026 年。libvirt 虚拟化 API。https://libvirt.org/ 访问日期：2026-05-19。[51] Linux 基金会。2026 年。Kubernetes：生产级容器编排。https://kubernetes.io/ 访问日期：2026-05-19。[52] Linux 内核基金会。2026 年。CPU 集。https://docs.kernel.org/ admin-guide/cgroup-v1/cpusets.html。访问日期：2026-05-20。[53] Linux 内核基金会。2026 年。KVM：基于内核的虚拟机。https://www.linux-kvm.org/。访问日期：2026-05-18。[54] Linux 内核基金会。2026 年。KVM PVclock。https://www.linux- kvm.org/page/KVMClock。访问日期：2026-05-20。[55] András Varga 和 Rudolf Hornig。2008 年。OM- NeT++ 仿真环境概述。发表于第 *1* 届通信、网络与系统仿真工具和技术国际会议论文集（*SIMUTools '08*）。ICST，法国马赛，1–10。doi:10.4108/ICST.SIMUTOOLS2008.3027[56] Xizheng Wang、Qingxu Li、Yichi Xu、Gang Lu、Dan Li、Li Chen、Heyang Zhou、Linkang Zheng、Sen Zhang、Yikai Zhu 等。2025 年。{SimAI}：以可扩展性和精度统一大规模语言模型训练的架构设计与性能调优。发表于第 *22* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 25*）。541–558.[57] Elias Weingärtner、Florian Schmidt、Hendrik vom Lehn、Tobias Heer 和 Klaus Wehrle。2011 年。SliceTime：可扩展且精确的网络仿真平台。发表于第 *8* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI '11*）。USENIX 协会，马萨诸塞州波士顿。[58] Yechen Xu、Yifei Wang、Nathanael Ren、Yiran Chen 和 Danyang Zhuo。2026 年。Nixie：面向消费级 GPU 的高效、透明时间复用。发表于第 *20* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 26*）。USENIX 协会，美国华盛顿州西雅图，2085–2101。[59] Qingqing Yang、Xi Peng、Li Chen、Libin Liu、Jingze Zhang、Hong Xu、Baochun Li 和 Gong Zhang。2022 年。DeepQueueNet：借助数据包级可见性实现可扩展且通用的网络性能估算。发表于*ACM SIGCOMM 2022* 会议论文集。441–457。[60] Yanan Yang、Zhengxiong Jiang、Meiqi Zhu、Hongqiang Xu、Yujun Wang、Liang Li、Jiansong zhang 和 Jie Wu。2026 年。gShare：在多租户 FaaS 平台中通过激进调度实现高效 GPU 共享。发表于第 *31* 届 *ACM* 编程语言与操作系统体系结构支持国际会议论文集，第 *2* 卷（美国）（*ASPLOS '26*）。计算机协会，纽约，美国，838–859。doi:10.1145/3779212.3790168[61] Quan Yi、Xu Liu 和 Shengtian Yao。2020 年。精确事件采样的精度探讨。发表于第 *11* 届 *ACM SIGOPS* 亚太系统研讨会论文集（*APSys '20*）。98–105。doi:10.1145/3409963. 3410974[62] Qizhen Zhang、Kelvin KW Ng、Charles Kazer、Shen Yan、João Sedoc 和 Vincent Liu。2021 年。MimicNet：利用机器学习快速估算数据

中心网络的性能。发表于*2021 年 ACM SIGCOMM 2021 会议论文集*。287–304。[63] Shulai Zhang、Ao Xu、Quan Chen、Han Zhao、Weihao Cui、Zhen Wang、Yan Li、Limin Xiao 和 Minyi Guo。2025 年。高效且注重性能的通过内核空间拦截实现兼容性与隔离的 GPU 共享。载于*2025 年 USENIX 年度技术大会（USENIX ATC 25）*。USENIX 协会，马萨诸塞州波士顿，1003–1019。