openEuler 24.03 LTS SP4 技术白皮书

1. 概述

OpenAtom openEuler（简称“openEuler”） 社区是一个面向数字基础设施操作系统的开源社区。由开放原子开源基金会（以下简称“基金会”）孵化及运营。

openEuler 是一个面向数字基础设施的操作系统，支持服务器、云计算、边缘计算、嵌入式等应用场景，支持多样性计算，致力于提供安全、稳定、易用的操作系统。通过为应用提供确定性保障能力，支持 OT 领域应用及 OT 与 ICT 的融合。

openEuler 社区通过开放的社区形式与全球的开发者共同构建一个开放、多元和架构包容的软件生态体系，孵化支持多种处理器架构、覆盖数字基础设施全场景，推动企业数字基础设施软硬件、应用生态繁荣发展。

2019 年 12 月 31 日，面向多样性计算的操作系统开源社区 openEuler 正式成立。

2020 年 3 月 30 日， openEuler 20.03 LTS （Long Term Support，简写为 LTS，中文为长生命周期支持）版本正式发布，为 Linux 世界带来一个全新的具备独立技术演进能力的 Linux 发行版。

2020 年 9 月 30 日，首个 openEuler 20.09 创新版发布，该版本是 openEuler 社区中的多个企业、团队、独立开发者协同开发的成果，在 openEuler 社区的发展进程中具有里程碑式的意义，也是中国开源历史上的标志性事件。

2021 年 3 月 31 日，发布 openEuler 21.03 内核创新版，该版本将内核升级到 5.10，并在内核方向实现内核热升级、内存分级扩展等多个创新特性，加速提升多核性能，构筑千核运算能力。

2021 年 9 月 30 日，全新 openEuler 21.09 创新版如期而至，这是 openEuler 全新发布后的第一个社区版本，实现了全场景支持。增强服务器和云计算的特性，发布面向云原生的业务混部 CPU 调度算法、容器化操作系统 KubeOS 等关键技术；同时发布边缘和嵌入式版本。

2022 年 3 月 30 日，基于统一的 5.10 内核，发布面向服务器、云计算、边缘计算、嵌入式的全场景 openEuler 22.03 LTS 版本，聚焦算力释放，持续提升资源利用率，打造全场景协同的数字基础设施操作系统。

2022 年 9 月 30 日，发布 openEuler 22.09 创新版本，持续补齐全场景的支持。

2022 年 12 月 30 日，发布 openEuler 22.03 LTS SP1 版本，打造最佳迁移工具实现业务无感迁移，性能持续领先。

2023 年 3 月 30 日，发布 openEuler 23.03 内核创新版本，采用 6.1 内核，为未来 openEuler 长生命周期版本采用 6.x 内核提前进行技术探索，方便开发者进行硬件适配、基础技术创新及上层应用创新。

2023 年 6 月 30 日，发布 openEuler 22.03 LTS SP2 版本，场景化竞争力特性增强，性能持续提升。

2023 年 9 月 30 日，发布 openEuler 23.09 创新版本，是基于 6.4 内核的创新版本（参见版本生命周期），提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2023 年 11 月 30 日，发布 openEuler 20.03 LTS SP4 版本，其作为 20.03 LTS 版本的增强扩展版本，面向服务器、云原生、边缘计算场景，提供更多新特性和功能增强。

2023 年 12 月 30 日，发布 openEuler 22.03 LTS SP3 版本，是 22.03 LTS 版本增强扩展版本，面向服务器、云原生、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 5 月 30 日，发布 openEuler 24.03 LTS，基于 6.6 内核的长周期 LTS 版本（参见版本生命周期），面向服务器、云、边缘计算、AI 和嵌入式场景，提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 6 月 30 日，发布 openEuler 22.03 LTS SP4，是 22.03 LTS 版本增强扩展版本，面向服务器、云原生、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 9 月 30 日，发布 openEuler 24.09，基于 6.6 内核的创新版本，提供更多新特性和功能。

2024 年 12 月 30 日，发布 openEuler 24.03 LTS SP1，基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 3 月 30 日，发布 openEuler 25.03 是基于 6.6 内核的创新版本，面向服务器、云、边缘计算和嵌入式场景，提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

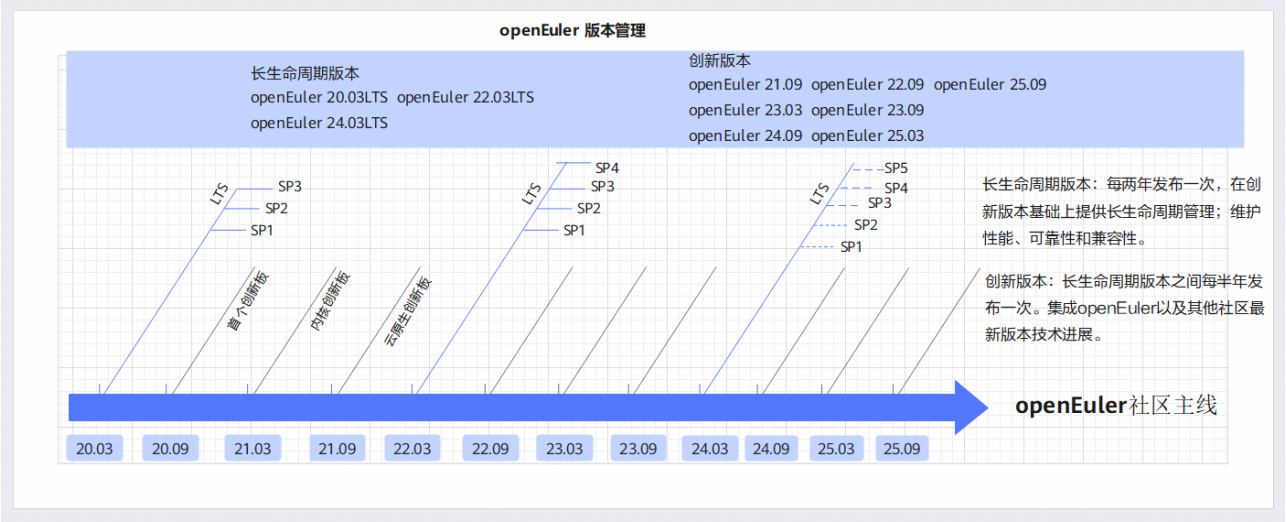
2025 年 6 月 30 日，发布 openEuler 24.03 LTS SP2，基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、AI 和嵌入式场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、众核高密、机密容器、多核多实例混部等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 9 月 30 日，发布 openEuler 25.09，基于 6.6 内核创新版本，面向服务器、云、AI 和嵌入式场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、众核高密、机密容器、机密虚机、

多核多实例混部、编译器、可信计算、密码加速等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 12 月 30 日，openEuler 首个支持超节点的版本正式发布。新版本 openEuler 24.03 LTS SP3 是基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、AI 场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、智能诊断、机密虚机、编译器、RISC-V 架构优化、智能开发者桌面、安全加固、灵衢超节点、身份认证、虚拟化等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2026 年 6 月 30 日，发布 openEuler 24.03 LTS SP4，基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、AI 场景，持续提供更多新特性和功能扩展，包括内核优化、灵衢超节点可靠性&易用性、NPU 算力切分、推理服务快恢、E2B 沙箱、智能诊断&调优&运维、编译器、机密虚机等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

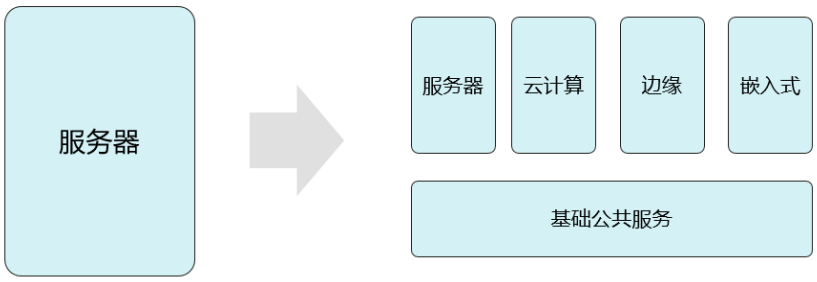


openEuler 作为一个操作系统发行版平台，其 LTS 版本为企业级用户提供一个安全稳定可靠的操作系统。

openEuler 也是一个技术孵化器。通过每半年发布一个创新版，快速集成 openEuler 以及其他社区的最新技术成果，将社区验证成熟的特性逐步回合到发行版中。这些新特性以单个开源项目的方式存在于社区，方便开发者获得源代码，也方便其他开源社区使用。

社区中的最新技术成果持续合入社区发行版，社区发行版通过用户反馈反哺技术，激发社区创新活力，从而不断孵化新技术。发行版平台和技术孵化器互相促进、互相推动、牵引版本持续演进。

openEuler 覆盖全场景的创新平台



openEuler 已支持 X86、ARM、RISC-V、LoongArch 多处理器架构，持续完善多样性算力生态体验。

openEuler 社区面向场景化的 SIG 不断组建，推动 openEuler 应用边界从最初的服务器场景，逐步拓展到云计算、边缘计算、嵌入式等更多场景，openEuler 正成为覆盖数字基础设施全场景的操作系统。

openEuler 希望与广大生态伙伴、用户、开发者一起，通过联合创新、社区共建，不断增强场景化能力，最终实现统一操作系统支持多设备，应用一次开发覆盖全场景。

openEuler 开放透明的开源软件供应链管理

开源操作系统的构建过程，也是供应链聚合优化的过程。拥有可靠开源软件供应链，是大规模商用操作系统的基础。 openEuler 从用户场景出发，回溯梳理相应的软件依赖关系，理清所有软件包的上游社区地址、源码和上游对应验证。完成构建验证、分发、实现生命周期管理。开源软件的构建、运行依赖关系、上游社区，三者之前形成闭环且完整透明的软件供应链管理。

2. 平台架构

系统框架

openEuler 是覆盖全场景的创新平台，在引领内核创新，夯实云化基座的基础上，面向计算架构互联总线、存储介质发展新趋势，创新分布式、实时加速引擎和基础服务，结合边缘、嵌入式领域竞争力探索，打造全场景协同的面向数字基础设施的开源操作系统。

openEuler 24.03 LTS SP4 发布面向服务器、云原生、边缘场景的全场景操作系统版本，统一基于 Linux Kernel 6.6 构建，对外接口遵循 POSIX 标准，具备天然协同基础。同时 openEuler 24.03 LTS SP4 版本集成分布式软总线、K3s 边云协同框架等能力，进一步提升数字基础设施协同能力，构建万物互联的基础。

面向未来，社区将持续创新、社区共建、繁荣生态，夯实数字基座。

夯实云化基座

容器操作系统 KubeOS：云原生场景，实现 OS 容器化部署、运维，提供与业务容器一致的基于 K8S 的管理体验。

安全容器方案：iSulad+shimv2+StratoVirt 安全容器方案，相比传统 Docker+QEMU 方案，底噪和启动时间优化 40%。

机密容器方案：iSulad+Kuasar+secGear 机密容器方案，提供兼容云原生生态的隐私数据保护方案。

机密计算方案：基于 ARM CCA 机密计算架构规范，新增支持网卡、NVME 盘、国密加速卡等外设直通虚机，提供机密虚机内数据和代码的机密性和完整性保护，即使面对拥有特权的基础设施软件或云服务提供商，也能得到有效保护。

全场景

边缘计算：发布面向边缘计算场景的版本，支持 K3s 边云协同框架，具备边云应用统一管理和发放等基础能力。

嵌入式：发布面向嵌入式领域的版本，该版本镜像大小 < 5M，启动时间 < 5s；弹性虚拟化底座以及混合关键性部署框架，支持多核多实例同时部署，跨 OS 通信为不同 OS 之间提供一套基于共享内存的高效通信机制。

AI 原生 OS：OS 使能 AI 软件栈，开箱即用；异构融合内存，调度，训推场景降本增效；智能化交互平台，赋能开发者及管理员。

超节点 OS：OS 基于原有框架扩展支持超节点，分层构建超节点系统软件栈，提供超节点能力和接口，赋能灵衢超节点生态落地和价值释放。

繁荣社区生态

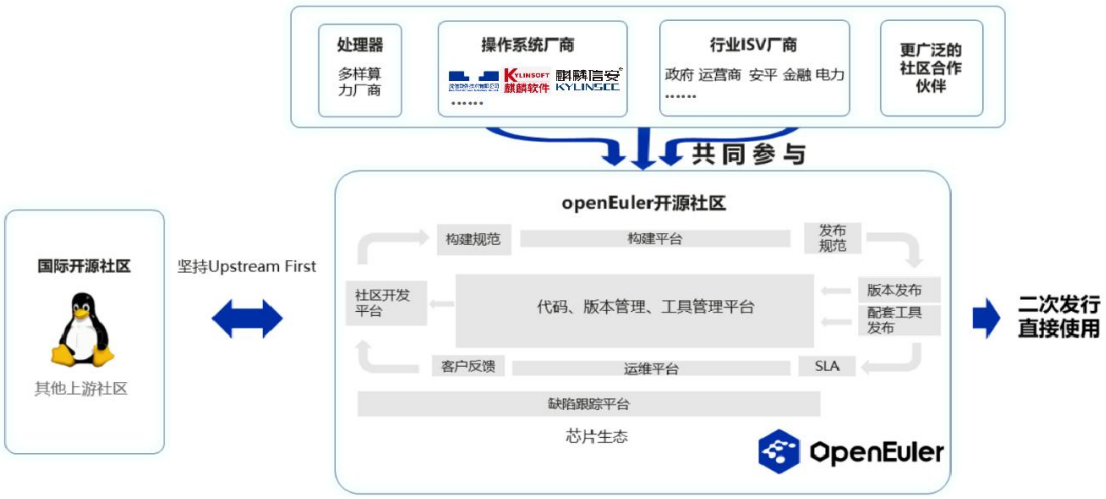
友好桌面环境：UKUI、DDE 、Kiran-desktop，丰富社区桌面环境生态。

openEuler DevKit：支持操作系统迁移、兼容性评估、简化安全配置 secPaver 等更多开发工具。

DevStation 开发者工作站：基于 openEuler 的智能开发者工作站，旨在提供开箱即用、高效安全的开发环境，打通从部署、编码、编译、构建到发布的全流程，通过新增 mcp AI 智能引擎，快速完成社区工具链调用，实现从基础设施搭建到应用开发的效率飞跃，深度集成 CVE 智能修复系统。极大提升了安全漏洞的响应与修复效率。

平台框架

openEuler 社区与上下游生态建立连接，构建多样性的社区合作伙伴和协作模式，共同推进版本演进。



硬件支持

openEuler 社区当前已与多个设备厂商建立丰富的南向生态，比如 Intel、AMD 等主流芯片厂商的加入和参与，openEuler 全版本支持 x86、Arm、龙芯、RISC-V 四种架构，并支持多款 CPU 芯片，包括龙芯 3 号 、兆芯开先/开胜系列、Intel Sierra Forest/Granite Rapids、AMD EPYC 4/5 等芯片系列，支持多个硬件厂商发布的多款整机型号、板卡型号，支持网卡、RAID、FC、GPU&AI、DPU、SSD、安全卡七种类型的板卡，具备良好的兼容性。

支持的 CPU 架构如下：

硬件类型	x86_64	arm64	LoongArch	RISC-V
CPU	Intel、AMD、Hygon、兆芯	鲲鹏、飞腾	龙芯	Sophgo、THead 等

全版本支持的硬件型号可在兼容性网站查询兼容性列表。

3. 运行环境

服务器

若需要在物理机环境上安装 openEuler 操作系统，则物理机硬件需要满足以下兼容性和最小硬件要求。

硬件兼容支持请查看 openEuler 兼容性列表：兼容性列表。

部件名称	最小硬件要求
架构	ARM64、x86_64、RISC-V、LoongArch64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

虚拟机

openEuler 安装时，应注意虚拟机的兼容性问题，当前已测试可以兼容的虚拟机及组件如下所示。

1. 以 openEuler 24.03 LTS SP4 为 HostOS，组件版本如下：

- libvirt-9.10.0-34.oe2403sp4
- libvirt-client-9.10.0-34.oe2403sp4
- libvirt-daemon-9.10.0-34.oe2403sp4
- qemu-8.2.0-78.oe2403sp4
- qemu-img-8.2.0-78.oe2403sp4

2. 兼容的虚拟机列表如下：

HostOS	GuestOS(虚拟机)	架构
openEuler 24.03 LTS SP4	Centos 6	x86_64
openEuler 24.03 LTS SP4	Centos 7	aarch64
openEuler 24.03 LTS SP4	Centos 7	x86_64
openEuler 24.03 LTS SP4	Centos 8	aarch64
openEuler 24.03 LTS SP4	Centos 8	x86_64
openEuler 24.03 LTS SP4	Windows Server 2016	x86_64
openEuler 24.03 LTS SP4	Windows Server 2019	x86_64

部件名称	最小虚拟化空间要求
架构	ARM64、x86_64
CPU	2 个 CPU
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

边缘设备

若需要在边缘设备环境上安装 openEuler 操作系统，则边缘设备硬件需要满足以下兼容性和最小硬件要求。

部件名称	最小硬件要求
架构	ARM64、x86_64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

4. 场景创新

AI 场景创新

智能时代，操作系统需要面向 AI 不断演进。一方面，在操作系统开发、部署、运维全流程以 AI 加持，让操作系统更智能；另一方面，openEuler 已支持 ARM，x86，RISC-V 等全部主流通用计算架构，在智能时代，openEuler 也率先支持 NVIDIA、昇腾等主流 AI 处理器，成为使能多样性算力的首选。

OS for AI

开箱易用

openEuler 兼容 NVIDIA、Ascend 等主流算力平台的软件栈，为用户提供高效的开发运行环境。通过将不同 AI 算力平台的软件栈进行容器化封装，即可简化用户部署过程，提供开箱即用的体验。同时，openEuler 也提供丰富的 AI 框架，方便大家快速在 openEuler 上使用 AI 能力。

功能描述

1. openEuler 已兼容 CANN、CUDA 等硬件 SDK，以及 TensorFlow、PyTorch、MindSpore 等相应的 AI 框架软件，支持 AI 应用在 openEuler 上高效开发与运行。update 版本新增如下软件兼容性测试，用户可以直接使用 pip 进行安装：

机器学习/深度学习框架 & 库： FastAI, Jax, PyMC, Stable-Baselines3, Scikit-Image

自然语言处理（NLP）： spaCy, TextBlob, Gensim, Pattern（需要源码编译）

大语言模型（LLM）工具链： LlamaIndex, OpenAI Python, Diffusers

计算机视觉/数据增强： Imgaug, Albumentations, Kornia, Detectron2

自动机器学习 (AutoML) & 超参数优化： Optuna

实验跟踪： Weights & Biases

相似性搜索/近邻搜索库： Annoy, Nmslib, hnswlib

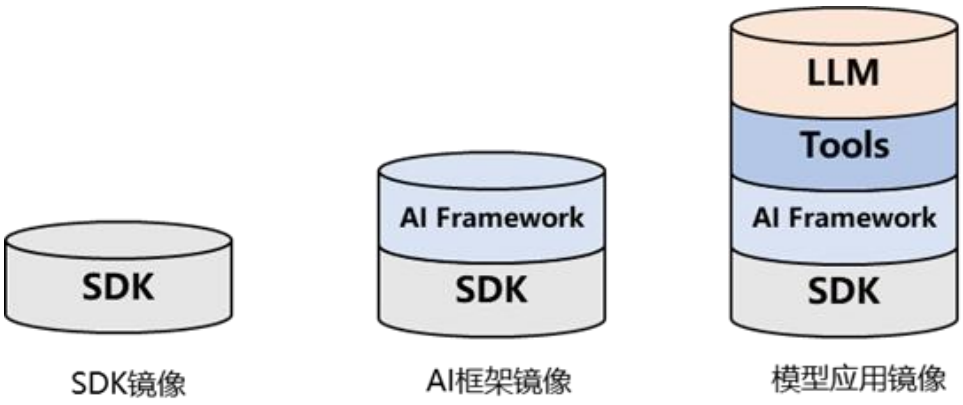
流式学习/在线机器学习： River

并行计算： Dask

数学/符号计算： SymPy

线性规划/优化： PuLP

2. openEuler AI 软件栈容器化封装优化环境部署过程，并面向不同场景提供以下三类容器镜像。



- SDK 镜像：以 openEuler 为基础镜像，安装相应硬件平台的 SDK，如 Ascend 平台的 CANN 或 NVIDIA 的 CUDA 软件。
- AI 框架镜像：以 SDK 镜像为基础，安装 AI 框架软件，如 PyTorch 或 TensorFlow。此外，通过此部分镜像也可快速搭建 AI 分布式场景，如 Ray 等 AI 分布式框架。
- 模型应用镜像：在 AI 框架镜像的基础上，包含完整的工具链和模型应用。

相关使用方式请参考 openEuler AI 容器镜像用户指南。

应用场景

openEuler 使能 AI，向用户提供更多 OS 选择。基于 openEuler 的 AI 容器镜像可以解决开发运行环境部署门槛高的问题，用户根据自身需求选择对应的容器镜像即可一键部署，三类容器镜像的应用场景如下。

- SDK 镜像：提供对应硬件的计算加速工具包和开发环境，用户可进行 Ascend CANN 或 NVIDIA CUDA 等应用的开发和调试。同时，可在该类容器中运行高性能计算任务，例如大规模数据处理、并行计算等。
- AI 框架镜像：用户可直接在该类容器中进行 AI 模型开发、训练及推理等任务。
- 模型应用镜像：已预置完整的 AI 软件栈和特定的模型，用户可根据自身需求选择相应的模型应用镜像来开展模型推理或微调任务。

XPU Turbo 大语言模型异构协同加速运行时

功能描述

XPU Turbo 大语言模型异构协同加速运行时专注于单机多卡环境下大模型推理任务的性能提升，针对鲲鹏+XPU（GPU、NPU 等）的异构算力协同，显著提升大模型的吞吐量和并发量：

- CPU 推理加速：通过 NUMA 亲和调度、矩阵运算并行加速、SVE 指令集推理算子适配等方式，提升 CPU 的吞吐量。
- 异构融合调度：XPU Turbo 支持 PD 分离与 AF 分离两种调度模式。在 PD 分离模式下 XPU Turbo 支持在 GPU 侧任务满载时，动态将推理请求的 prefill 阶段在 GPU 上执行，decode 阶段放在 CPU 上执行。在 AF 分离模式下 XPU Turbo 会将 FFN 过程放在 CPU 上执行。

应用场景

XPU Turbo 大语言模型推理优化方案当前支持 DeepSeek、Qwen 等 transformer 架构的模型。其中，CPU 推理加速能力已完成了对 DeepSeek 7B、14B、32B 以及 Qwen2.5、Qwen3、Qwen3-MOE 系列模型的适配。目前支持的硬件包括 Nvidia、昇腾等主流硬件。此外，在开启 AF 分离功能后，XPU Turbo 支持在单张 NPU 卡上进行 Qwen3-235B-A22B 模型的推理。XPU Turbo 主要适用于以下典型场景：

- 数据中心场景：XPU Turbo 通过上述技术，利用 CPU 填充推理任务，充分利用 CPU 资源，增加大模型并发量与吞吐量。

XPU Turbo 使用方式请参考 XPU Turbo 部署指南。

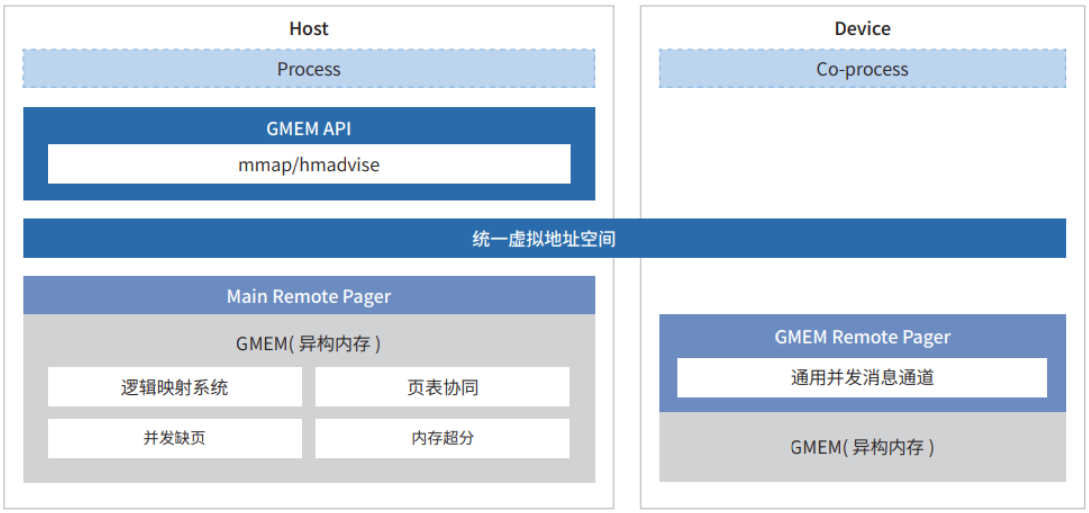
异构融合 GMem

在后摩尔时代，GPU、TPU 和 FPGA 等专用异构加速器设备正不断涌现，它们与 CPU 类似，需要将数据放在本地内存（例如 LPDDR 或 HBM）中以提高计算速度。加速器厂商们也不可避免地需要开发复杂的内存管理系统。 现行加速器内存管理方案存在诸多缺陷:

- CPU 侧内存管理与加速器侧分离，数据显式搬移，加速器内存管理的易用性和性能难以平衡。
- 大模型场景下加速器设备 HBM 内存（Hign BandWidth Memory）严重不足，现有的手动 swap 方案性能损耗大且 通用性差。
- 搜推、大数据场景存在大量无效数据搬移，缺少高效内存池化方案。

Linux 现有的 HMM 框架，编程复杂度高且依赖人工调优，性能和可移植性差，引发 OS 社区反弹，最终导致 HMM 方案搁浅。异构加速器领域亟需高效的统一内存管理机制。异构通用内存管理框架 GMem （Generalized Memory Management），提供了异构内存互联的中心化管理机制，且 GMem API 与 Linux 原生内存管理 API 保持统一，易用性强，性能与可移植性好。加速器使用 GMem API 将内存接入统一地址空间后，可自动获得 GMem 面向异构内存编程优化的能力。与此同时，加速器驱动无需重复实现内存管理框架，大幅降低开发维护带来的成本。开发者使用一套统一申请、释放的 API，即可完成异构内存编程，无需处理内存搬移等细节。在加速器 HBM 内存不足时，GMem 可将 CPU 内存作为加速器缓存，透明地超分 HBM，无需应用手动 swap。GMem 提供高效免搬移的内存池化方案，当内存池以共享方式接入后，可解决数据反复搬移的痛点。

功能描述



GMem 革新了 Linux 内核中的内存管理架构，其中逻辑映射系统屏蔽了 CPU 和加速器地址访问差异，remote_pager 内存消息交互框架提供了设备接入抽象层。在统一的地址空间下，GMem 可以在数据需要被访问或换页时，自动地迁移数据到 OS 或加速器端。

- **异构内存特性**

为了结合加速器算力与 CPU 通用算力，实现统一的内存管理和透明内存访问，GMem 设计了统一虚拟内存地址空间机制，将原本的 OS 与加速器并行的两套地址空间合并为统一虚拟地址空间。

GMem 建立了一套新的逻辑页表去维护这个统一虚拟地址空间，通过利用逻辑页表的信息，可以维护不同处理器、不同微架构间多份页表的一致性。基于逻辑页表的访存一致性机制，内存访问时，通过内核缺页流程即可将待访问内存在主机与加速器进行搬移。在实际使用时，加速器可在内存不足时可以借用主机内存，同时回收加速器内的冷内存，达到内存超分的效果，突破模型参数受限于加速器内存的限制，实现低成本的大模型训练。

- **分级内存管理**

通过在内核中提供 GMem 高层 API，允许加速器驱动通过注册 GMem 规范所定义的 MMU 函数直接获取内存管理功能， 建立逻辑页表并进行内存超分。逻辑页表将内存管理的高层逻辑与 CPU 的硬件相关层解耦，从而抽象出能让各类加速器复用的高层内存管理逻辑。加速器只需要注册底层函数，不再需要实现任何统一地址空间协同的高层逻辑。 由系统自动管理数据在不同存储介质间的迁移，例如 dram 和 hbm。

- **remote Pager 内存消息交互框架**

Remote Pager 作为 OS 内核外延的内存管理框架，设计并实现了主机和加速器设备之间协作的消息通道、进程管理、 内存交换和内存预取等模块，由独立驱动 remote_pager.ko 使能。通过 Remote Pager 抽象层可以让第三方加速器很容易地接入 GMem 系统，简化设备适配难度。

- **用户 API**

用户可以直接使用 OS 的 mmap 分配统一虚拟内存，GMem 在 mmap 系统调用中新增分配统一虚拟内存的标志（MMAP_PEER_SHARED）。

同时 libgmem 用户态库提供了内存预取语义 hmadvise 接口，协助用户优化加速器内存访问效率（参考 https://atomgit.com/openeuler/libgmem）。

- **约束限制**

硬件约束：需要使用昇腾 npu 卡。

软件约束：目前仅支持 2M 大页，所以 host OS 以及 NPU 卡内 OS 的透明大页需要默认开启。 通过 MAP_PEER_SHARED 申请的异构内存目前不支持 fork 时继承。

GMem 使用方法可参考以下链接：

https://atomgit.com/openeuler/docs-centralized/tree/master/docs/zh/docs/GMEM

应用场景

- 异构统一内存编程

在面向异构内存编程时，使用 GMem 可分配 CPU 和加速器之间的统一虚拟内存，CPU 内存与加速器内存可共享一个指针，显著降低了异构编程复杂度。当前基于 NPU 试点，驱动仅需百行修改即可接入 GMem，替换原有约 4000 行内存管理框架代码。

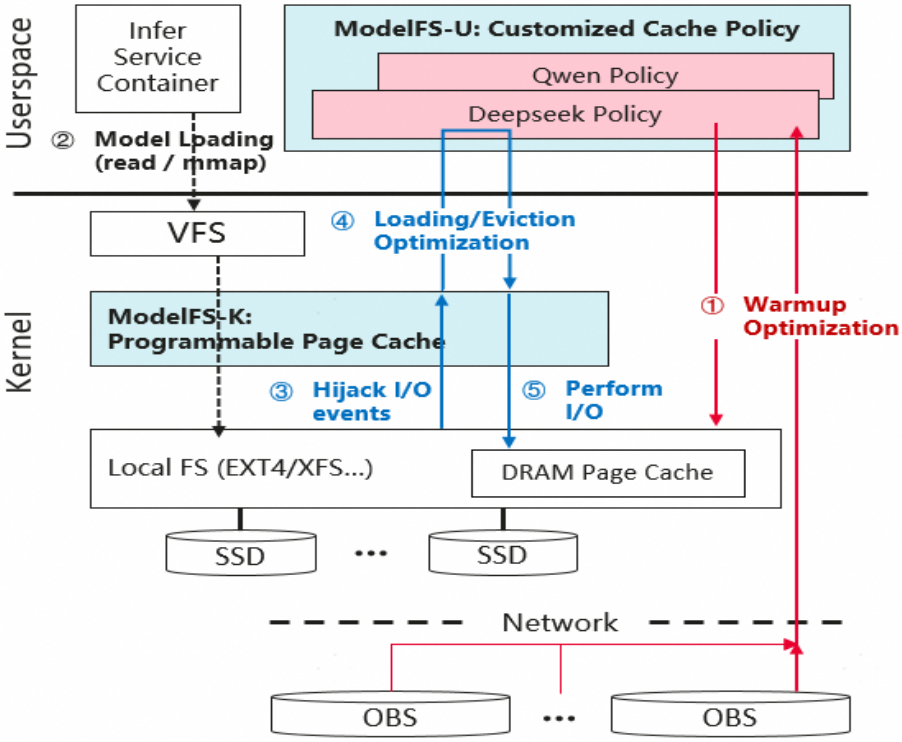
- 加速器内存自动超分

使用 GMem 接口分配内存时，将不受加速器的物理内存容量所限制，应用可以透明地超分内存（当前上限为 CPU 的 DRAM 容量）。GMem 将较冷的设备内存页换出到 CPU 内存上，拓展了应用处理的问题规模，实现高性能、低门槛训推。 通过 GMem 提供的极简异构内存管理框架，在超大模型训练中，GMem 性能领先 NVIDIA-UVM。随着内存使用量增长，领先比例不断提升，在超分两倍以上时可领先 NVIDIA-UVM 60% 以上（数据基于 NPU-Ascend910 与 GPU-A100 硬件，在 相同 HBM 内存条件下测试）。

ModelFS 模型启动加速

ModelFS 针对大语言模型（LLM）推理启动阶段的模型加载瓶颈进行优化。 现有相关工作主要通过优化推理框架来提升模型加载性能，这种方法往往以牺牲兼容性为代价。然而，基于我们的工业实践经验，兼容性是决定一项技术能否在实际场景中广泛应用的关键因素。本工作在保证强兼容性的前提下，通过优化文件系统的缓存策略，实现了模型加载性能的领先水平。 ModelFS 在内核中设计了一个非侵入式、灵活且轻量级的可编程页缓存框架，允许用户自定义文件系统的页缓存策略。 基于可编程页缓存，我们进一步设计了面向模型加载优化的缓存策略的参考实现。

功能描述



ModelFS-K：

ModelFS 的内核模块，提供文件系统（FS）页缓存可编程框架。核心设计是一个堆叠 FS，可以挂载到现有 FS 之上，ModelFS-K 将底层文件系统的相关逻辑用 UPC（用户态调用）重定向到用户态实现的 prefetch 和 evict 函数。

ModelFS-U：

ModelFS 的用户态模块，提供缓存策略的运行时。提供一套 VFS-like 的用户态编程框架，模型实现者/IO 优化者可以根据模型的加载 IO 特性自定义缓存策略。用户需实现 init()，exit()， prefetch()，evict()，ModelFS-U 会将它们注册到 ModelFS-K 中，并在运行过程中负责解析 IO 事件和进行异步数据预取/淘汰。

应用场景

ModelFS 可用于多种类型存储后端的 IO 加速，包括本地 FS，分布式 FS，对象存储（OBS）等，以下是两种典型的用法。

1) 远端 OBS 存储权重预热：对于模型权重存储于远端 OBS 的情况，由于 OBS 带宽低，推理系统往往会将热点权重提前预热到本地存储。ModelFS 可以通过实现 IO 聚合的缓存策略，将多个 OBS 桶的带宽聚合来加速权重预热。

2) 本地 SSD 权重加载加速：对于模型权重存储在本地 SSD 的情况，性能瓶颈在于本地 SSD 带宽利用率低。ModelFS 实现了一种基于 I/O 模板的机制，在第一次运行时追踪每个推理服务的 IO 加载行为，而在后续推理启动过程中可以精确感知未来的 IO 行为，进而充分利用 SSD 带宽、XPU 亲和性以提升预取与淘汰机制的效率。

NPU 算力切分

在计算领域，计算能力是推动 AI 技术发展的驱动力之一，为了提升 AI 运算速度，各大厂商纷纷推出了自己的 AI 计算设备。但多数 AI 卡缺乏优先级、公平性、抢占、算力带宽控制等高级调度能力，无法满足中小 AI 模型多任务混部场景的诉求，比如在线-在线推理混部，在线-离线推理混部等。中小 AI 模型独占部署，导致 AI 卡资源利用率低，造成严重的资源浪费。

功能描述

xSched 是面向中小模型的通用的调度框架，能够为 NPU 卡提供基础的调度机制，包括任务抢占、时间片切分、组调度、算力带宽管控、显存容量管控等，并构建公平调度策略、实时调度策略等多种调度策略，满足不同 AI 场景调度算法的诉求。

应用场景

1) 在多推理任务混部场景（AI 任务在线-在线混部），提供按需分配单卡算力的能力，解决 NPU 按卡粒度分配资源导致过分配的问题；

2) 在多训推混部或在离线推理混部场景（AI 任务在离线混部），实现推理任务快速抢占离线推理或训练任务，保障高优先级任务性能不劣化的前提下，提升 NPU 卡资源利用率。

故障快恢

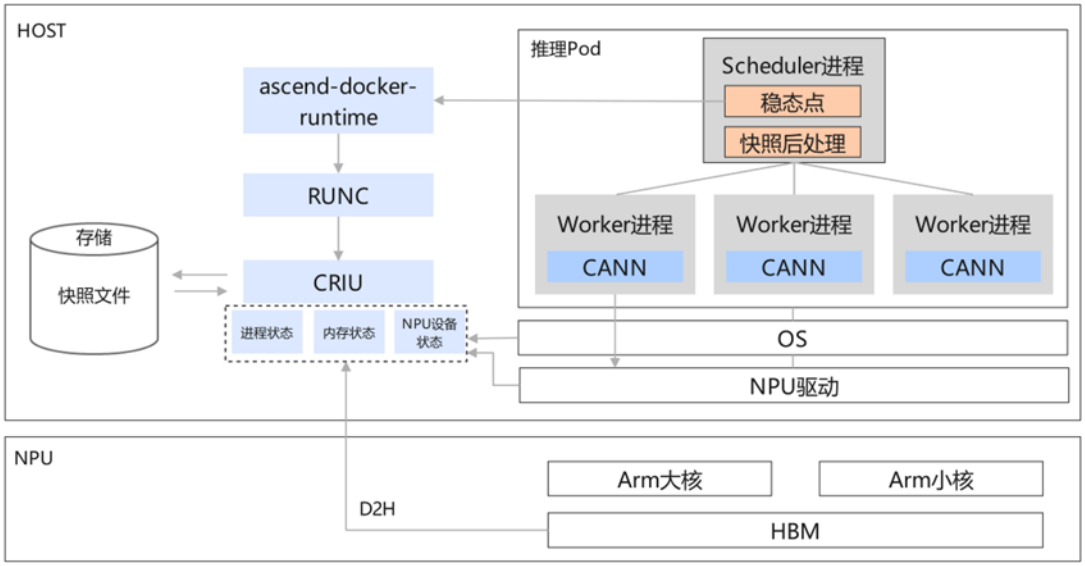
在当今的 AI 推理生产环境中，推理服务通常以容器化形式部署并长时间运行，然而，运行过程中的故障是不可避免的，传统的故障恢复路径往往需要数分钟甚至更长时间，会导致业务中断过久，造成了资源的浪费。

功能描述

为 AI 推理容器提供秒级故障快速恢复能力，将恢复时间从分钟级降低到秒级，基于支持进程状态 dump/restore 能力的组件（如 CRIU），支持 npu 驱动的状态的导入导出，并实现与 k8s 生态的对接，实现推理容器的秒级恢复，具体架构如图所示。

快照流程如下：推理 Pod 中的推理框架发起快照，先调用 NPU 驱动快照接口，将 NPU 侧内存及状态保存并 D2H 传输到 Host 侧，再调用 Host 侧快照接口，最终调用到 CRIU，进行打快照，保存推理 Pod 在 Host 侧所有状态。

恢复流程如下：由 k8s 触发恢复 pod，CRIU 会对推理 Pod 及其 Host 侧状态进行恢复，然后调用 NPU 驱动恢复接口，进行 NPU 内存及状态的恢复。



应用场景

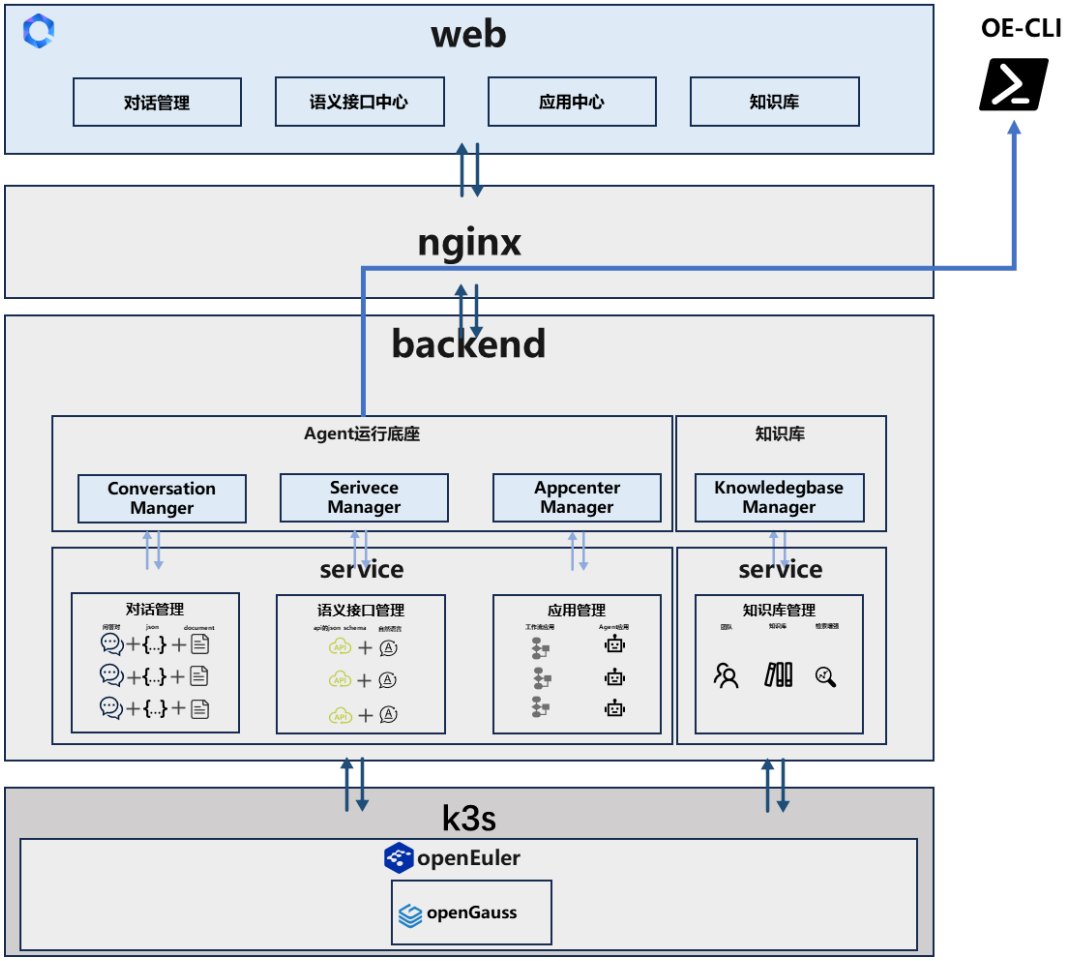
容器快照可以用于 AI 推理容器的故障快恢，弹性扩缩容等场景。

故障快恢：在推理场景下，通过容器快照能力，可以快速通过快照恢复推理容器，业务中断时间短;

弹性扩缩容：通过极致的快照启动速度，结合 k8s，实现推理容器的弹性扩缩容能力。

AI for OS

当前，openEuler 和 AI 深度结合，一方面，基于 openEuler 操作系统，研发出了 Witty，初步实现基于知识库的智能问答、基于语义接口的工作流编排、基于 MCP 的 Agent 构建等功能，在此基础上，Witty 集成了部分系统服务，让 openEuler 更智能。



智能问答

功能描述

Witty 目前支持 Web 和智能 Shell 两个入口。

- Web 入口：用户可以通过 Web 入口以可视化的形式进行知识库的构建和使用、知识库准确率的自动化测试与评估结果获取、openapi 形式的语义接口注册、基于语义接口的工作流应用的构建和使用、mcp 的注册安装和激活和基于 mcp 的 Agent 应用的构建和使用，Web 入口便携了新手用户对 openEuler 知识的获取和对 openEuler AI 能力的使用。
- 智能 Shell 入口：用户可以通过 Shell 入口调用智能体框架中的智能问答或者是预集成的 Agent 应用，Shell 入口便携了运维人员对操作系统的使用，能通过亲和操作系统的方式与 openEuler 进行智能交互，降低运维成本。

智能规划、调度和推荐

- 智能规划：Witty 的 Agent 应用可以基于用户的输入和当前可用的工具实时规划运行步骤，直至完成用户目标或者达到步骤执行上限。

- 智能调度：Witty 支持用户在一个工作流应用中定义多个工作流，基于用户的查询，Witty 会自动的提取参数且选择最为合适的工作流进行工作。
- 智能推荐：Witty 基于用户的查询和工作流的运行结果，推荐用户接下来可能会使用的工作流，增加任务的完成概率，简便应用的使用。

工作流应用

- 语义接口：语义接口是指含有自然语言注释的接口形式，Witty 提供了两种语义接口注册方式：首先，Witty 允许用户将 api 接口以 openapi（3.0+）yaml 文件形式注册到系统中，用户需要在编写 openapi yaml 文件时，对接口添加自然语言注释，后续在这些接口调用过程中，大模型会根据接口的注释对接口进行选择和填参；其次，Witty 也允许用户将功能以 python 编码的形式注册到 Witty 中，这种形式对于 Witty 而言更为亲和。以上两种形式产生的语义接口最终都可以通过可视化形式编排为工作流。
- 工作流编排及调用：Witty 允许用户将系统提供的语义接口以及用户注册的语义接口以可视化的形式连线成工作流，并支持用户对工作流进行调试且以应用的形式进行发布和使用，工作流在调试和使用过程中会展示中间结果，降低用户调试成本，提升用户交互体验。

Agent 应用

- mcp 注册、安装和激活：mcp 是当前比较主流的一种 AI 相关协议，它支持用 sdk 将复杂多样服务统一封装，带有天然的语义信息，并支持 AI 对基于 mcp 改造后服务下的工具进行比较便捷的调用。
- Agent 构建和应用：当前 Witty 支持以 mcp 和不同的大模型结合构建 Agent，这些构建完成的 Agent 可以基于配置的大模型信息以及后续用户输入的目标，将用户的目标拆解成阶段性需求，最后使用 mcp 服务下的工具将阶段性需求完成，直至达成用户的目标。

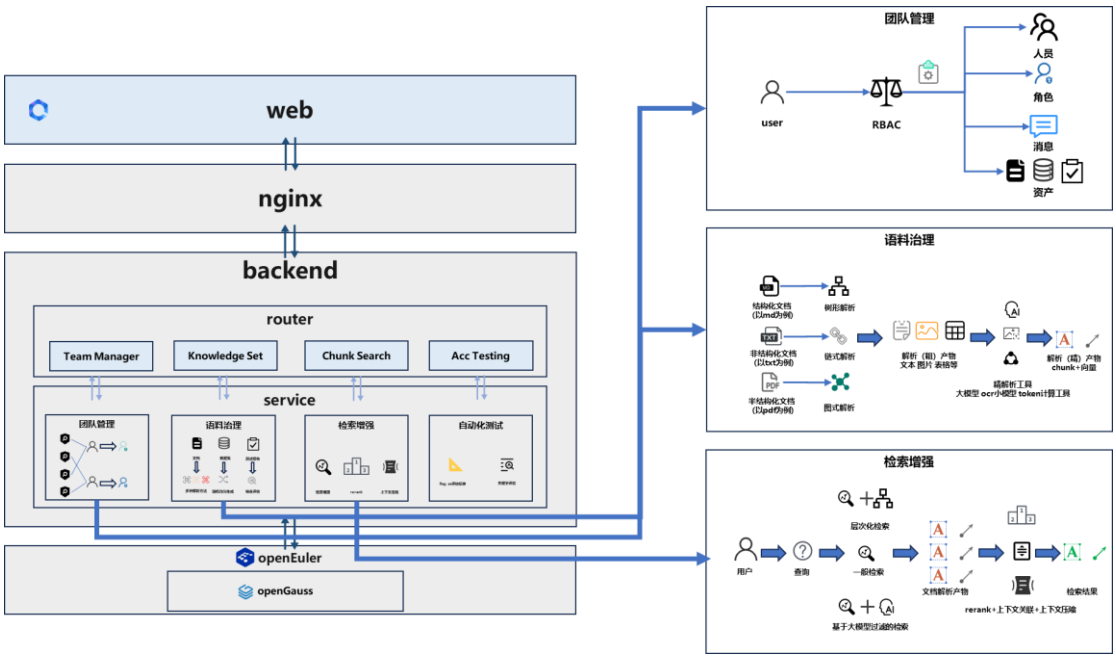
RAG

RAG(检索增强技术)是为了增强大模型长期记忆能力和降低大模型训练成本诞生的技术，相较传统 RAG，Witty 中的 RAG 技术在检索前处理、知识索引、检索增强算法和检索后处理方面做了改进：

- **检索前处理**：对于信息缺失的用户查询，基于历史上下文和用户意图，提供查询改写的能力，提升检索准确率；
- **知识索引**：对于格式和内容多样化的文档，提供摘要、文本特征提取、树形解析、ocr 等文档解析能力，建立片段特征索引，提升检索增强命中率；

- **检索增强算法**：对于多样化的检索场景，提供 8 种检索方法，其中基于动态关键字权重的检索方法和基于大模型过滤的检索方法能极大提升开箱及用的准确率；
- **检索后处理**：对于上下文缺失的片段，提供均匀随机化上下文补全的方法，增加片段信息完整度，降低最终模型拟合幻想程度；对于 token 阈值上限的片段（集合），提供基于杰卡德距离的 rerank 方法、基于通用词去除及随机丢弃的 token 压缩方法和基于二分的 token 截断方法，允许一些资源受限的场景也能正常进行智能问答。

通过以上能力，相较传统的 RAG 技术，Witty 中的 RAG 技术能适应多种文档格式和内容场景，在不为系统增加较大负担的情况下，增强问答服务体验。



团队管理

团队管理是 Witty 中的 RAG 技术的基础能力之一，其通 RBAC 的机制来管控团队内成员对角色、成员和资产的访问，以增强知识库整体的易用性：

对于角色：涉及对角色原子权限构成的编辑、角色的增删改和成员角色的赋予，这些能力的原子化是实现团队权限实例化的核心。

对于成员：涉及对成员的邀请和申请请求的处理以及成员的增删改查，这些能力的原子化是实现管理成员进组及其权限赋予的核心。

对于资产：涉及对资产库和资产库内的文档、数据集和测试结果的增删改查，这些能力的原子化是实现资产数据按需访问的核心。

语料治理

语料治理是 Witty 中的 RAG 技术的基础能力之一，其通过上下文位置信息提取、文本摘要和 OCR 增强等方式将语料以合适形态入库，以增强用户查询命中期望文档的概率：

- 上下文位置信息提取：对于文档内容，留存文档相对位置关系，包过片段的全局相对偏移和局部相对偏移，为上下文补全提供基础数据；
- 文本摘要：对复杂文档或者片段，通过滑动窗口+大模型对文本进行摘要，为后续多层次检索的方式提供基础数据；
- OCR 增强：对图文混合的文档，基于图片文字内容+图片上下文进行摘要，为后续针对图片的提问提供基础数据；
- 文档溯源：Witty 在生成答案的过程中，基于返回的片段和相关的文档信息，在对应的句子右下方生成角注，点击角注可以获取文档的名称和摘要，增加问答生成的可信度。

自动化测试

自动化测试是 Witty 中的 RAG 技术的基础能力之一，其通过自动化数据集生成和测试评估识别知识库和检索增强算法等配置的不足：

- 数据集生成：用户选择解析完成的文档，基于用户选择的文档，随机选择部分解析结果，并将这些解析结果发送给大模型，让大模型生成及过滤问答对，最终形成含有查询、标准答案和原始片段的高质量测试数据集；
- 测试评估：基于用户生成的数据集和配置的检索增强算法、大模型、topk 片段限制等参数展开测试，最终基于测试集中的查询、标准答案、原始片段、关联到的片段和大模型拟合结果进行打分，最终使用精确率、召回率、忠实值、可解释性、最长公共子串得分、编辑距离得分和杰卡德相似系数对测试结果进行评估。

应用场景

- 面向 openEuler 普通用户：基于 openEuler 社区文献例如白皮书或者用户本地文档，可以构建定制化智能助手。
- 面向 openEuler 开发者：基于 web 端，开发者可以构建自己的工作流或者 Agent 应用，降低一些场景（代码审计等）重复劳动。
- 面向 openEuler 运维人员：基于 shell 端，运维人员可以通过自然语言与 os 交互，降低复杂命令构造和修改成本。

相关使用方式请参考 openEuler-Witty-专区

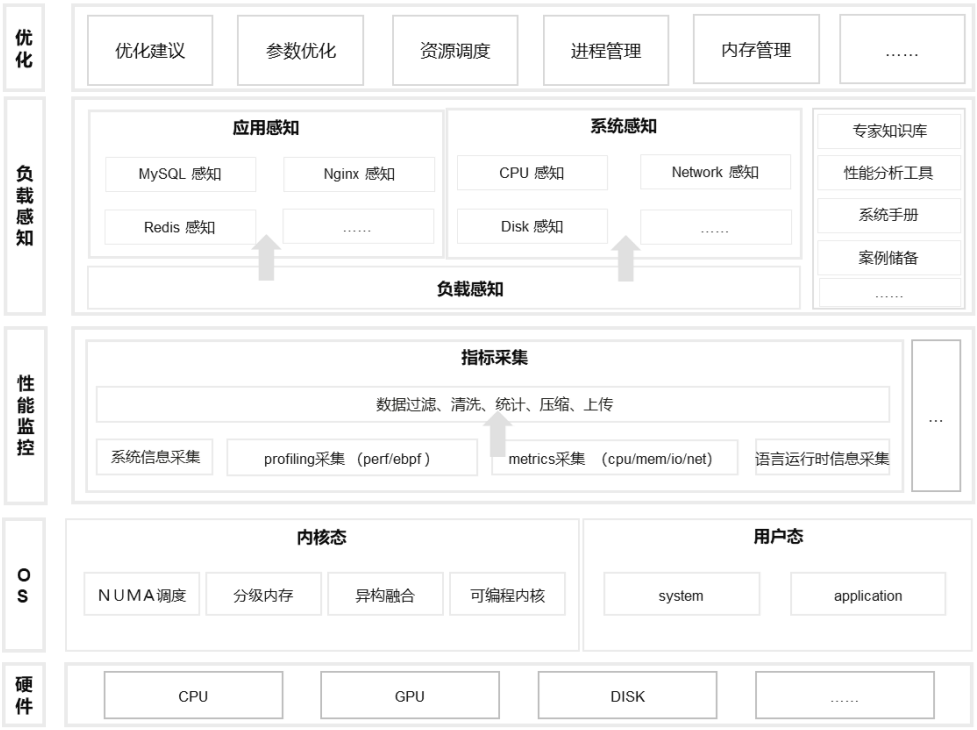
智能调优

功能描述

基于 LLM 的系统参数寻优

Witty 智能调优功能目前支持智能 shell 入口。

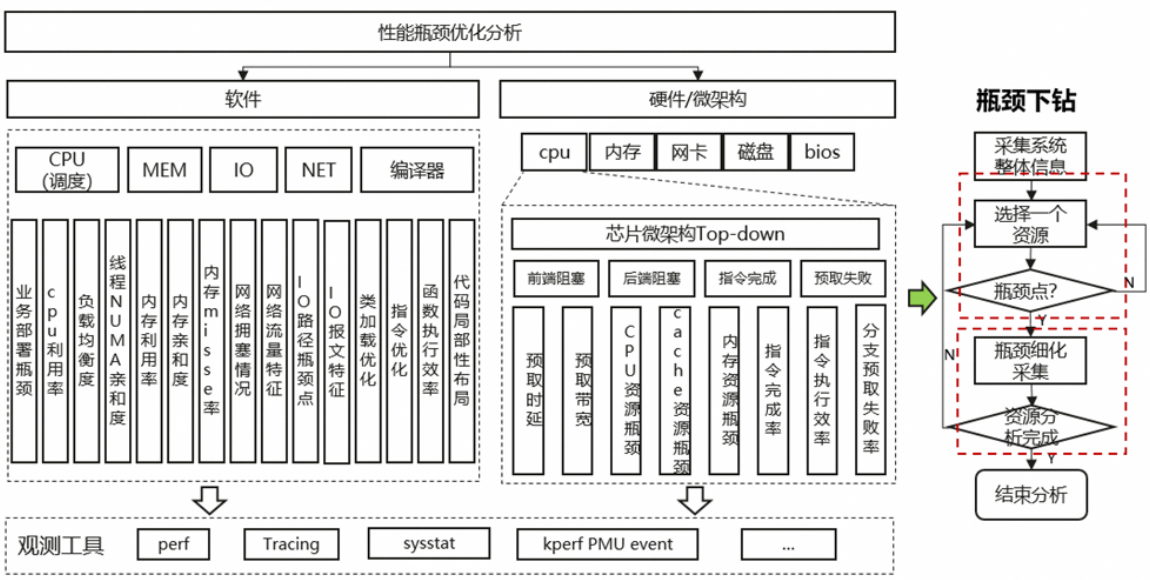
在上述功能入口，用户可通过与 Witty 进行自然语言交互，完成性能数据采集、系统性能分析、系统性能优化等作业，实现启发式调优，已支持通过 MCP 协议实现调优意图识别。



基于 SKILL 的 TopDown 性能瓶颈自主识别

自上而下深入定位性能瓶颈点，基于「证据链」进行瓶颈分析：观察到粗粒度瓶颈指标→分析多个可能原因→细化收集→进一步确认证据，结合系统实际状态构建瓶颈链条；根据 SKILL 中提供的关键指标指引，覆盖多类常见 OS 层性能瓶颈，以提供针对性优化建议。

对接多种 OS 层原生性能分析接口及工具，多层次采集并 TopDown 定位性能瓶颈，支持下钻识别调度/锁/IO/内存/网络 5 大类细化瓶颈。

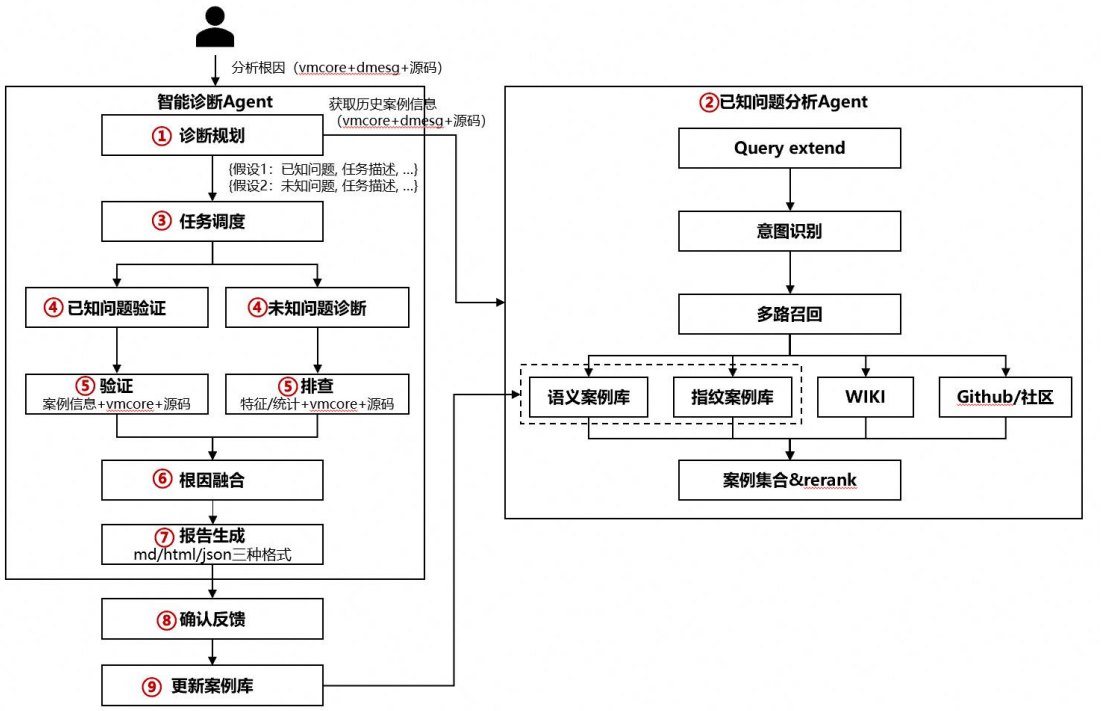


应用场景

- 快速获取系统重要性能指标数据：可快速获取当前系统中 CPU/IO/DISK/NETWORK/微架构等多个重要维度的性能指标以及指定应用的性能指标，帮助用户快速了解系统性能数据。
- 分析系统性能状况：可生成性能分析报告，报告从 CPU/IO/DISK/NETWORK/微架构等多个重要维度分析系统性能状况以及分析指定应用的性能状况，并提示当前系统可能存在的性能瓶颈。
- 推荐系统性能优化建议：可针对当前系统性能情况推荐一组合适的应用或系统参数，并自动使能推荐参数配置，可生成一键式执行的性能优化脚本，用户在审核脚本内容后，可执行该脚本，对系统及指定应用的配置进行优化。

智能诊断

系统宕机智能诊断



功能描述

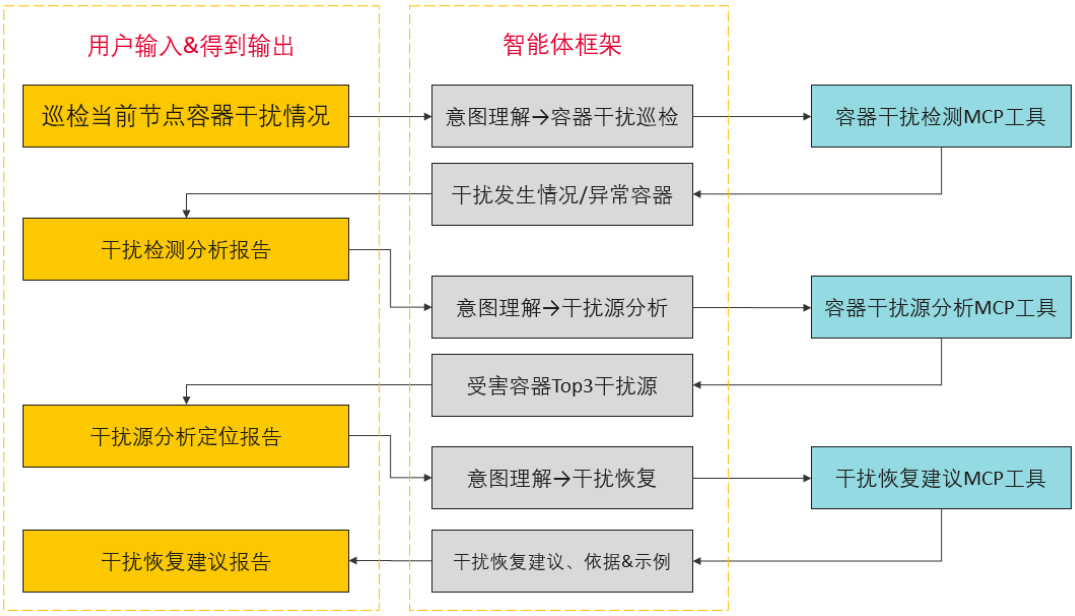
1. 智能诊断 Agent：采用假设生成-并行验证的故障排查范式，基于 dmesg 日志、内核源码、故障描述多维度原始输入，调用已知问题分析 Agent 检索存量历史案例，同步生成已知故障、未知故障两类假设并行分析：针对已知故障假设，比对当前现场数据与历史案例的分析流程、结论特征完成匹配核验；针对未知故障假设，自动调用 crash 工具结合源码解析推导底层根因，最终融合两条链路分析结论，自动输出 HTML、Markdown、JSON 三类诊断报告，报告完整涵盖根因结论、故障时间线、故障传播链、因果逻辑链、完整排查步骤及修复方案。
2. 已知问题 Agent：内置日志智能解析引擎，自动抽取日志中的核心异常特征，包含报错码、异常告警、内核堆栈、寄存器信息等关键线索；融合轻量化 RAG 检索与多策略混合搜索算法，跨语义案例库、故障指纹库、知识库 Wiki、GitHub 开源社区做精准匹配，快速召回同类型历史故障参考案例，为全流程故障排查提供前置高效依据。
3. 案例沉淀：闭环承接已核验、用户确认生效的完整诊断报告、原始故障场景描述、现场原始数据特征，自动化标准化转换为结构化历史案例入库至语义案例库；持续

扩充高质量故障样本，构建数据自迭代飞轮，持续优化检索匹配精度、假设生成准确率，达成工具越使用、故障定位越快速精准的正向迭代效果。

应用场景

线上业务服务器发生 OOM、空指针、死锁、软死锁、硬件异常等内核崩溃并生成 vmcore 转储文件后，运维人员无需手动操作 crash 工具、逐行梳理海量日志，上传转储文件、系统日志与故障现象后，通过智能诊断 Agent+已知问题分析 Agent 组合即可自动完成全链路诊断，替代传统数小时人工调试，快速支撑应急止损。

容器干扰检测



功能描述

1. 干扰检测巡检：调用容器干扰检测工具，对节点中存在监控的节点在指定的时间段内的干扰进行检测，报告节点中存在的干扰情况，报告存在干扰的容器和对应指标。
2. 干扰源分析：调用容器干扰源分析工具，对节点中存在的所有受干扰容器进行分析，报告各个受干扰容器的 Top3 干扰源容器和对应的关联指标（如 CPU run delay）。
3. 干扰恢复建议生成：根据干扰源分析定位结果，调用干扰恢复建议工具，对存在的干扰进行分析并给出干扰恢复建议报告，提供各条建议的依据和示例指令。

应用场景

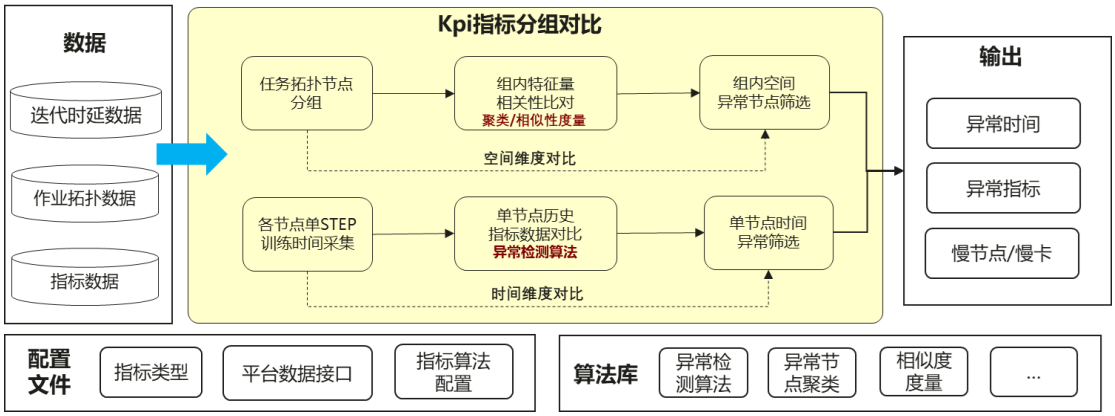
智能诊断接口在本次 openEuler 24.03 LTS SP4 版本的功能范围是：容器干扰检测、干扰源分析、干扰恢复建议生成。

- 其中容器干扰检测指的是：对单机上的已有监控的容器指标进行干扰检测，报告受干扰容器。
- 其中干扰源分析指的是：结合干扰检测结果和节点中的监控指标，进行干扰源分析，输出各个受干扰容器的 top3 根因指标。
- 其中干扰恢复建议生成指的是：基于干扰源分析结果，生成对应干扰恢复建议报告。

AI 集群慢节点定界

AI 集群慢节点定位需要实时性和智能性，当前 Agent 基于 sysTrace 提供的时序分析算法对 AI 集群训练任务下的慢节点/慢卡进行检测，对比传统的分析方法具有更高的准确率和泛用性，并且用户可以通过自然语言的形式让 Agent 自主的对特定集群或集群的特定子集进行诊断和分析，直至定位慢节点。

功能描述



sysTrace 的核心能力是时序分析技术，该技术主要功能如下：

- 配置文件：主要包括待观测指标类型、指标算法配置参数以及数据接口，用于初始化慢节点检测算法。
- 算法库：包括常用的时序异常检测算法 dbscan 算法，k-sigma 算法，异常节点聚类算法和相似度度量算法。
- 数据：采集到的各个节点的指标数据，包括 python 堆栈信息、io 相关信息、hccl 算子活动跟踪数据。

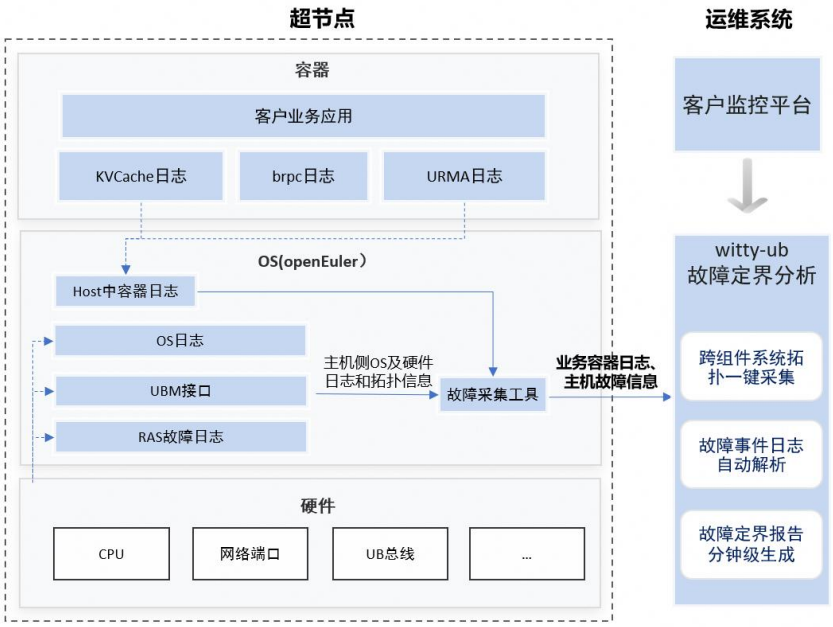
- 指标分组对比：包括组内空间异常节点筛选和单节点时间异常筛选。组内空间异常节点筛选根据异常聚类算法输出异常节点；单节点时间异常筛选根据单节点历史数据进行时序异常检测判断节点是否异常。

应用场景

sysTrace 支持对 XPU 的采集数据进行慢卡慢节点异常检测，告警展示。

AI 模型训练场景：适用于 AI 模型分布式训练任务，通过该功能可快速发现慢卡慢节点，便于系统自愈或者运维人员修复问题。

超节点通断&时延问题定界



功能描述

UB 超节点故障发生后，根据日志、指标等观测数据，推断故障发生的组件，对于明确故障处理参与方、加速故障恢复具有重要的意义。本功能面向超节点 KVCache 和 URMA 通信时延和通断故障定界场景，提供了以下能力：

1、跨组件系统拓扑一键采集：基于 UBM 调测接口、组件日志等提供的信息，一键采集通信资源拓扑，汇聚全局拓扑信息，用于故障影响面分析，支撑故障定界。

2、故障事件日志自动解析：以事件为核心的通信故障日志解析，自动提取故障元信息、整合上下游组件日志，大幅降低待分析日志量，聚焦关键故障信息。

3、故障定界报告分钟级生成：基于拓扑采集和日志解析结果识别通信时延和通断类故障，集成超节点故障案例库和组件知识，分钟级自动生成定界报告，支撑故障快速定界。

应用场景

本功能面向超节点 KVCache 和 URMA 通信场景，为超节点系统运维人员提供时延和通断故障定界能力。该功能支持作为独立工具使用和接入外部运维系统两种应用模式：

1、作为独立工具使用：提供完整、易用的图形化用户操作界面，用户启动前后端服务，即可在界面内完成数据上传、查询、分析过程。

2、接入外部运维系统：提供符合 RESTful 规范的北向接口，用户启动后端服务后可调用接口使用主要功能，以接入外部运维系统。

智能容器镜像拉取

功能描述

Witty 目前支持通过自然语言调用环境资源，在本地协助用户基于实际物理资源拉取容器镜像，并且建立适合算力设备调试的开发环境。

当前版本支持三类容器，并且镜像源已同步在 dockerhub 发布，用户可手动拉取运行：

1. SDK 层：仅封装使能 AI 硬件资源的组件库，例如：cuda、cann 等。
2. SDK + 训练/推理框架：在 SDK 层的基础上加装 tensorflow、pytorch 等框架，例如：tensorflow2.15.0-cuda12.2.0、pytorch2.1.0.a1-cann7.0.RC1 等。
3. SDK + 训练/推理框架 + 大模型：在第 2 类容器上选配几个模型进行封装，例如 llama2-7b、chatglm2-13b 等语言模型。

当前支持的容器镜像汇总：

registry	repository	image_name	tag
docker.io	openeuler	cann	8.0.RC1-oe2203sp4
			cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	oneapi-runtime	2024.2.0-oe2403lts
docker.io	openeuler	oneapi-basekit	2024.2.0-oe2403lts
docker.io	openeuler	llm-server	1.0.0-oe2203sp3
docker.io	openeuler	mlflow	2.11.1-oe2203sp3
			2.13.1-oe2203sp3
docker.io	openeuler	llm	chatglm2_6b-pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2

			llama2-7b-q8_0-oe2203sp2
			chatglm2-6b-q8_0-oe2203sp2
			fastchat-pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	tensorflow	tensorflow2.15.0-oe2203sp2
			tensorflow2.15.0-cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2
docker.io	openeuler	pytorch	pytorch2.1.0-oe2203sp2
			pytorch2.1.0-cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2
			pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	cuda	cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2

应用场景

- 面向 openEuler 普通用户：简化深度学习开发环境构建流程，节省物理资源的调用前提，比如实现在 openEuler 系统上搭建昇腾计算的开发环境。
- 面向 openEuler 开发者：熟悉 openEulerAI 软件栈，减少组件配套试错成本。

超节点场景创新

算力需求指数级增长与高速互联技术突破，驱动硬件从单节点向超节点加速演进，未来数据类、资源类、业务类的核心诉求从单机性能渐变到超节点，具有以下特点：

- 资源池化：计算、内存、互联和存储等均可池化，支持任意多对多协同。
- 规模扩展：按需灵活扩展，利用率高，大范围、整系统相同编址机制，控制静态、传输、动态时延。
- 长稳可靠：10 倍提升长稳运行时间，自动避障，自动恢复，易部署，机房环境适应性强，易扩展，从模组到柜级均可扩展。

面对超节点调度、池化、通信、虚拟化等诉求，openEuler 异构融合系统升级，支持超节点形态，加速释放超节点异构算力。

超节点 OS 架构

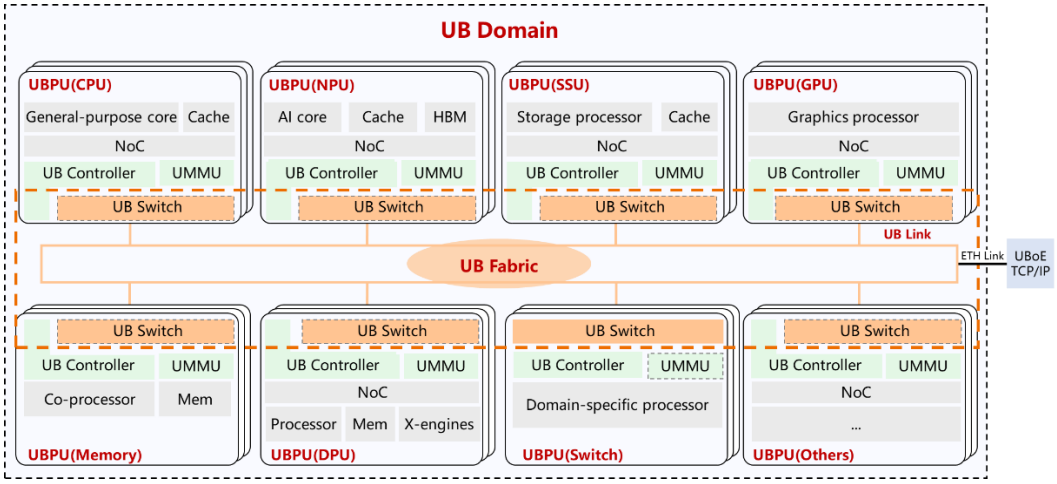
在超节点应用落地过程中，业界面临三大核心挑战：如何简化开发流程，实现业务“零修改”平滑迁移，如何精准匹配多样化算力供给，实现按需高效释放，如何应对系统复杂性提升，保障应用高可用性。openEuler 异构融合系统针对上述挑战方案如下：

- 第一，新增系统高阶服务能力，将业务迁移从“复杂适配”简化为“零修改适配”，大幅降低开发与迁移成本。
- 第二，强化异构融合核心子系统，提供异构融合通信与虚拟化能力，精准匹配算力需求，实现利用率最大化。
- 第三，完善池化设备管理体系，通过灵活扩展支持超节点形态，以池化高可靠管理防范故障扩散，筑牢业务稳定运行根基。

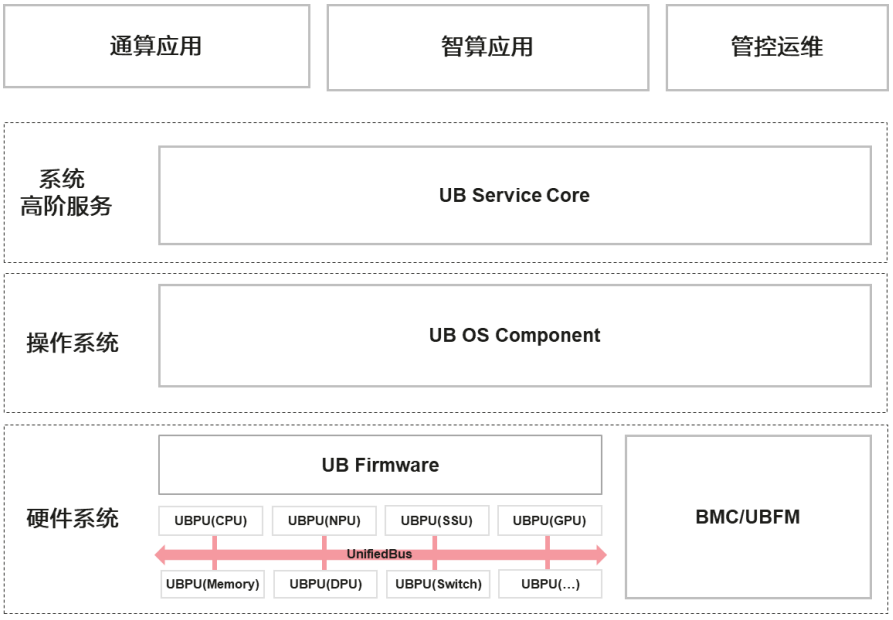
灵衢超节点 OS 实现方案

灵衢系统架构

超节点是灵衢计算系统的核心，通过灵衢重构计算系统，打破计算主机边界，实现智算和通算超节点扩展并获得性能规格收益，是面向未来 AI 时代的目标计算系统架构。灵衢超节点计算系统架构示意如下：



灵衢超节点打破传统主机边界，在超低时延、多协议归一的灵衢总线级互联的超节点域内，通过计算资源全量池化、平等互联，实现计算资源的灵活组合，超大规模组网和系统高可用性。灵衢计算系统整体参考架构如下图所示：



灵衢计算系统参考架构分成四层，共同发挥灵衢计算系统超节点架构优势：

1. 灵衢硬件系统，实现超节点可定义计算

- UnifiedBus（UB/UB Firmware/BMC）：灵衢超节点计算系统基础，基于灵衢实现计算集群总线级互联、协议归一、平等协同、全量池化、大规模组网、高可用性。
- 灵衢互联结构管理器（UBFM）：完成灵衢的互联配置，实现计算资源的灵活、高效聚合，优化计算互联 SLA（时延、带宽、可靠性），实现灵衢超节点灵活算力使能。

2. **操作系统灵衢组件（UB OS Component）完成灵衢和设备的使能，进行统一抽象与管理**

- UB OS Component 通过扩展 Linux 内核，原生支持超节点计算系统，实现对灵衢设备的统一抽象与管理。
- 同时保持 POSIX 接口兼容，实现现有应用的快速迁移。通过 openEuler 社区，使能操作系统生态，实现灵衢计算系统的开放创新。

3. **灵衢系统高阶服务（UB Service Core），使能超节点性能最优**

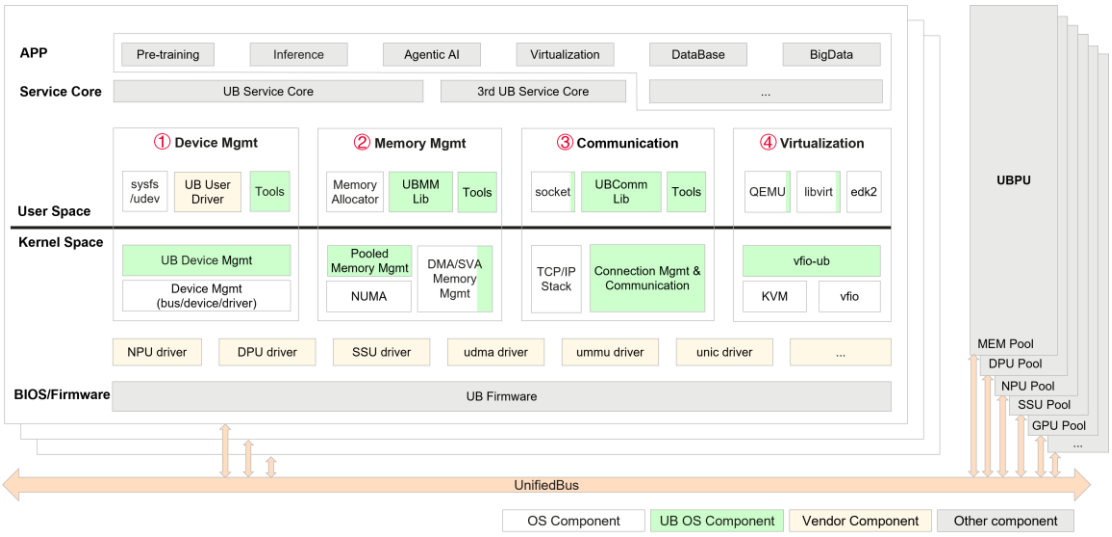
- 为灵衢系统提供内存、SSU、NPU 等多样性算力管理调度，打造灵衢系统的 ScaleUp 新能力，简化灵衢资源管理与调度的复杂性；构建内存池化、通信、IO、虚拟化等场景能力，提供多种方式的生态对接，充分释放超节点架构优势。

应用场景

以灵衢为基础构建的超节点，在面向人工智能计算与通用计算领域的 10 大核心业务场景，如大模型预训练、中心推理、后训练与强化学习、多模态内容理解与生成、Agentic AI、虚拟化、大数据、数据库、分布式存储和高性能计算等，均可提供领先的系统能力，带来计算业务性能和资源利用率提升。

详情请见灵衢社区《基于灵衢®的超节点参考架构白皮书》。

操作系统灵衢组件



操作系统灵衢组件（UB OS Component）是在 OS 原有内存管理、通信、设备管理和虚拟化框架上扩展支持灵衢，扩展的 4 个功能分别是：

1. **Device Mgmt：提供 UB 总线、UB 设备管理能力，实现计算节点内 UB 设备热插拔、配置。包含模块：**

- UB Device Mgmt：UB 设备管理基于 Linux 的 Device Mgmt（bus/device/driver）模型扩展支持 UB 设备管理，包含 UB 总线驱动、UB Firmware 交互、UB 设备热插拔等，同时提供设备驱动接口用于 UB 设备驱动开发。
- sysfs：UB 设备管理会在 sysfs 生成 UB 设备、驱动、总线信息，用户可通过 sysfs 查看和使用。
- udev：UB 设备管理会生成 uevent 事件，用户可通过已有 udev 工具获取 UB 设备的热插拔状态，实现对 UB 设备的管理。
- UB User Driver：UB 设备用户态驱动，用户可通过此接口将 UB 设备暴露到用户态，供用户态驱动或者虚拟化软件使用。

2. **Memory Mgmt：提供 UB 总线域内内存语义访问能力，实现跨计算节点跨设备内存借用、共享。包含模块：**

- DMA/SVA Memory Mgmt：基于 Linux 已有的 DMA/SVA 框架扩展实现对 UMMU 单元的管理，驱动或其他组件可通过本模块将主机内存注册给 UB 设备使用。
- Pooled Memory Mgmt：新增本地内存导出、远端内存导入、共享内存一致性维护等功能。基于 Linux 已有的 NUMA 内存管理框架上支持远端内存上线到 NUMA node 使用，或上线到内存设备（在/dev 下生成内存字符设备）通过 mmap 映射使用。
- UBMM Lib：用户态新增封装 Pooled Memory Mgmt 提供的接口，UBPRM 可通过上述接口实现 Home 和 User 侧的内存管理操作。
- Memory Allocator：兼容已有 glibc/jemalloc/tcmalloc 等内存管理接口，UBPRM 从其他计算节点借用内存并通过 UBMM Lib 上线到 NUMA node 后，应用程序无需修改即可通过 glibc/jemalloc/tcmalloc/llibnuma 等提供的 malloc/free/numa_alloc_onnode 等内存管理接口申请/释放借用的内存。

3. **Communication：提供跨计算节点、跨设备通信和远程调用功能。包含模块：**

- Connection Mgmt & Communication：实现异步通信的连接管理，同时提供内核态的异步通信接口供其他模块使用。
- UBComm Lib：新增提供灵衢异步通信和统一远程过程调用接口，应用程序可通过此接口实现超节点内节点间通信和函数调用。

- socket：兼容已有 glibc 等提供 socket 接口的库，应用程序无需修改或少量修改即可通过 socket 接口实现两个节点间通信。

4. **Virtualization：提供 UB 设备直通虚拟机能力。包含模块：**

- qemu：在现有 qemu 上扩展实现 Bus Controller、UB 设备、UMMU 的虚拟化。
- libvirt：在现有 libvirt 上扩展支持 UB 设备的创建、删除等功能。
- vfio-ub：基于 Linux 已有 vfio 框架上新增支持 UB 设备虚拟化。

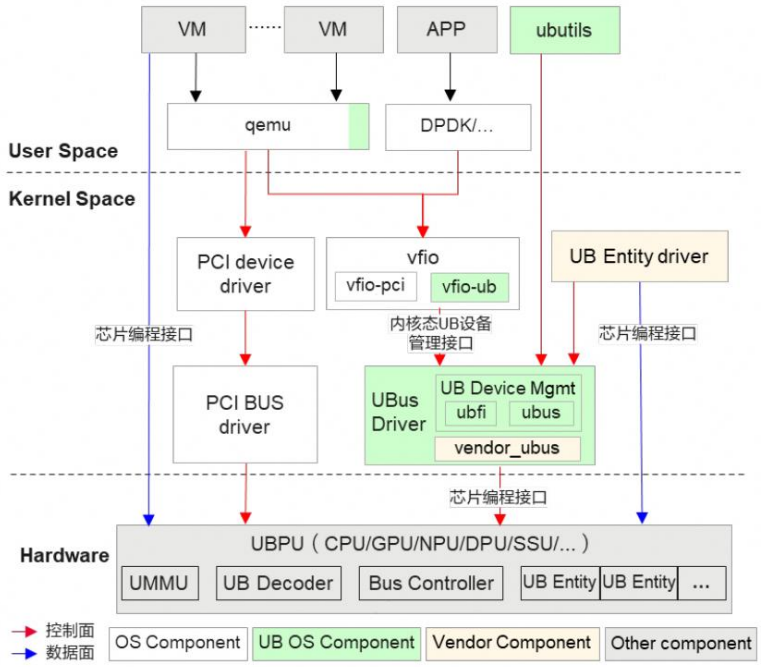
操作系统灵衢组件的功能介绍和代码介绍见灵衢社区《灵衢®使能操作系统参考设计》。

灵衢设备管理

UB 设备管理包括 UBus Driver、vfio-ub 和 ubutils 三个模块，对用户提供 UB 设备管理接口，包括给 UB 设备驱动提供了基本的设备发现、设备注册、中断使能等总线服务，UB 设备直通用户态的服务，以及给用户提供查询 UB 设备信息和配置服务。各类 UB 设备可以注册到 UB 总线上，对用户提供相应的功能。

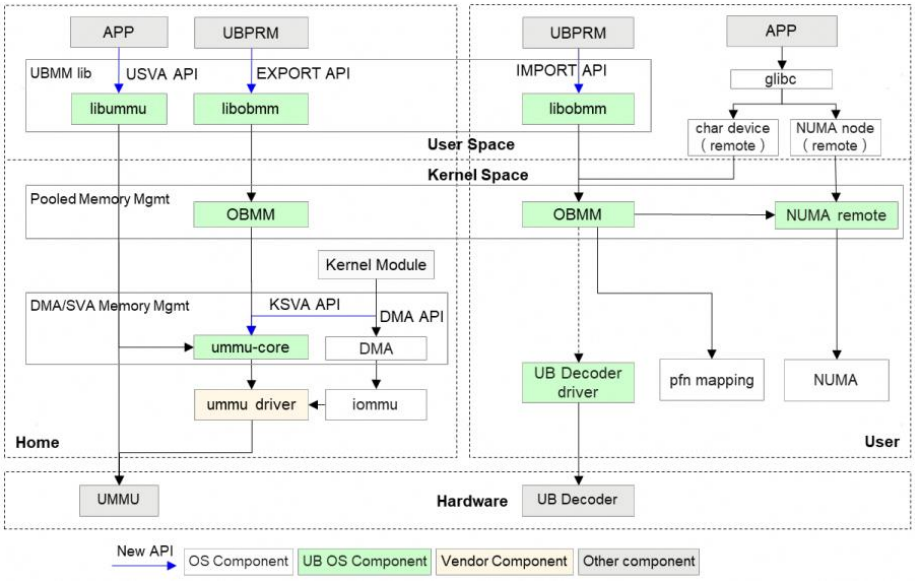
在 OS 内，UB 设备的发现、注册、驱动加载均由设备管理实现，分单机和集群场景：

1. 单机场景：单台服务器主机独立工作，每台服务器独享连接在主机上的所有 UB 设备，单机上的 UBus Driver 完成所有 UB 设备的枚举发现和使能，配合 vfio-ub 提供 UB 设备直通用户态能力。
2. 集群场景：超节点形态下 UBus Driver 提供分配给计算节点上的 UB Entity 发现和使能，并支持池化设备接入和移除，配合 vfio-ub 提供 UB 设备直通用户态能力。



灵衢内存管理

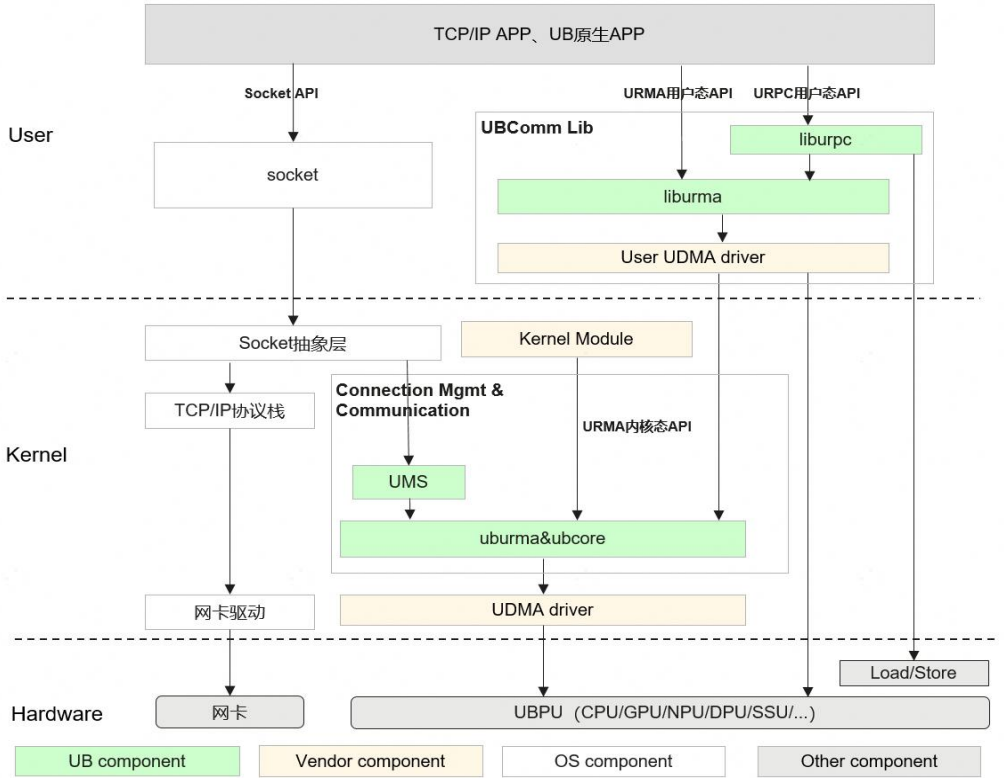
现有总线采用 Host-Device 模型，以 Host（CPU）为中心，Host 管理每一个 Device。在这个模型之下，Host 访问 Device 内存、Device 访问 Host 内存、Device 访问 Device 内存共三类访存路径有单独实现。UB 统一上述访存模型，根据一个访存发起流程将相关的节点分为 User 和 Home，User 可以通过同步（Load/Store）或者异步（DMA）的方式访问 Home 侧内存资源。



基于 UB 总线的能力，OBMM 模块提供了节点间数据通路配置与跨节点数据一致性维护能力。完成配置后，用户可以在 UB 互联的集群中，在一个节点上访问另一个节点上的内存，实现跨节点内存共享、内存池化与节点间借用等功能。

灵衢通信

灵衢通信库是分布式通信软件库，为数据中心网络、超节点内、服务器内的卡与卡之间提供高性能的通信接口，使能和释放 UB 硬件能力。

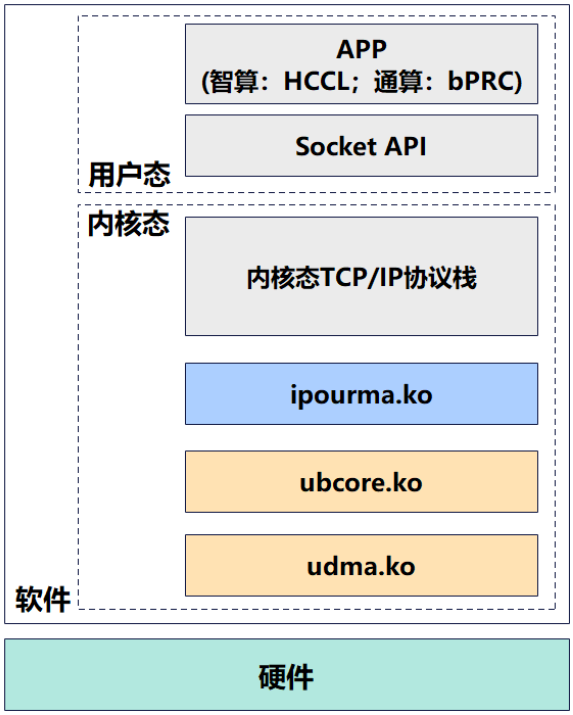


- UMS：UMS 支持对接 Socket 抽象层，是北向兼容 Socket 编程接口，南向基于灵衢高性能网络进行数据传输的内核网络协议栈，透明加速 TCP 应用，实现性能提升。
- URMA：URMA 是灵衢通信的基础软件库，屏蔽硬件差异，基于灵衢高性能网络提供读写、收发、原子操作等远端内存访问语义，是灵衢通信应用的基础。
- URPC：URPC 是灵衢原生的统一远程过程调用，支持灵衢原生高性能主机间和设备间 RPC 通信，以及 RPC 加速。

IP over URMA

功能描述

IPoURMA 提供了一种基于 UB 协议和硬件传输 IP 数据包的标准化方法。IPoURMA 作为内核模块，基于 UB entity 创建 netdev 设备并注册基本的 netdev 回调函数，以提供内核网络协议栈控制 UB 硬件的抽象层[1]。此外 IPoURMA 还提供了 ethtool 等维测工具的实现，能够统计基于 UB 硬件的网络包维测信息。



应用场景

IP over URMA 提供了 UB 通信使能 TCP/IP 协议的功能，旨在融合 Ubus（UB）高性能特性与以太网的广泛兼容性，此外在 UB 网络能够取代以太网卡提供标准 TCP 协议栈的 socket API，提升 UB 易用性。在以下场景都存在应用价值：

（1）AI 超节点和集群训练

IP over URMA 使能标准 TCP 通信，昇腾智算领域的集合通信等采用此功能，对接 TCP/IP 协议实现控制面通信，从而节省以太网卡的硬件成本。

（2）通用计算组网通信

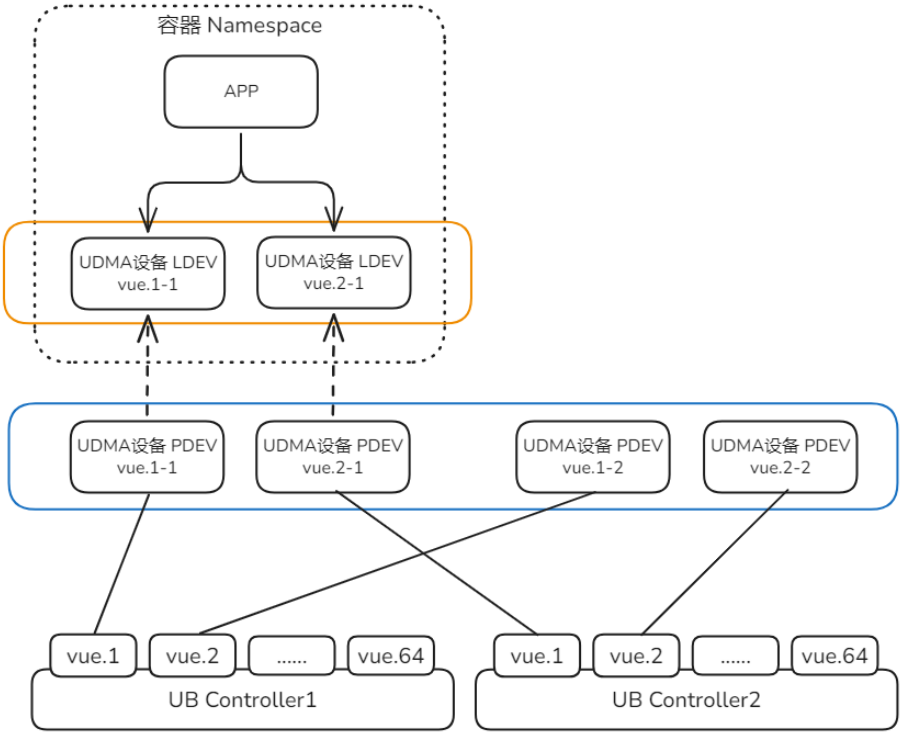
[1] IP over URMA 约束当前仅支持 IPv6。

通用计算组网通信场景，采用 IP over URMA 使能 TCP 通信，能够结合 TCP 与标准 TLS 认证方法实现安全通信。同样的，此场景可节省以太网卡硬件成本。

URMA 容器场景通信

功能描述

该 URMA 容器方案在容器化环境中构建了一种**设备级共享、EID 隔离**的远程直接内存访问能力。通过硬件虚拟化与内核中介技术，方案允许同一物理 URMA 设备同时暴露给指定的多个容器网络命名空间，实现硬件资源的集约复用；同时，URMA 设备的每个 EID 在被绑定至单一命名空间，确保租户或应用间的 URMA 资源无法相互越界访问。



应用场景

在多租户环境下，这套 URMA 容器方案能够根据实际资源情况和隔离需求，灵活分配设备和 EID 资源。如果物理设备充足，可以为每个租户单独分配一整块 URMA 设备，实现硬件级的完全隔离，租户之间互不干扰。当设备数量有限时，多个租户、或者一个租户的多个服务也可以安全地共享同一台 URMA 设备——虽然底层硬件是共用的，但每个容器命名空间拥有自己独立的 EID，严格隔离了内存访问权限，租户 A 无法感知或访问租户 B 的资源。

URMA 高性能分发通信加速

应用使用 URMA 通信时，如果在业务的逻辑连接采用 Jetty/JFS 与 JFR 一比一配比创建，其 JFR 接收缓存消耗将随逻辑连接数线性增长。以通信内存按 8KB 分片管理，并且 JFR 深度为 1024 为例，每个业务的逻辑连接需预占用 8MB 接收缓存，当逻辑连接规模达到 1K~10K 时，仅接收缓存预留就需要 8GB~80GB。为减少内存预占用，UMQ 提供基于 URMA 共享 JFR 的通信模式，支持应用的多个逻辑连接共享一个 JFR 接收数据，并使用业务关联的标签高效地分发接收数据到正确的逻辑连接上。

功能描述

UMQ 提供基于 URMA 共享 JFR 的通信模式，支持从共享 JFR 接收的 RQE 到业务逻辑连接句柄的映射关系配置，便于应用高效分发 RQE，也提供了根据共享 JFR 实时接收缓存深度的流量控制机制，避免出现流量超发。该特性主要功能如下：

- 使能共享 JFR：创建一组 UMQ 时均指定同一个 UMQ 作为 JFR 共享源，则这一组 UMQ 底层使用同一个 JFR，应用监控被共享的 UMQ 可获取这一组 UMQ 的收包事件。
- 支持高性能分发：UMQ 创建时可设置应用逻辑连接关联的标签等信息，UMQ 上报的接收数据也携带了这个信息，支持应用在获取数据后高效分发到对应的逻辑连接。
- 自适应流控：接收侧维护共享 JFR 中接收缓存队列深度关联的信令池，发送侧发送数据时需拥有足够的信令。当信令不足时，发送侧根据历史信令消耗情况自适应调整本次信令申请数量，如果信令发生超时空闲，则需归还信令。

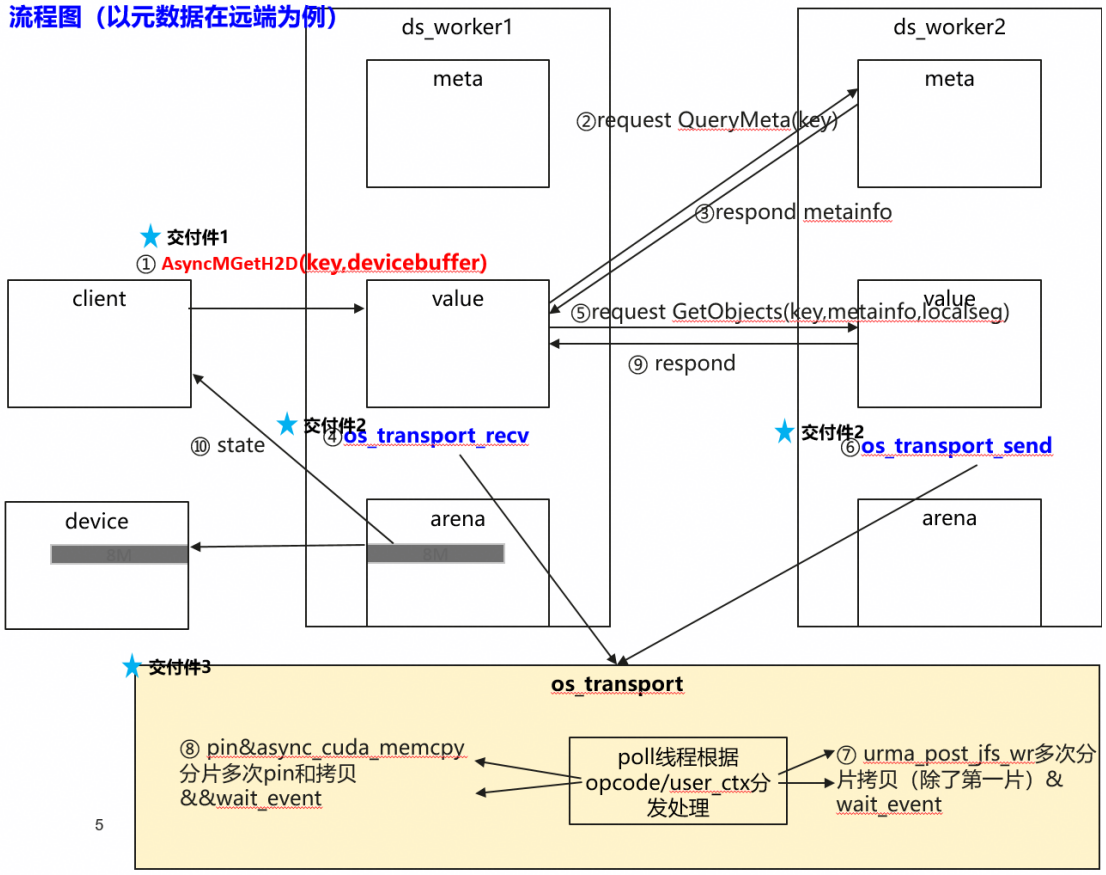
应用场景

在节点数多、连接规模大的场景下使用该方案，应用可根据业务的并发度和活跃连接数合理规划通信内存，降低通信路径的内存成本。如在搜推广业务场景下，使用 RPC 通信管理全局的任务调度，各节点 fullmesh 连接，但实际活跃连接数（100 量级）不高，采用该方案可在 1K 连接规模下降低通信内存至 1GB，并且保持使用 URMA 通信的性能优势。

RH2D 多级缓存直通加速

功能描述

RH2D 面向异构融合 OS、分布式缓存场景，提供将远端 Host 上的 KvCache 通过异步流水传输到本端 Device 的能力。其核心目标是在 KVCache 等大对象读取过程中，减少传统“远端读取到本地内存、再由客户端拷贝到 GPU”的串行等待开销。该能力基于 URMA 高速传输通道，将远端对象按固定大小的分片切分，通过异步流水实现远端数据搬运与本端 GPU 写入的并行推进。对于大块数据，RH2D 可以边接收、边投递、边拷贝，缩短端到端获取 KvCache 的时延，提升 KvCache 从远端写入 GPU 的效率。



应用场景

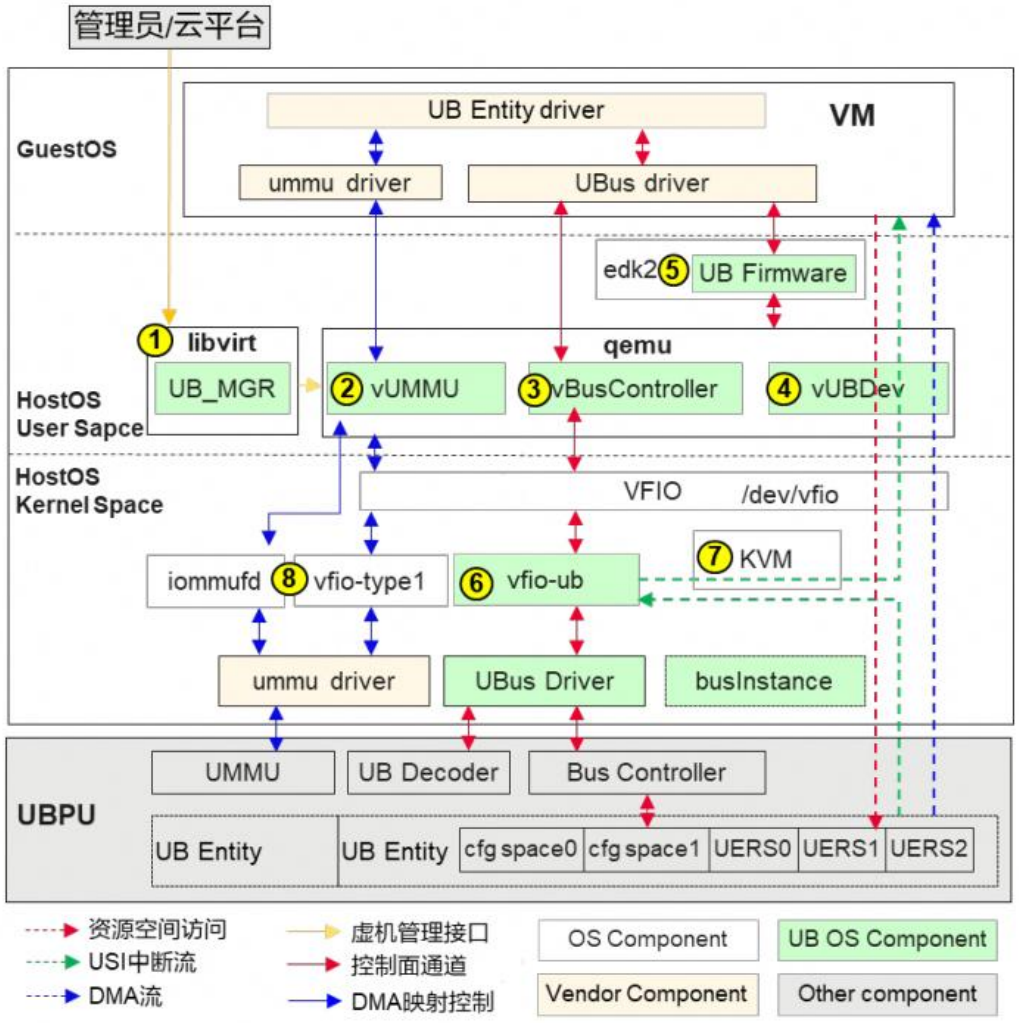
RH2D 适用于 AI 推理、训练、分布式缓存和异构计算等需要频繁从远端节点获取大块数据并写入 GPU 的场景，适合大 value、多并发、多线程读取场景，可降低数据获取过程中串行等待时间，例如大模型推理中的 KVCache 拉取、跨节点样本缓存读取等。

灵衢设备虚拟化

在云计算场景中，通过虚拟化将物理机上的硬件资源隔离给多个虚拟机，极大提升了资源利用率。UB 作为新一代高速互联总线，同样支持虚拟化能力。

当前设备虚拟化的主流技术主要包括如下几种：IO 全虚拟化、IO 半虚拟化、硬件辅助 IO 虚拟化。传统架构下每一台服务器都是通过插在其上的网卡连接到 TOR 交换机，所以最大可支持的带宽是固定的，在实际应用过程中会带来闲时带宽利用率低、忙时带宽不够用、服务器间带宽压力不均衡等问题。

在 UB 总线中，UE（UB Entity）作为一个 UB 设备相对隔离的功能单元，UB 虚拟化的功能同样支持通过直通的方式将其直接分配给虚拟机使用，以达到设备原生的性能，UB 设备虚拟化在传统的硬件辅助 IO 虚拟化之上扩展对池化 UB 设备的直通访问。通过将网卡、DPU 设备资源池化，可以解决上述问题。所有设备资源统一在设备池中，所有服务器通过共用设备池里的网卡资源，可以根据服务器实际的网络负载情况动态申请使用相对应的网卡资源，达到提升设备资源利用率，避免网络带宽瓶颈。同时，灵衢特有的内存管理与大带宽通信特性，能给虚机提供更灵活的内存超分与极速热迁移能力。



- UBNative：基于 UB 协议构建的高性能设备虚拟化解决方案，通过模拟 UB 设备模型，实现虚拟机对 UB 总线的无缝接入。依托 VFIO 技术，将 UB 设备以直通方式给虚拟机使用，确保在虚拟化环境中充分发挥设备原生性能，实现近硬件级的运行效率与资源利用率，为高吞吐、低延迟场景提供稳定支撑。

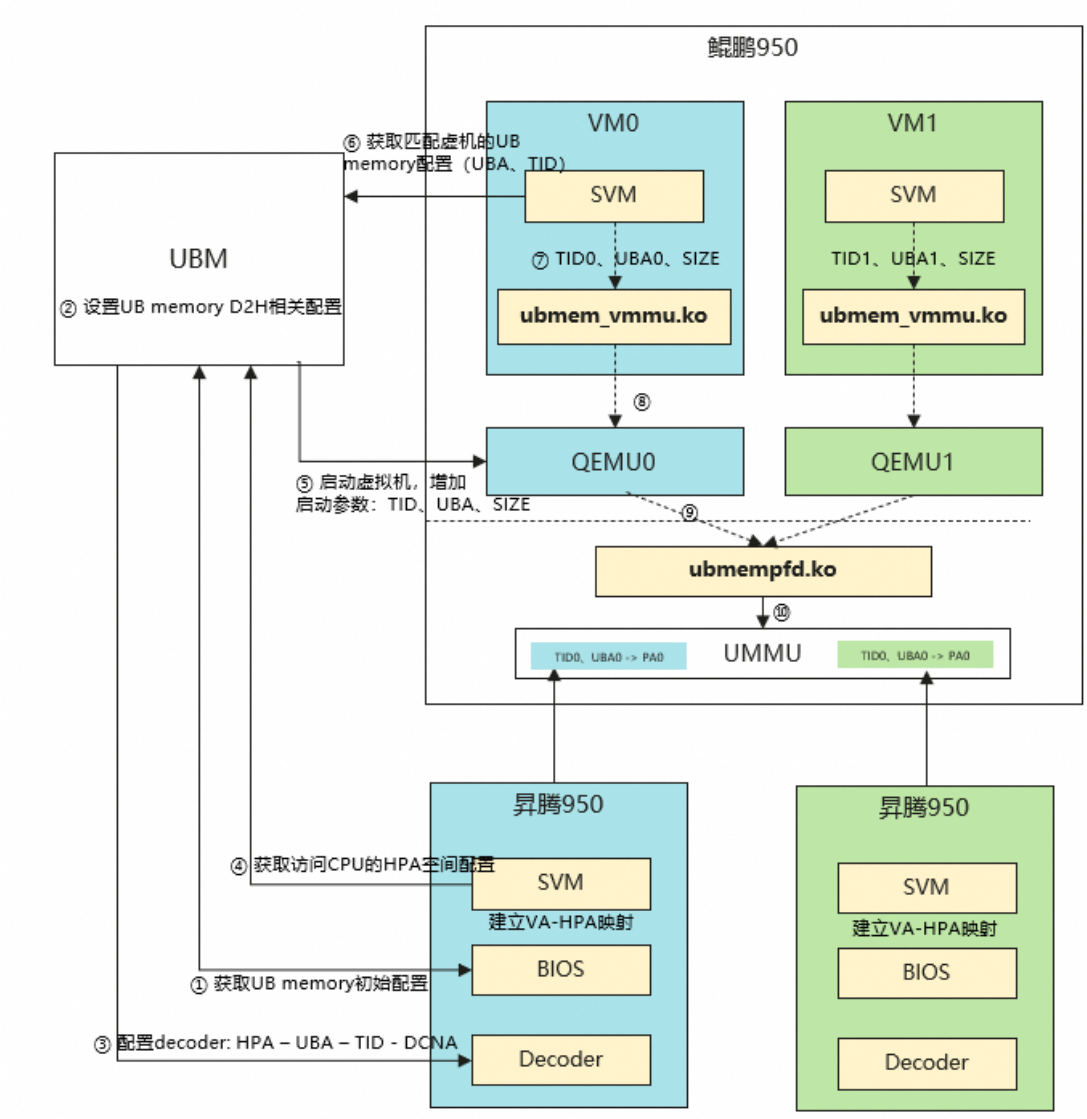
- 内存超分：随着主机内存规格的增大，内存成本在整体 TCO 占比持续走高，云场景下提升内存利用率的目标越来越重要。实现基于灵衢超节点架构下的内存超分功能，构建面向大页虚拟机、直通虚拟机的内存回收方案，充分回收虚拟机内部的无用内存，转换让其发挥业务价值，从而能提升虚拟化内存利用率，进一步降低云厂商 TCO。

- 极速热迁移：基于 URMA 内存读写语义，利用灵衢高性能网络进行内存数据迁移，减少迁移中断时间、减少资源开销、并提升高内存压力虚拟机的迁移成功率。

设备直通灵衢虚拟机

QEMU 支持 NPU 卡 D2H

功能描述



本功能是 UB Memory 子系统的一部分，UB Memory 位于 GuestOS 业务驱动与 Host UMMU 之间，提供接口供 GuestOS 上的业务驱动子系统或者模块调用，上层主要为 SVM 模块。

UB Memory 子系统南向使用 QEMU 模拟的 VUMMU 硬件，QEMU 实现对 VUMMU 设备的模拟，然后通过 UB Memory Mapping Driver 提供的接口，完成 Guest 内 map/unmap 操作到 Host 内存的 map/unmap 的功能透传。

使用方式：

虚拟机 xml 中新增 ubmem_vmmu 设备配置：

```
<qemu:commandline>
  <qemu:arg value='-device'/>
  <qemu:arg value='ubmem_vmmu,tid=1,uba=0,size=204800000'/>
</qemu:commandline>
```

参数说明：

ubmem_vmmu：设备固定类型，无需改变；

tid：TokenID，由 UB 管控面指定，每个虚拟机都有一个不同的 tid，用于 NPU 卡发送 UB 报文时找到目的虚拟机是哪个虚拟机。在命令行中指定的作用是进行合法性校验（防止 GuestOS 内部驱动被攻破）；

uba、size：UB memory address 的范围，由 UB 管控面指定，NPU 卡发起的 UB 报文的目的地址。在命令行中指定的作用是进行合法性校验（防止 GuestOS 内部驱动被攻破）；

应用场景

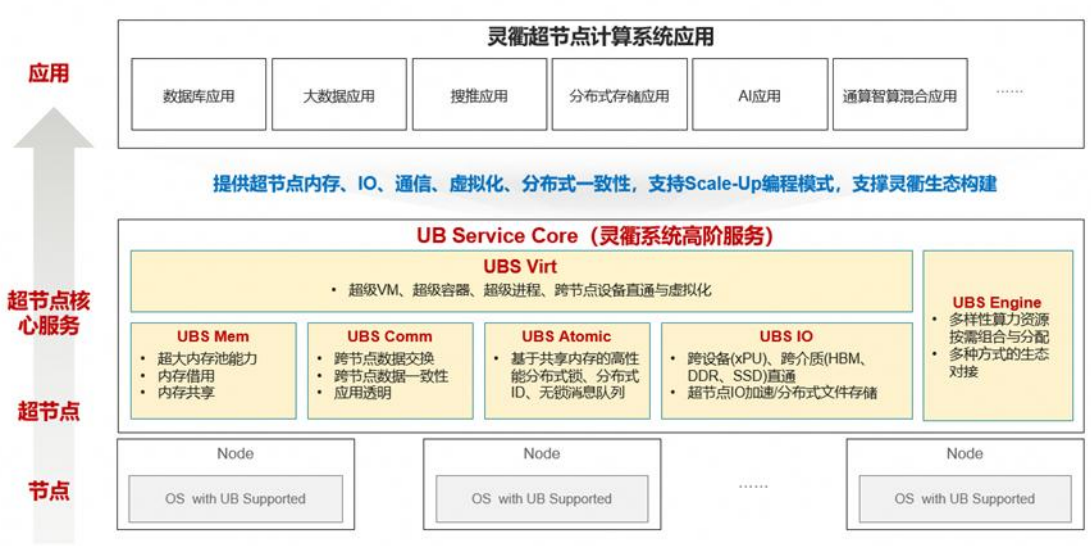
在使用 PCIE 总线连接 NPU 的虚拟化场景，为了兼顾安全隔离和性能，会将 NPU 卡直通给虚拟机内部租户使用，此时租户的 KVCache 是存放在虚拟机内部的内存中（租户感知不到 Host），NPU 卡可以通过 PCIE 总线和 SMMU 硬件直接访问位于虚拟机内部的 KVCache，从而避免 KVCache 从 Guest 传输到 NPU 卡内部造成的性能下降。

在使用 UB 总线连接 NPU 的虚拟化场景，使用高性能 UB 总线代替 PCIE 总线连接 NPU 卡和 CPU，由于当前 UB 总线暂不支持 NPU 以全虚拟化的方式访问虚拟机内部内存（类似于 SMMU 页表），所以需要提供一种半虚拟化的方式，让位于虚拟机内部的 UB 驱动主动将 KVCache 的信息（GPA、SIZE 等）传递给 Host，然后由 Host 侧建立 UMMU 页表，此后 NPU 就可以通过 UB 总线访问这些 KVCache 内存了。

灵衢系统高阶服务

面向灵衢超节点，为了应用快速使能超节点能力，灵衢系统构建了 UB Service Core，封装 UB 底层能力、集群拓扑等，简化并兼容现有生态，让应用用本地资源一样使用超节点资源，简化应用开发，使能灵衢超节点。

UB Service Core 构筑 6 大集群级系统服务，释放超节点平等互联架构优势，全面使能应用加速 30~50%，促进灵衢系统软件生态构筑，灵衢系统高阶服务参考架构如下图所示：



UB Service Core (简写 **UBS Core**)，包含如下 6 大部分：

1. **UB Service Core Engine**（简写 **UBS Engine**）：支持内存、SSU、NPU 等资源池化管理与动态调度，支持分布式自选主，支持 N 节点场景下最多 N-1 节点失效的高可用，是灵衢计算系统的控制面核心参考实现。
2. **UB Service Core Memory**（简写 **UBS Mem**）：支持统一内存编程，实现灵衢超节点的共享内存、池化内存。
3. **UB Service Core Communication**（简写 **UBS Comm**）：基于超节点提供高性能、高可靠以及生态兼容（用户态 Socket/Verbs over UB）的通信协议。
4. **UB Service Core IO**（简写 **UBS IO**）：基于超节点，提供应用亲和的全局数据读写缓存系统高阶 IO 服务。
5. **UB Service Core Virt**（简写 **UBS Virt**）：支持虚拟化池化，热迁移策略决策，极速快恢与容灾，虚机/容器间极速通信等能力，使能虚拟化性能提升。

6. **UB Service Core Atomic**（简写 **UBS Atomic**）：利用 UB 原子操作指令和跨节点共享内存功能，提供集群分布式锁、全局事务 ID 管理、集群队列通信能力。

详情请见灵衢社区《灵衢®系统高阶服务软件架构参考设计》。

灵衢可靠性

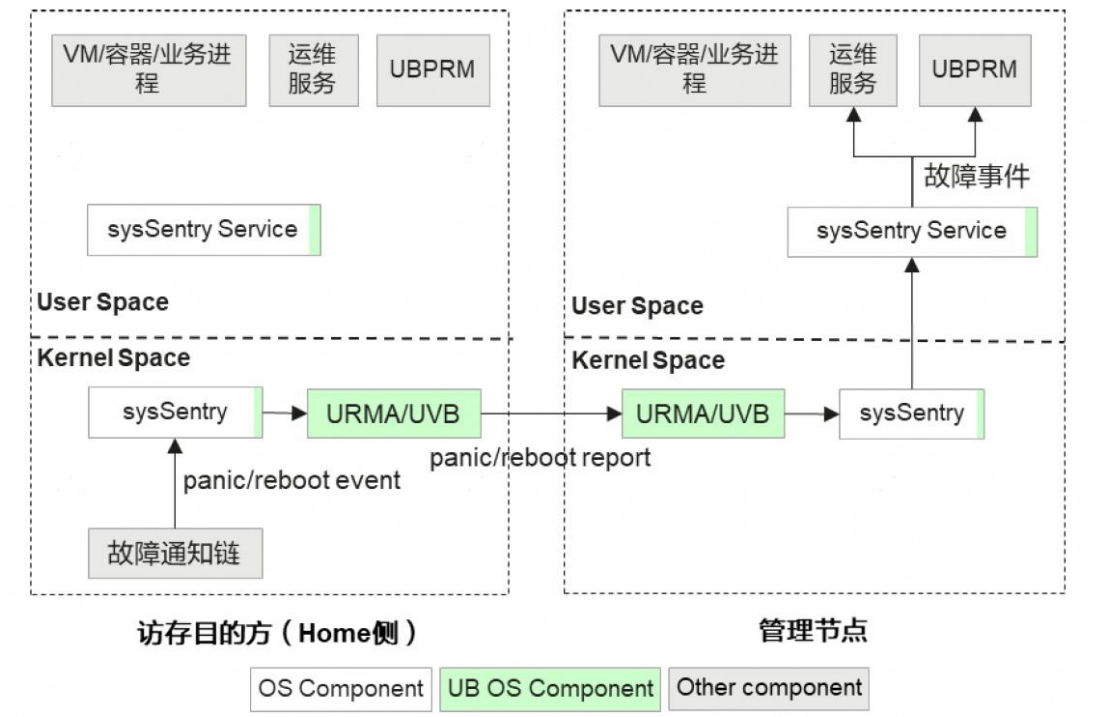
灵衢可靠性插件

功能描述

当紧急事件（OOM、Panic、reboot 等）发生时将相关的紧急事件阻塞，并上报事件到 UBPRM 中，防止发生数据丢失或业务中断，同时也负责统一管理通过 UB event 上报的事件。

节点故障检测与恢复

当 Home 侧节点发生 Panic/reboot 时，该节点借出的内存将无法访问，会影响 User 侧的业务进程，在此情况下需要将保存在 Home 侧内存中的数据迁移出来。



详细通知流程如下：

Home 侧节点内核故障流程中（例如 panic、reboot、shutdown 流程等）通知 sysSentry 即将复位重启。

1. sysSentry 通过 URMA/UVB 通道进行故障事件广播。
2. 故障事件从 Home 侧节点的 URMA/UVB 通道发送到管理节点的 URMA/UVB 通道。

3. 管理节点的sysSentry收到故障事件，通过sysSentry Service通知到管理节点UBPRM。
4. UBPRM 可通过 sentryctl 相关命令配置 sysSentry 开启对 Panic/reboot 事件劫持、配置跨节点通信必备参数配置，通过订阅 OS Panic 以及 OS reboot 类型事件来监听 Panic/reboot 事件。

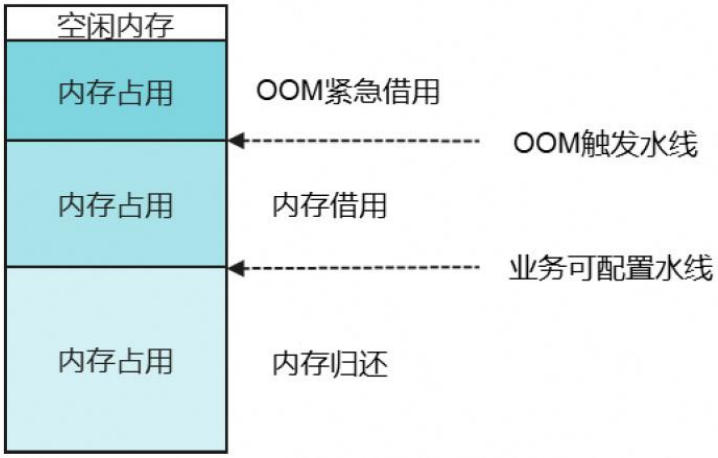
当 UBPRM 收到 sysSentry Service 上报的故障通知后，可采取相应的故障隔离和恢复措施，例如将 Home 侧节点内存迁移至其他节点并解除和故障节点的借用关系，确保使用池化内存的业务不受影响。

处理完成后，需返回故障处理结果，系统按照如下流程进行恢复：

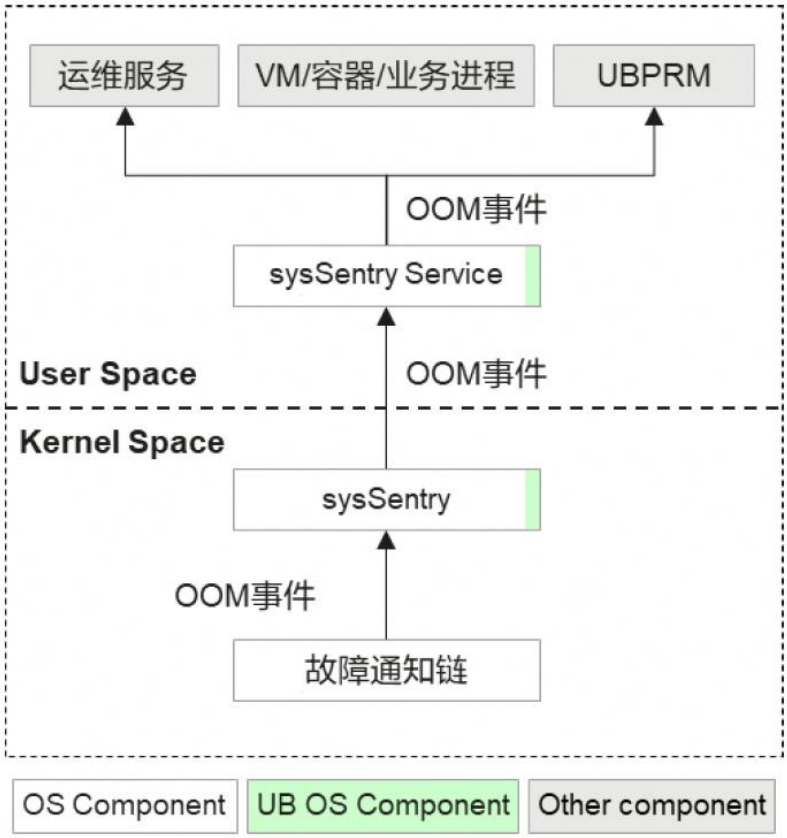
1. UBPRM 通知 sysSentry Service 故障处理结果。
2. 故障处理结果经过 sysSentry 和 URMA/UVB 通道发送到 Home 侧节点 URMA/UVB 通道。
3. Home 侧节点 sysSentry 从 URMA/UVB 通道拿到故障处理结果。
4. Home 侧节点返回内核故障流程，继续复位重启。
5. Home 侧节点如果在一定时间内未收到 UBPRM 的通知，则同样会复位重启。

OOM 检测、预防与恢复

在内存借用场景下，UBPRM 会按照固定的时间周期检测内存占用水位情况，该水位可配置，根据实际的使用情况采用内存归还/内存借用的策略。当短时间内发生大量的内存占用，导致内存水位迅速上涨时，UBPRM 检测机制可能无法及时发现该现象，导致业务节点发生进程被杀死或节点重启，发生业务中断。此时可通过 OOM 预防机制将 OOM 事件上报给 UBPRM，触发 OOM 的紧急借用策略。总体处理策略如下图所示：



当触发 OOM 紧急借用时，通知流程如下：



1. 发生 OOM 的节点通过 OOM 故障通知链上报 OOM 事件至 sysSentry。
2. sysSentry 将 OOM 事件上报至 sysSentry Service。
3. sysSentry Service 将 OOM 事件上报至 UBPRM 或运维服务。

UBPRM 可订阅 sysSentry 的 OOM 类型事件，当节点发生 OOM 事件后，UBPRM 会收到 sysSentry 服务上报的事件通知，并做相应的处理，建议处理流程：

1. UBPRM 借用其他节点内存，并上线到业务节点的 OS。
2. OS 在 OOM 中尝试申请内存，如果申请到，则返回给对应的应用，否则杀死对应的应用。如果是内核态申请内存出现 OOM，则走入 Panic 流程整机复位。

应用场景

灵衢内存池化场景下，节点发生异常故障导致借用内存即将失效时，通过劫持对应故障事件，通知管控面进行内存紧急迁移，避免业务中断。

内存池化故障劫持&通知

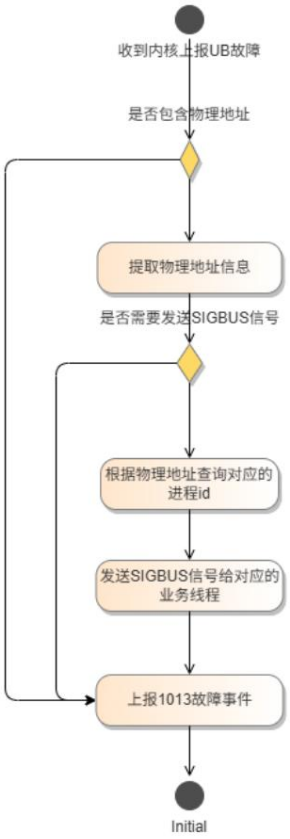
功能描述

操作系统对于访存故障有标准的处理方式，访存故障根据硬件规范上报为 SEA 或 SEI，内核基于规范进行处理，处理方式包括将内核直接 panic 或隔离对应内存并给使用该内存的用户态线程发送 SIGBUS 信号。操作系统内核对于在内核内存管理子系统中的内存都可以进行上述故障处理流程。

SEA/SEI 针对主动访存（LD/ST 指令）操作触发的同步或异步故障事件上报，针对灵衢还有一类影响范围较大的 linkdown 故障事件。由于所有的灵衢跨节点内存访问都需要经过 UB 链路，如果链路本身发生断链等故障可能导致所有远端访存失败。UB 提供链路断链故障的主动上报，硬件检测到断链故障后可通过 BIOS 上报给内核 UBUS 组件，sysSentry 订阅该事件后上报给用户态，用户态收到该事件后可进行自定义隔离或修复等操作。

超节点内存池化提供借用和共享两类使用方式，其中借用内存上线至操作系统，受内存管理子系统控制；借用内存与操作系统本地内存故障处理流程类似，由于借用内存通过 remote_numa 上线当前操作系统，并通过内存管理子系统对外分配使用，业务可通过特定的访存策略申请分配，内存故障时通过现有的 SEA/SEI 方式上报 OS，OS 通过已有的 memory_failure 流程进行内存故障处理，包含内存隔离与杀死进程等操作；共享内存通过独立内存设备上线，绕过内核内存管理子系统，以设备的形式被用户态进程直接 mmap 使用，这类内存故障需要额外处理。灵衢超节点针对内存故障额外提供 UBEvent 异步事件上报访存故障信息，该事件由 UBUS 驱动直接处理，并对外提供钩子函数。sysSentry 注册该钩子函数处理上报的 UBEvent 内存事件，针对上报的故障内存地址及故障进行处理，通过该地址反向查找内存设备文件，在用户态遍历所有映射使用该内存设备的进程并发送 sigbus 通知线程该内存故障。

通过这一特性补齐共享内存场景下内存故障的业务通知能力，业务感知故障后可自行进行处理。



应用场景

以使用灵衢共享内存的数据库业务为例，跨节点多数据库实例通过共享内存作为全局 Buffer Pool；数据库业务遇到内存故障后会收到 sigbus，数据库可以截获 sigbus 自行处理，避免业务整体被杀死，只针对该内存故障所对应的事务进行回滚操作，并隔离该内存，通过日志重放等操作恢复数据，降低内存故障对业务的影响。

在离线混部场景使用内存借用，在线业务优先级较高使用本地内存，离线业务使用借用内存避免对在线业务造成性能影响；离线业务通过亲和性设置配置使用远端内存 numa 节点，内存出现故障时会通过已有内核 memory_failure 处理流程进行内存地址隔离与进程 sigbus 信号处理。

内存借用文件缓存故障防扩散

功能描述

pagecache 是操作系统提供的，跨进程共享内存的一种方式；灵衢超节点场景下，机器上会存在不和 cpu 关联的 numa node，称之为 cpuless node；在 cpuless node 上分配的内存，物理位置可能不在本机；因为 pagecache 共享的特质，一旦出现错误，就会导致所有使用这块 pagecache 的进程均出现问题。

新增一个 sysctl 参数：vm.filemap_alloc_local，用于控制 pagecache 的分配策略，该参数配置为 1 时，pagecache 不会分配到 cpuless 的 numa node 上，为 0 时则保持默认分配逻辑。

mempolicy 支持：

参数配置为 1 时，在分配 pagecache 时，会从进程配置的 mempolicy 的 numa node 中，去除所有 cpuless 的 numa node，policy 保持不变；如果 mempolicy 中仅有 cpuless 的 numa node，则会改为取进程的 mems_allowed 配置并去除 cpuless 的 numa node 作为新的 mempolicy 可用 node，policy 本身保持不变(bind/prefer/interleave)。

如果开启 cgroup 级别，cpuset 的 spread_mem，则会按照 spread_mem 逻辑在非 cpuless 的 numa node 上交织分配 pagecache。

参数配置为 0 时，则对 pagecache 分配无影响。

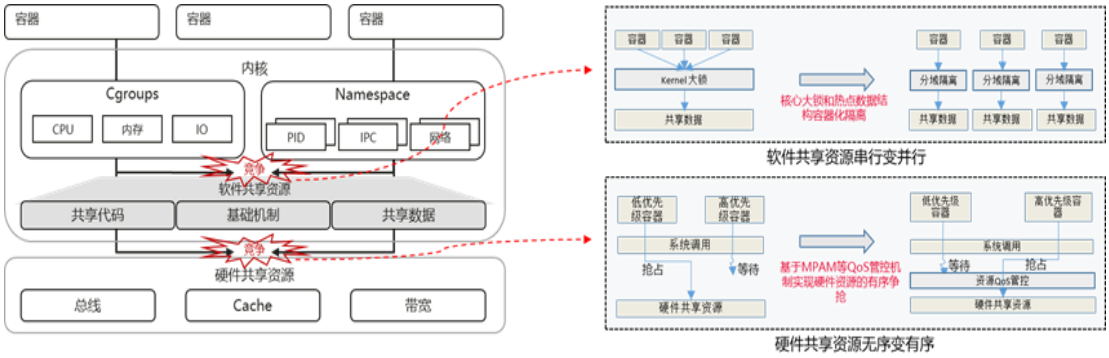
应用场景

针对远端和本地存在文件共享的场景，通过约束 pagecache 的分配位置，能够有效避免远端 pagecache 出现故障时，故障扩散到本地的情况。

云原生场景创新

众核高密

服务器芯片由多核进入众核时代（>256C），对操作系统提出新的挑战。提升 Rack 计算密度、降低数据中心 TCO，众核服务器已成为互联网行业主流选择，随着云技术和业务规模发展，容器化部署成为互联网行业的主流业务部署形态，在这种场景下，系统串行开销和同步开销限制可扩展性，干扰问题凸显，资源利用率低，影响容器部署扩展性的串行访问开销和同步开销主要来自软硬共享资源争用。



功能描述

本期主要采用轻量虚拟化按 NUMA 分域拆分资源、域内实现资源容器级隔离增强，降低因软硬件资源争用导致的性能干扰，提升容器部署扩展性。关键技术特性如下：

- 虚拟机内存 QoS 控制：当多个租户的虚拟机（VM）部署于同一物理主机时，若内存密集型虚拟机占用大量内存带宽，可能引发资源争用，导致其他虚拟机无法获得足够的内存带宽以满足其业务性能需求，进而影响整体系统服务质量。基于鲲鹏处理器提供的内存带宽监控与调控能力（MPAM, Memory Power and Access Monitoring），结合操作系统层面的 resctrl（Resource Control）机制，系统可实现对最多 30 个虚拟机的内存带宽使用情况进行精细化监测与动态控制。该能力支持对虚拟机内存带宽的上限、下限及优先级策略进行配置，从而构建多租户环境下的内存带宽资源隔离与保障体系。具体而言：内存带宽上限控制：通过为各虚拟机配置最大内存带宽使用阈值，有效防止单个虚拟机过度占用内存带宽资源，避免对其他租户虚拟机造成性能干扰；内存带宽下限保障：支持设定最低带宽保障值，确保在虚拟机实际负载较低时，系统可自动提升其带宽使用优先级，实现资源的动态优化与高效利用；优先级调度策略：支持基于业务重要性配置虚拟机的内存带宽优先级，优先保障关键业务虚拟机的带宽稳定供给，提升高优先级工作负载的可用性与服务质量；

- 虚拟设备 NUMA 亲和：PCI 设备也具有 Numa 亲和性，在主机侧直接访问，OS 调度系统会根据设备亲和性进行调度优化，以防止跨 Numa 访问 PCI 设备而造成性能损耗。对于虚拟机设备直通来说，当前还不具备在虚机内部呈现 PCI 设备亲和的 Numa 节点。本功能基于 PCI 扩展桥(PXB)扩展虚拟机 PCI 设备拓扑结构，支持在虚拟机内呈现虚拟设备所在 NUMA，便于系统 OS 优化调度或者用户根据虚拟设备所在 NUMA 部署业务应用，减少跨 NUMA 资源访问导致到性能损耗，提高虚拟机内业务应用性能；

- 轻量虚拟化：虚拟设备中断卸载实现虚拟 virtio 设备中断卸载至硬件注入，降低存储虚拟化损耗；PMD 队列负载均衡实现虚拟磁盘 IO 队列从单 reactor 均衡分散到多 reactor，消除 cpu 瓶颈，降低存储虚拟化损耗；
- CPU 分域调度：CPU 基于硬件拓扑划分子域部署容器，一个容器一个独立子调度域，实现容器之间干扰隔离，降低跨 cluster cache 同步次数和 cache/NUMA 内存等硬件资源争抢，减轻容器之间的相互干扰。对于 redis 多并发场景性能提升 10%+；
- 文件系统块分配干扰隔离：优化 ext4 块分配释放流程中的 group lock 和 s_md_lock 两个主要争抢的锁，以提高 EXT4 块分配流程的可扩展性。通过允许在当前目标块组被占用时尝试使用其他空闲块组进行分配，从而减少了多个容器争抢同一个块组造成的 CPU 浪费，并充分利用了 ext4 多块组的优势，缓解了 group lock 的竞争。其次通过将流式配的全局目标拆分为多个，从而减少了全局锁 s_md_lock 的竞争文件数据也更加聚集。在 64 容器并发场景下，块分配和块释放混合场景 OPS 提升 5 倍以上，单块分配场景提升 10 倍以上；
- 高效 slab 回收：将 slab 内存回收的读写锁，优化为 RCU 无锁化 slab 回收，不同 slab 之间的内存回收互不干扰，回收效率显著增加，多容器并发场景下，系统调用性能显著增强；
- 网络 tcp hash 干扰隔离：tcp_hashinfo bash、ehash 存在锁竞争，ehash 计算频繁，导致高并发下带宽下降，时延变大。将 tcp_hashinfo bash、ehash 的自旋锁改为 rcu，ehash 计算方式改为 lport 递增减少查询时间和计算次数，减少 tcp connect hash 的锁竞争；
- Cgroup 隔离增强：user namespace 通过 percpu counter 替换原来的原子操作，避免不同 namespace 相同父节点竞争访问，消除容器间 rlimit 计数干扰。 解决 will-it-scale/signal1 用例线性度问题，64 个容器并发吞吐性能提升 2 倍。通过对 memcg 实现批量释放处理，避免大量的小内存释放对于相同父节点计数竞争，提升内存计数的可扩展性，tlb-flush2 测试用例 64 容器吞吐提升 1.5 倍；基于 eBPF 可编程内核能力，提供主机容器信息隔离与过滤机制，高效实现容器资源视图。相较业界 LXCFS 方案，本方案避免了内核态-用户态切换开销，消除了 LXCFS 进程的性能与可靠性瓶颈，单容器内资源视图吞吐量在单容器场景下性能提升 1 倍，在 64 容器场景下提升 10 倍；
- 干扰监测：干扰监测回答的是容器有没有被干扰、是什么干扰、干扰程度如何这三个问题，从结果上看干扰可以分为干扰导致指令得不到执行、干扰导致指令执行变慢和干扰导致指令执行变多三类，干扰监测从内核角度，针对每一类的典型干扰在运行时进行统计，

当前支持在线统计 schedule latency、throttling、softirq、hardirq、spinlock、mutex 和 smt 干扰，性能开销在 5%以内；

- 鲲鹏内存/Cache QoS 管控机制 MPAM：内存带宽流量和各级缓存占用量，可按照使用量上限/保底/优先级方式进行配置，根据不同业务，以线程为粒度部署不同隔离策略。支持业务资源实时监控，在客户业务层面和线程级别，实时对共享资源的使用情况进行跟踪监控，将资源使用情况反馈给控制策略，形成闭环控制效果。此外，MPAM 联动 SMMU 扩展外设 IO QoS 方案，支持对外围设备和异构加速器 IO 带宽流量进行隔离配置，按设备粒度级别进行资源监控；

- QoS 策略动态配置：提供集群级别的 mpam Qos 管理插件，基于 mpam 提供的 Qos 接口，在插件中根据用户定义，实现为所有节点自动分解各级别优先级，并根据用户的声明自动设置在离线任务 mpam Qos 优先级，从而实现在混部场景下对资源的充分利用：在线业务繁忙时自动抢占离线任务的 llc 及内存带宽，在线业务空闲时自动释放 llc 及内存带宽资源提升离线业务处理性能；

- IO 混部场景 IO QoS 控制器 IOInflight：在高密混部场景中，IO 资源争抢常导致在线业务干扰率高达 40% 以上，而传统的 blk-throttle 静态限流虽能降低干扰却严重浪费磁盘带宽。为此引入 IOInflight 控制器，通过基于时延监控与队列深度的动态节流机制，实现在线业务的精准抢占。该控制器将干扰率与底噪控制在 5% 以内，有效保障在线业务时延，同时使离线业务带宽较传统硬限制模式提升 2 倍。

应用场景

众核服务器场景下，业务容器高密度部署场景，通过降低容器间干扰，提升容器部署密度，进而提升资源利用率。

内存密集型场景：适用于多台虚拟机同时竞争带宽的场景，通过该功能可对每台虚拟机内存带宽占用进行监测和控制，保障对性能要求高的虚拟机业务能正常进行。

Agent 沙箱

功能描述

1. 极速启动与弹性扩展

- 百毫秒级冷启动 —— 基于 Firecracker microVM 技术，比传统虚拟机快数百倍
- 在单集群下，支持从单个沙箱扩展到数百个并发实例，自动弹性伸缩

2. 多语言代码执行

- 支持 Python、JavaScript 等任意编程语言

```python
from e2b_code_interpreter import Sandbox

with Sandbox.create() as sandbox:
    sandbox.run_code("x = 1")
    execution = sandbox.run_code("x += 1; x")
    print(execution.text)  # 输出: 2
```

3. 企业级安全隔离

- 硬件级隔离：每个沙箱运行在独立的 Firecracker microVM 中，拥有专属内核
- 多重安全机制：Jailer 进程、cgroups/namespace 隔离、网络访问控制
- 资源限制：CPU、内存、存储配额防止资源耗尽攻击

4. 完整的环境控制

- 文件系统　　完整的文件读写、上传下载能力
- 进程管理　　执行任意系统命令、安装软件包
- 网络访问　　支持发起 HTTP 请求、调用外部 API
- 动态依赖　　运行时安装 Python/Node.js 等包
- 状态保持　　变量和状态在多次执行间持久化

5. 长会话与高级生命周期管理

- 支持 Pause/Resume/Checkpoint/Fork 等高级操作
- 自动超时清理，也可手动销毁
- 会话存活时间可配置，最长支持数小时长会话
- 优雅停机机制，虚机自动停机启动，高效节省资源

6. 自定义模板

- 通过 Dockerfile 构建预装环境的自定义沙箱

```bash
e2b template init      # 初始化模板
# 编辑 Dockerfile 添加依赖
e2b template build     # 构建
e2b template deploy    # 部署
```

应用场景

•AI Agent 代码执行：让 LLM 能够安全地运行代码、调用工具、处理数据
•自动化工作流：支持定时任务、数据处理、ETL 等批处理场景
•云端 IDE：为开发者提供轻量级、可扩展的云端开发环境

5. 内核创新

openEuler 内核中的新特性

openEuler 24.03 LTS SP4 基于 Linux Kernel 6.6 内核构建，在此基础上，同时吸收了社区高版本的有益特性及社区创新特性。

- openEuler 发布 64K 内核，arm 镜像支持 4K/64K 可选内核安装，在保证 arm 默认安装 4K 内核行为不变的前提下，增加安装 64K 内核的可选行为；默认持平 upstream 社区特性兼容性，基于 64K 特性提升 OS 基础场景性能，64K 内核兼容性约束见附录 3。
- 文件系统支持可编程页缓存：针对大模型推理场景中模型加载 I/O 效率低下问题，实现一种页缓存可编程框架。该框架以透明化方式堆叠于当前文件系统之上，将文件系统缺页事件转发到用户态，使应用可以根据负载特性在用户态对文件系统的缓存策略进行定制，从而对不同模型的加载进行 I/O 效率的显著优化。
- 跨进程零拷贝数据传输机制：提供高效的节点内进程间数据传输接口，应用可将源进程中的给定虚拟内存地址空间上关联的页面映射到目的进程中的虚拟地址空间上，兼容 PMD 大页和 PTE 小页映射。映射后的目的地址访问权限和源地址保持一致。
- FUSE 文件系统支持 io_uring 通信接口：当前 FUSE 架构中，用户态守护进程与内核态驱动模块通过字符设备/dev/fuse 进行通信存在性能瓶颈。主要体现在三个方面：1. 多线程环境中，都需要通过/dev/fuse 设备获取 fuse 请求，容易出现锁争用情况；2. 当前 I/O 请求采用逐条传递机制，缺乏批量处理能力，导致 I/O 密集型任务效率较低；3. 同时发起读写的用户程序和实现文件读写的后端 fuse 线程通常在不同 CPU 核上，会带来较多的核心切换，降低效率。因此，采用 Fuse over io_uring 技术突破上述瓶颈，具体的方案有：1. 通过使用 io_uring 通信接口替代/dev/fuse，

提高并发能力；2. 后端 fuse 服务可以灵活选择走 io_uring 或者传统的/dev/fuse 字符设备通信；3. 在功能上对前台业务程序使用 fuse 文件系统无影响。

- IO 混部场景 IO QoS 控制器 IOInflight：详情见众核高密。
- 基于 Xcall 的 epoll_wait 异步预取：面向特定系统调用的性能优化场景，Dynamic Xcall 通过劫持机制执行定制系统调用，允许用户在不侵入式修改内核的情况下，以应用程序 ELF 文件的粒度进行系统调用劫持，执行定制化系统调用。基于该框架实现了一套非内核侵入式修改的 epoll 异步预取示例内核模块，可以在 Redis 等场景获得一定的性能收益。
- sockmap 加速同主机 tcp 流：Redis、Nginx 等场景存在大量的回环报文并且都是走完整的 tcp 协议栈，耗时较长。通过本特性，利用 bpf 程序将 socket pair 的 ops 替换，将两个 socket 发送和接收队列连接起来，直接将数据包 redirect 到对端，以此来 bypass 协议栈，这样可以减少报文转发流程，降低时延。
- HiSock 功能增强：基于原有 HiSock 能力，针对互联网场景进行功能增强，包括本地加速、包解析逻辑增强、抓包逻辑、地址转换、邻居处理等能力，提升鲲鹏产品在承载高并发、低时延网络负载时的性能表现和吞吐能力。
- copy_from_user 优化特性：通过临时关闭 PAN，允许 ldp 访问用户内存，以单指令 16 字节替代 ldtr 双指令加载，拷贝完成后立即恢复 PAN，大幅减少指令条数。提升拷贝吞吐效率。
- 动态 SMT：为了提高整机 CPU 利用率，会把延迟敏感（Latency-Sensitive, LS）任务和尽力而为（Best-Effort, BE）任务的场景部署到同物理核的两个 SMT 上运行，但 BE 任务和 LS 任务会竞争底层物理核资源导致 LS 任务时延出现大幅劣化。本方案通过针对这一痛点，设计了平衡混部 SMT 干扰管控力度和提升整机 CPU 利用率的方案，优先保证 LS 任务的运行，BE 任务自动被节流并让出微架构资源，同时 BE 任务在主 SMT 核和空闲从核上插空运行来提高 CPU 利用率。
- 网络多路径 RPS 增强特性：网络多路径特性将网卡队列中断按策略绑定到不同 NUMA 节点的 CPU 上，并通过识别业务进程的流量特征，使指定业务的网络流量优先由该进程所在 NUMA 节点的网卡队列接收。RPS 增强特性则将 Loopback 网卡和物理网卡接收的报文，根据配置的 RPS 策略，分散到与业务进程位于同一 NUMA 或 Cluster 的 CPU 核上处理。前者将中断分散到不同 NUMA 节点的网卡队列，后者在此基础上将报文处理在同一 NUMA 或 Cluster 内进一步打散。两者配合

在实现多核 CPU 负载均衡的同时，兼顾 NUMA 和 Cluster 亲和性，有效降低内存访问延迟，提升网络吞吐性能。

- NetKit 是一种直通式虚拟网络设备，核心旨在根治传统 veth pair 在高并发、大吞吐场景下引发的时延阶跃与长尾效应。它彻底解决了因数据包跨命名空间内存复制、频繁触发软中断导致的多核上下文切换损耗，以及无效的 MAC 地址解析与二层报头封装等引致网络延迟的历史包袱。该机制原生支持 eBPF 内核级流量治理，支持自定义多 eBPF 程序的链式动态调度。
- cgroup v2 特性增强：支持开源 cgroup v2 功能，更好的达到 k8s 等开源三方件对于 cgroup 的兼容要求；并且在原有功能基础上，支持 cgroup 级别的异步回收功能；可以解决由于 cgroup 级别的同步内存回收带来的性能损耗，在缓解内存短高峰带来的性能抖动的同时，按照场景需要，提升业务性能；
- sched_ext 可编程调度框架：基于 eBPF 技术实现了系统运行时动态加卸载自定义调度算法的能力，无需重新编译内核，降低了调度器开发周期和门槛，实现对特定工作负载极致优化，内置安全保护机制，BPF 程序异常时自动切回系统默认调度器，避免系统卡死异常。

6. 特性增强

编译器

LLVM for openEuler 编译器

LLVM for openEuler 编译器基于开源 LLVM 软件进行深度打造，是面向服务器互联网行业、数据中心新应用、通算 AI 新场景和视频编解码等高算力场景的高性能编译器。同时在 openEuler 社区打造国内的 LLVM 基线，提供高性能、安全可靠、易创新的 LLVM 下游社区稳定的发行版，已支持主流的系统语言（C/C++）和芯片架构（X86/AArch64/RISCV/LoongArch 等）。

功能描述

LLVM for openEuler 编译器在 openEuler 24.03 LTS SP4 版本引入以下编译特性，提升了编译构建效率，削减 Debuginfo 信息膨胀以及使能 Triton-CPU 全量支持 FlagGems 算子。

Dwarfutils 功能增强，削减 Debuginfo 信息膨胀：ThinLTO 优化是编译器里常见的编译优化手段，传统的编译流程是将每个源码文件各自编译生成目标对象文件，最后再链接生成可执行程序，源码文件之间信息不互通，无法进一步优化，而在 ThinLTO 模式下，源码文件首先编译生成 LLVM IR 格式的文件，在链接时把所有文件整合成一个文件进行深度优化，类似于所有的源码都写在了同一个文件当中，优化更彻底。但是 ThinLTO 优化会大幅度增加编译时长，在构建大型应用的时候尤其明显，ThinLTO Split 技术通过在编译流程中进一步将编译单元进行分块并行编译，加速编译效率。而使能 ThinLTO Split 特性后会使得 Debuginfo 信息膨胀数倍，可能导致链接时寻址溢出，此时可以使用 Dwarfutils 工具消除冗余的 Debuginfo 信息，减少二进制体积。

Triton-CPU 全量支持 FlagGems 算子：Triton 是面向 GPU 的并行编程语言与编译器，它允许开发者使用类似于 Python 的语法编写可编译为 GPU 代码的程序，提供了一种高效灵活的编程方式。Triton 构建在 LLVM MLIR 框架之上，能够充分利用 LLVM 编译器的优化能力。Triton-CPU 是将 Triton 扩展到 CPU 后端，扩展了 Triton 的适用场景，但对于 AArch64 SVE/SME 等指令的适配程度还不佳，也不支持 FlagGems 常用算子。本次更新全量支持了 FlagGems 算子，并亲和了 AArch64 SVE/SME 等指令能力，提高了在 AArch64 架构下的实用性。

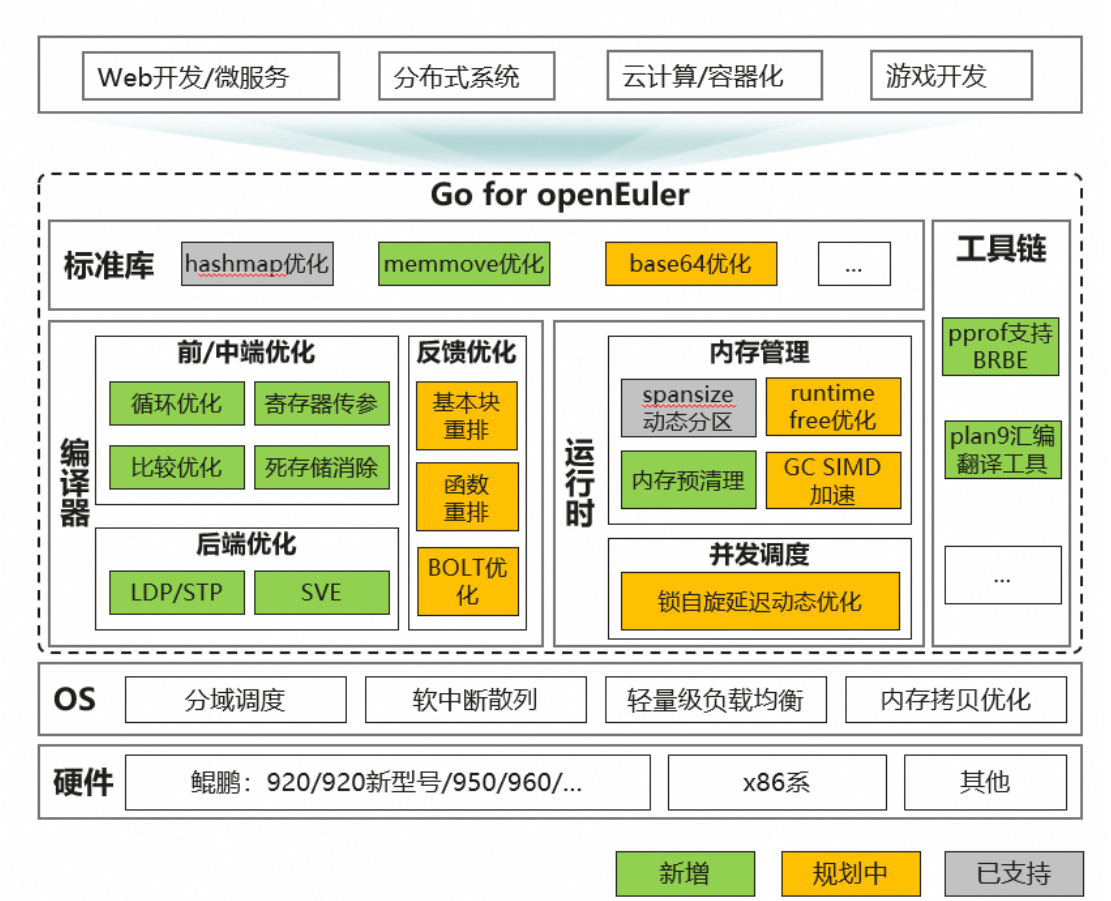
应用场景

使用 LLVM for openEuler 编译器的 ThinLTO Split 技术和 Dwarfutils 工具，能够在不影响调试能力的基础上，大幅提升应用的编译构建效率，帮助开发者提升开发效率；使用 Triton-CPU 编译器能够将原先 GPU 生态的使用 Triton 编写的应用顺利迁移到 CPU 上来，满足开发者多样性算力的需求。

Go for openEuler 编译器

Go for openEuler 是基于开源 Golang 开发，是一款高性能、高可靠、易开发的 Golang 发行版，致力于打造生态兼容、极致体验、亲和 openEuler 的高性能编译器。主要面向云原生、微服务应用等对敏捷开发和运行性能都有要求的容器云场景，围绕业界主流 Go 业务负载进行优化，解决实际业务中由原生 Golang 能力不足导致的性能问题，并适配国产龙芯、鲲鹏等硬件平台，充分释放国产硬件算力。

功能描述



关键特性描述 1：LoopRotate 循环优化增强

原生 LoopRotate 存在优化后跳转次数变多问题，导致性能提升不明显，通过增强 LoopRotate pass，重新调整代码块布局，使其更加紧凑，减少了一次跳转，提升了相关执行效率。

原始循环结构 --------→ 原 looprotate 优化 ---------→ looprotate 增强

关键特性描述 2：bytealg 函数寄存器传参优化

ARM64 平台架构下的汇编函数部分使用 ABI0 的调用约定，导致执行效率低，将使用 ABI0 调用约定升级到 ABIInternal 调用约定，使得优化前 ABI0 使用出入栈的方式传递函数出入参，优化后使用 ABIInternal 约定直接使用寄存器进行函数出入参传递。

关键特性描述 3：LDP/STP 指令优化

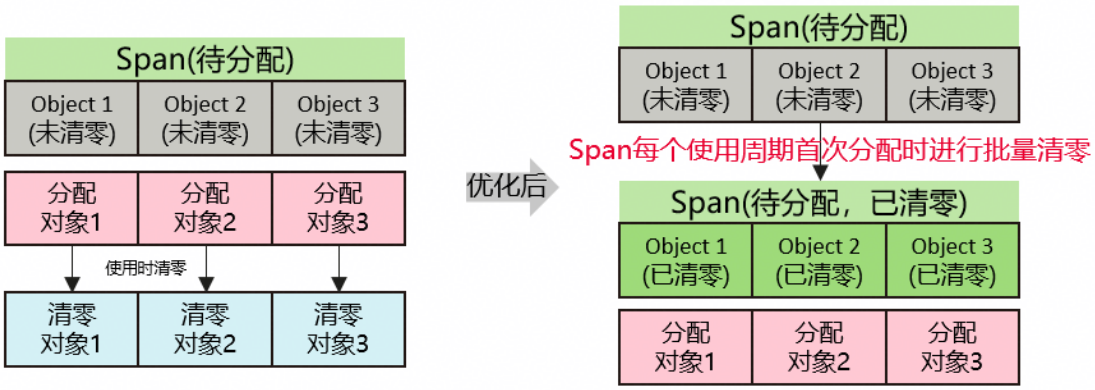
在连续内存地址加载和存储过程中，使用 LDR/STR 效率低，使用 LDP/STP 等鲲鹏支持的高效指令集，提升连续内存的加载和存储效率。

关键特性描述 4：内存块预清理优化

GO 对象释放时仅修改标志位，把清理工作遗留到后续逐个对象分配时进行，造成清理开销频繁，在 nextFree 函数中判断并直接清理整个 mspan，同时对标志位清零，将多次小范围清零操作合并为一次大范围清零操作，带来以下收益：

①优化缓存机制：Go 的内存分配器使用多级缓存 mcache 和 mcentral 管理内存。增大页面缓存粒度，可减少缓存申请次数和锁竞争。

②减少垃圾回收的开销：垃圾回收器需要遍历堆内存以检测未使用的内存块。增大页面后，垃圾回收器处理的内存块数量减少，降低回收的开销。



应用场景

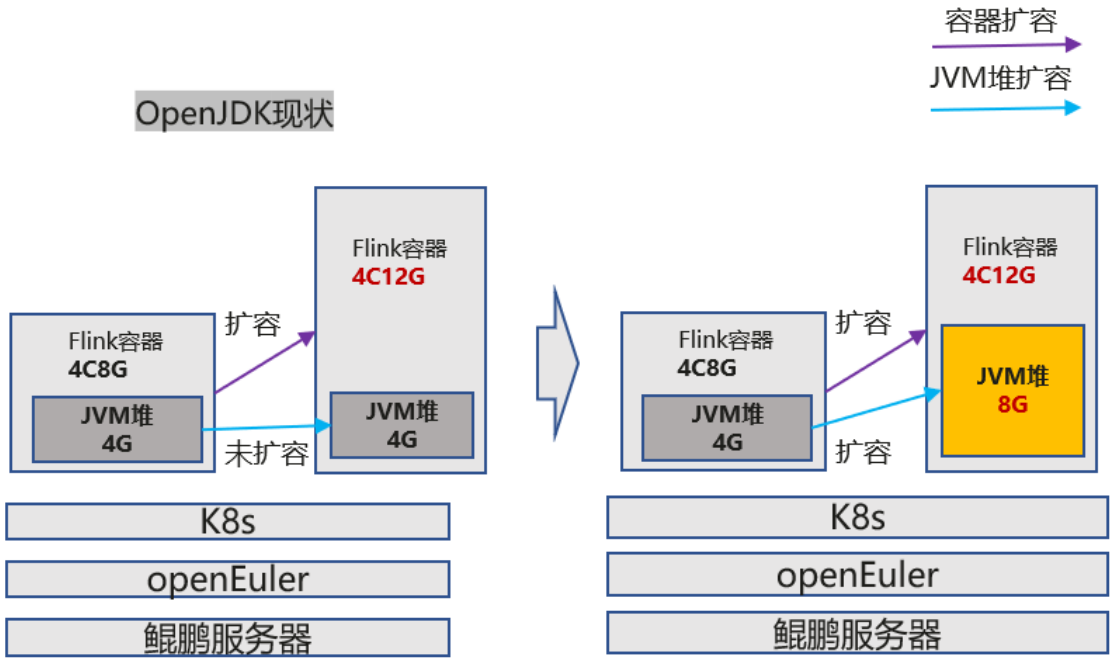
Go for openEuler 是基于 Golang 开发的 Go 基础软件，在 openEuler 等 Linux 环境里应用广泛，应用场景涵盖云原生、分布式存储、云游戏等场景。

毕昇 JDK

毕昇 JDK 支持堆内存扩容

功能描述

容器化部署应用的模式下，大部分客户容器场景下容器资源支持垂直伸缩，当前 OpenJDK8 的最大堆只能在启动时支持修改，无法支持在线动态扩缩，java 应用无法在线使用到容器扩容出的内存，需要 java 应用启动时重新设置最大堆；鉴于此问题，毕昇 JDK 在毕昇 JDK8、毕昇 JDK21 中的 G1GC、PSGC 上实现堆内存上限在线伸缩能力，允许用户在应用运行时动态更新 Java 堆内存的上限，而无需重启 JVM。



应用场景

容器场景业务，在容器在线扩容后需要 java 业务的堆内存大小也支持在线扩容的场景。目前已落地京东大数据场景。

毕昇 JDK8 特性增强

CompactStrings 内存优化

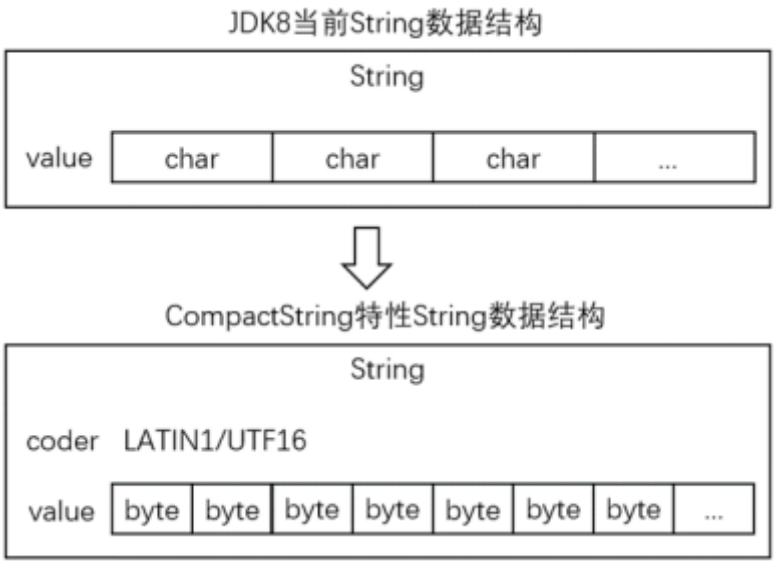
CompactStrings（紧凑字符串）是 Java 9 引入的一项重要内存优化特性，用于改进字符串的存储方式，显著减少内存占用。在 Java 8 及之前版本中，字符串内部使用 char[]字符数组存储，每个字符占用 2 个字节（UTF-16 编码），无论字符串实际内容如何。这导致在处理

大量仅包含 ASCII 字符的字符串时，内存使用效率低下。目前，CompactStrings 已从 JDK17 回合到毕昇 JDK8，进一步提升毕昇 JDK8 整体性能。

功能描述

CompactStrings 将 String 对象内部的表示由一个 UTF-16 的 char 数组改为一个 byte 数组加上一个 encoding 标识字段。这个新的 String 类可以根据字符串中的内容将字符编码为 Latin-1（一个字符 1byte）或者 UTF-16（一个字符 2byte），encoding 标识用于指示这个 String 是如何编码的，具体如下：

- 对于仅包含 Latin-1 字符的字符串，使用 byte[]字节数组存储，每个字符仅占用 1 字节。
- 对于包含非 Latin-1 字符（如中文、日文等）的字符串，仍然使用 char[]存储，每个字符占用 2 字节。
- 每个字符串对象包含一个编码标识（coder）字段，用于指示当前字符串的编码方式：
    - LATIN1（值为 0）：只包含 Latin-1 字符。
    - UTF16（值为 1）：包含其它字符（如中文、日文等）。



应用场景

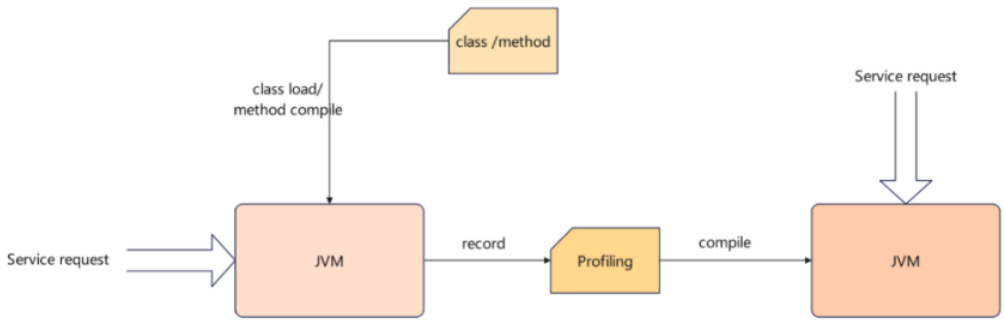
CompactStrings 可以节省字符串内存占用，对于仅包含 Latin-1 的字符串，内存占用节省约 50%。因此，CompactStrings 适用于字符串密集型应用中，可以极大节省内存占用，并进一步减少 GC 频率，提升整体性能。

JProfileCache

Java 应用启动阶段存在热点方法即时编译与业务请求处理对 CPU 资源的竞争问题，可能因为编译延迟导致系统性能爬坡缓慢。JProfileCache 特性基于收集上一次运行的 Profiling 信息再次启动时先触发热点方法编译使用戶Java 进程快速抵达峰值性能。

功能描述

毕昇 JDK JProfilecache 方案将应用程序的发布分成了两个阶段，分别是记录 Profiling 信息和预编译。在记录 Profiling 信息阶段，JVM 会接受线上的请求，同时记录 JVM 即时编译器它所编译方法的信息，并且将这些信息都输出到一个文件之中。等到第二次再去启动的时候，JVM 就可以去读取刚刚所记录的这些方法编译的信息，同时会主动的触发即时编译器编译刚刚记录的热点方法，使得在用户请求到来之前，就把热点方法编译成为性能较高的 Native Code，避免了在用户请求大量进入的时候做编译，这样就能够进一步提高应用程序的性能，节约 CPU 使用率。



应用场景

用于对 JVM 启动、预热敏感的场景，能够加速度过 JVM 启动、预热编译，例如大数据 Spark 场景。

AutoCDS

通过本特性，用户只需要通过指定一组参数，JVM 会自动识别当前所处的阶段，并根据需要启动相应的参数，自动执行相应操作。AppCDS 使能流程由三步简化为一步，减少了手动干预，避免了配置错误的发生。自动化流程不仅减少了操作步骤，还降低了用户配置的复杂度。

功能描述

在创建 VM 时，检查是否启用了 AutoSharedArchivePath 参数。如果未启用，则按照原有逻辑正常启动，没有任何差异；如果启用了 AutoSharedArchivePath，则根据指定路径保存生成的 appcds.lst 和 appcds.jsa 文件。

1. 判断是否存在 lst 文件：首先检查 base_path（保存 lst 和 jsa 文件假设为 base_path）目录下是否已有 lst 文件。若没有，程序将按照以下参数启动：-XX:+UseAppCDS -XX:DumpLoadedClassList=base_path/appcds.lst -Xshare:off 程序运行结束后，将生成 lst 文件。

2. 判断是否存在 jsa 文件：若 lst 文件已存在，则继续检查是否存在 jsa 文件。如果 jsa 文件不存在，则进入生成 jsa 文件的逻辑。此时，通过 fork 子进程的方式启动生成 jsa 文件。

3. 使能 AppCDS：若 jsa 文件已存在，则通过参数直接使能 AppCDS，完成 AppCDS 的启用过程。

应用场景

用于对 JVM 启动感的场景，能够加速度过 JVM 启动、类加载，例如大数据 Spark 场景。

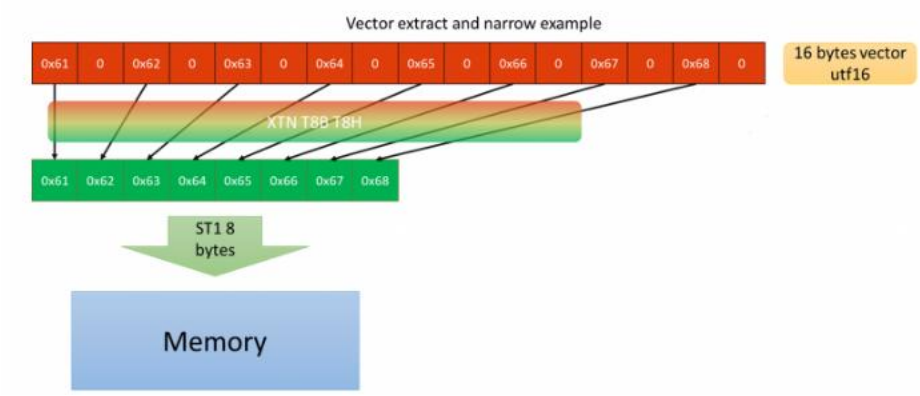
UTF8/16 编解码向量化

该特性当前仅在 AArch64 架构下生效，基于鲲鹏平台的向量化指令能力实现，通过使用 SIMD 指令，对多个字节或字符进行并行加载、转换与写回，从而提升热点路径执行效率。

对于满足向量化条件的输入，走 intrinsic 高性能路径；对于不满足条件的场景（比如 malformed surrogate），则回退至现有标量实现，以保证功能正确性和兼容性。

功能描述

参考 simdutf 的 native 方案，直接 intrinsic 化其实现，对 UTF-8 与 UTF-16 编解码过程进行批量化组织。其核心思想是利用 SIMD 指令一次处理多个字节或多个 16 位字符，基于鲲鹏平台的向量化指令能力实现，通过使用 ld1、st1、xtn、sli、usra 等指令，对多个字节或字符进行并行加载、转换与写回，从而减少传统逐字符处理中的循环次数和条件分支。



应用场景

大数据、微服务等涉及字符串序列化、反序列化的场景。尤其在网路传输时，为了节约传输字节，通常会将字符串压缩为 UTF8 进行传输，接收端收到 UTF8 后展开为适合内存索引的 UTF16。

毕昇 JDK17 特性增强

退优化可观测

JDK17 的 JFR Streaming API 功能，是 JFR 从“事后静态分析”迈向“实时监控”的关键特性。

在传统的 JFR 使用模式中，流程是：记录 -> 停止记录 -> 转储为 .jfr 文件 -> 用 JMC 离线分析。这种模式是“事后分析”，对于排查已经发生的问题非常有效。

Streaming API 引入了一种全新的模式：它允许 Java 应用程序在不中断 JFR 记录、不生成完整 .jfr 文件的情况下，实时、持续地从 JVM 内部订阅和消费 JFR 事件流。

功能描述

在使用 Streaming API 的时候，通过本功能可以在 ****** 处获取当前时间之前的一段 jfr event，例如退优化事件。

```
// 1. 创建一个 RecordingStream
RecordingStream rs = new RecordingStream();
// 2. 启用我们感兴趣的事件并配置设置
rs.enable("jdk.GCPhasePause").withPeriod(Duration.ofSeconds(1));
rs.enable("jdk.Deoptimization").withPeriod(Duration.ofSeconds(1));
// 3. 订阅特定事件并设置事件处理器（回调函数）
rs.onEvent("jdk.GCPhasePause", event -> {
// 从事件中读取字段
Duration duration = event.getDuration("duration");
String name = event.getString("name"); // 例如 "GC Pause"
    **********
    shell:    jcmd JFR.start delay=-1 filename=xxx.jfr
    **********
});
// 4. 启动流（这是一个非阻塞调用）
rs.startAsync();
```

应用场景

在线监控和事后分析配合的模式，在事后分析的 jfr 文件中，能够包含当前时间之前的一段时间内的所有 jfr 事件。

元数据压缩

Java 对象的存储会有额外的开销，即对象头，它用于存储与该对象相关的元数据信息。伴随存活对象的增多，以及大量小对象的存在，对象头占用空间问题也会愈发严重。压缩对象头特性由此而来，该特性隶属于 OpenJDK Lilliput 子项目，旨在研究如何将 Hotspot JVM 中的 Java 对象头从 128/96 bits 降低到 64 bits，从而减少 Java 堆内存（Heap Memory）占用，并且通常还能提升应用程序的工作负载性能。

功能描述

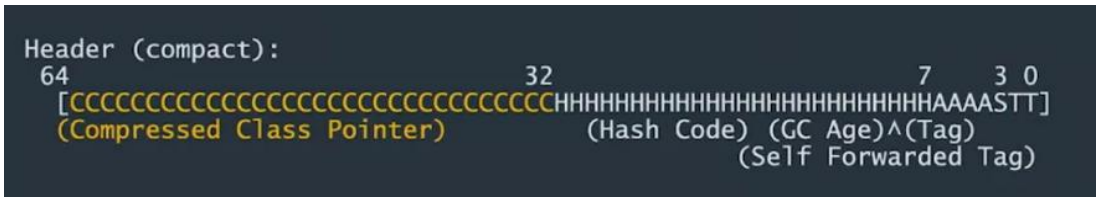
对象头通常包含两个主要部分：MarkWord 和类指针（Class Pointer），如果是数组对象还包括数组长度。其中 MarkWord 主要包含 GC 年龄、Hash 值、锁等运行时信息。原有的 Mark 的位分布如下：

锁状态	MarkWord 位分布（64位）	标志位
无锁 (Normal)	unused:25 | identity_hashcode:31 | unused:1 | age:4 | biased_lock:0 (25位未用) | (31位哈希码) | (1位未用) | (分代年龄) | (偏向锁禁用)	01
偏向锁 (Biased)	thread:54 | epoch:2 | unused:1 | age:4 | biased_lock:1 (54位线程ID) | (2位纪元) | (1位未用) | (分代年龄) | (偏向锁启用)	01
轻量级锁 (Thin)	ptr_to_lock_record:62 (62位指向线程栈锁记录的指针)	00
重量级锁 (Heavy)	ptr_to_monitor:62 (62位指向监视器Monitor的指针)	10
GC 标记 (Marked)	gc_info:62 (GC过程用到的信息)	11

该特性利用了正常情况下 MarkWord 高位未被使用的特点，将类指针移动到 MarkWord 的高位，并通过缩短 HashCode 和修改标志位等方式，将对象头压缩至 64 位。需要注意的是，放置在 MarkWord 中的类指针必须是压缩后的类指针。

该特性针对对象头做了以下修改:

- 将类指针部分移到 MarkWord 的高位，类指针仅支持压缩类指针（基本可以满足绝大数应用）
- HashCode 减少到 25 位，这对于超大 Hashtable 的性能可能会存在劣化
- 增加了 1 位 Self Forwarded Tag，用来表示 GC 复制阶段失败的情况（原始是将 MarkWord 替换成指向自己的指针）



原有的 Array 对象头，占用空间为 96/128bit（是否开启压缩类指针），Array 使用 16byte 对齐。使用压缩对象头后，Array 对象为指针,bool,byte,char,short,int,float 类型时，mark/klass 和 Array 长度占用 12byte，此时对象头对齐策略同时也改到到 12byte 来减少内存占用。举个例子，此时若是长度为 1 的 int 数组，原来需要占用 24byte，现在仅需 16byte，相对节省了三分之一的内存。如果程序中存在大量这样的数据，确实会节省不少的内存。

应用场景

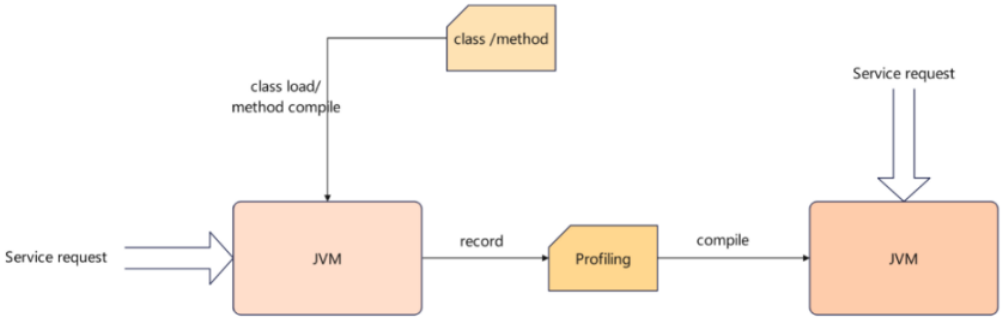
适用于有大量小对象分配的场景，小对象分配越多，对象头相对占比越大，本优化效果越明显。

JProfileCache

Java 应用启动阶段存在热点方法即时编译与业务请求处理对 CPU 资源的竞争问题，可能因为编译延迟导致系统性能爬坡缓慢。JProfileCache 特性基于收集上一次运行的 Profiling 信息再次启动时先触发热点方法编译使用戶Java 进程快速抵达峰值性能。

功能描述

毕昇 JDK JProfilecache 方案将应用程序的发布分成了两个阶段，分别是记录 Profiling 信息和预编译。在记录 Profiling 信息阶段，JVM 会接受线上的请求，同时记录 JVM 即时编译器它所编译方法的信息，并且将这些信息都输出到一个文件之中。等到第二次再去启动的时候，JVM 就可以去读取刚刚所记录的这些方法编译的信息，同时会主动的触发即时编译器编译刚刚记录的热点方法，使得在用户请求到来之前，就把热点方法编译成为性能较高的 Native Code，避免了在用户请求大量进入的时候做编译，这样就能够进一步提高应用程序的性能，节约 CPU 使用率。



应用场景

用于对 JVM 启动、预热敏感的场景，能够加速度过 JVM 启动、预热编译，例如大数据 Spark 场景。

AutoCDS

通过本特性，用户只需要通过指定一组参数，JVM 会自动识别当前所处的阶段，并根据需要启动相应的参数，自动执行相应操作。AppCDS 使能流程由三步简化为一步，减少了手

动干预，避免了配置错误的发生。自动化流程不仅减少了操作步骤，还降低了用户配置的复杂度。

功能描述

在创建 VM 时，检查是否启用了 AutoSharedArchivePath 参数。如果未启用，则按照原有逻辑正常启动，没有任何差异；如果启用了 AutoSharedArchivePath，则根据指定路径保存生成的 appcds.lst 和 appcds.jsa 文件。

1. 判断是否存在 lst 文件：首先检查 base_path（保存 lst 和 jsa 文件假设为 base_path）目录下是否已有 lst 文件。若没有，程序将按照以下参数启动：

-XX:+UseAppCDS -XX:DumpLoadedClassList=base_path/appcds.lst -Xshare:off 程序运行结束后，将生成 lst 文件。

2. 判断是否存在 jsa 文件：若 lst 文件已存在，则继续检查是否存在 jsa 文件。如果 jsa 文件不存在，则进入生成 jsa 文件的逻辑。此时，通过 fork 子进程的方式启动生成 jsa 文件。

3. 使能 AppCDS：若 jsa 文件已存在，则通过参数直接使能 AppCDS，完成 AppCDS 的启用过程。

应用场景

用于对 JVM 启动感的场景，能够加速度过 JVM 启动、类加载，例如大数据 Spark 场景。

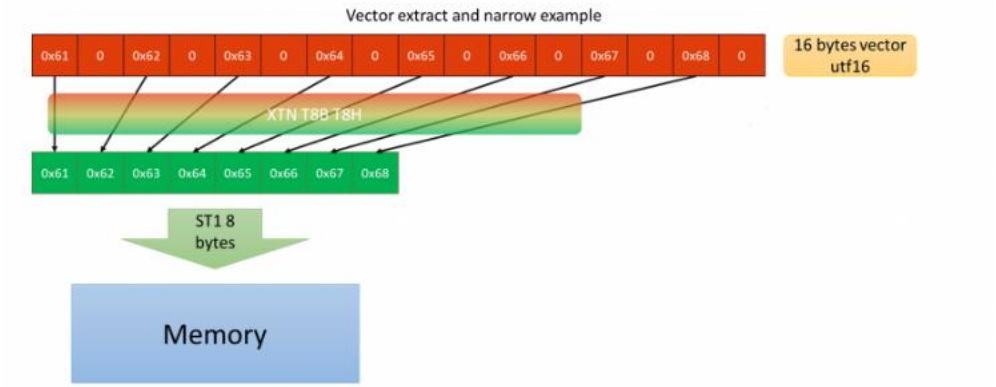
UTF8/16 编解码向量化

该特性当前仅在 AArch64 架构下生效，基于鲲鹏平台的向量化指令能力实现，通过使用 SIMD 指令，对多个字节或字符进行并行加载、转换与写回，从而提升热点路径执行效率。对于满足向量化条件的输入，走 intrinsic 高性能路径；对于不满足条件的场景（比如 malformed surrogate），则回退至现有标量实现，以保证功能正确性和兼容性。

功能描述

参考 simdutf 的 native 方案，直接 intrinsic 化其实现，对 UTF-8 与 UTF-16 编解码过程进行批量化组织。其核心思想是利用 SIMD 指令一次处理多个字节或多个 16 位字符，

基于鲲鹏平台的向量化指令能力实现，通过使用 ld1、st1、xtn、sli、usra 等指令，对多个字节或字符进行并行加载、转换与写回，从而减少传统逐字符处理中的循环次数和条件分支。



应用场景

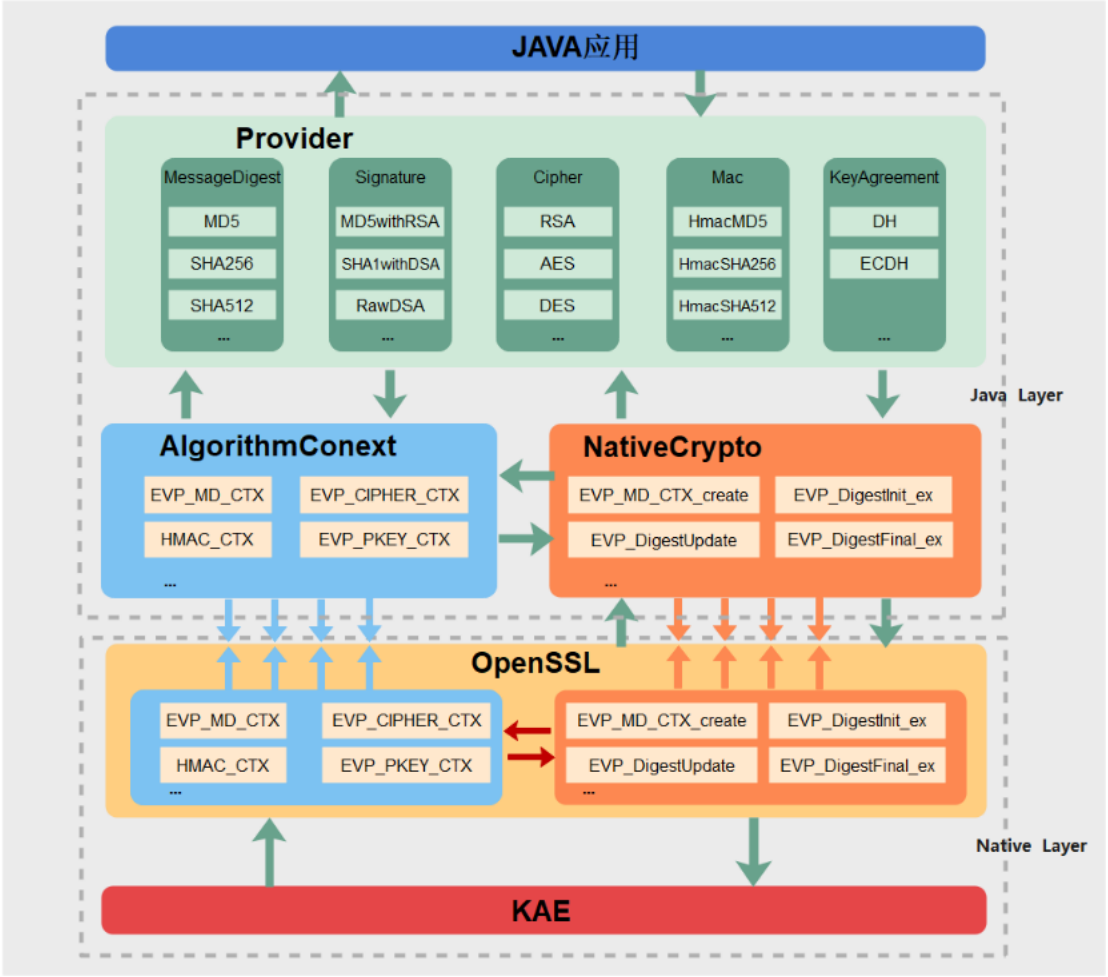
大数据、微服务等涉及字符串序列化、反序列化的场景。尤其在网路传输时，为了节约传输字节，通常会将字符串压缩为 UTF8 进行传输，接收端收到 UTF8 后展开为适合内存索引的 UTF16。

毕昇 JDK21 特性增强

KAE 加速器支持

KAE 加解密是鲲鹏加速引擎的加解密模块，其依托修改 OpenSSL 库实现对底层 KAE 硬件的调用。

因此应用代码只需调用 OpenSSL 库，KAE 作为 OpenSSL 的底层引擎提供加速能力。KAEProvider 介于应用和 OpenSSL 之间，提供一种便利的使用 OpenSSL 库及使能 KAE 加速的方式。



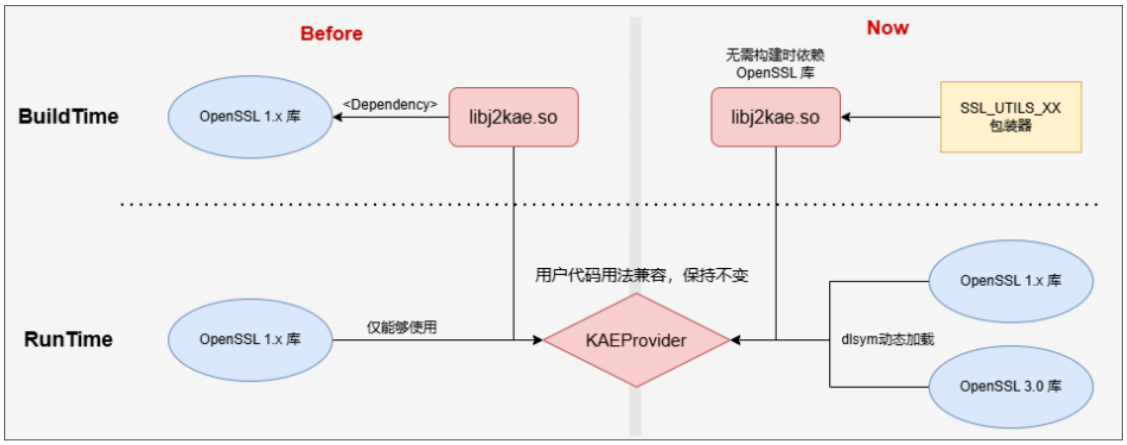
KAE 压缩模块支持 ZLIB、GZIP 数据格式压缩，压缩带宽大大提升。现 Java 应用中或多或少都会使用到压缩解压缩特性，尤其在一些数据存储、http 等场景压缩 CPU 占比可能高达 70%+，开源 zlib 压缩速度成为瓶颈，若 Java 应用可以使能 KAE-zlib 将极大提高业务的性能。

功能描述

过去的版本里 KAEProvider 支持 OpenSSL1.x 版本。随着 2021 年 OpenSSL3.0 版本发布，在架构上、可扩展性、安全性上都有所提升，是一个重大版本更新，后续进行安全通信开发的用户很可能会选择采用升级版本。然而 OpenSSL1.x 到 OpenSSL3.0 一部分 API 发生了废弃、新增或变化，因此 KAEProvider 无法直接支持使用，需要进行代码适配。

本特性添加后 KAEProvider 构建结果不变，但构建过程中不再依赖构建环境本地的 OpenSSL 库环境(之前构建目标产物 libj2kae.so 时需依赖构建环境下的 OpenSSL1.x 环境)，使用哪个 OpenSSL 版本会在程序运行时动态判断和加载。

KAEProvider 用法保持不变，并且用户使用本增强特性时也需要手动打开（修改 OPENSSL_ENGINES 环境变量和 kae.useOpenSSLVersion 配置），不修改原有加解密逻辑，不影响 JDK 原有功能和使用方式。



对于 KAE zip 支持，JDK 中将动态库拆分：拆分 libzip.so 为 libzip.so 和 libz.so，使其分别包含 JNI 和 zlib 代码。使能 KAE-zlib：通过-DGZIP_USE_KAE 参数来控制使能，压缩解压缩过程 Java 侧不做文件头尾的处理，使其压缩数据结构的生成和解析均依赖于底层的 zlib 库的 ZLIB、GZIP 压缩算法来完成。

应用场景

涉及加解密、压缩解压缩的场景，KAE Provider、KAE zip 对于支持的算法能够有效提升性能。

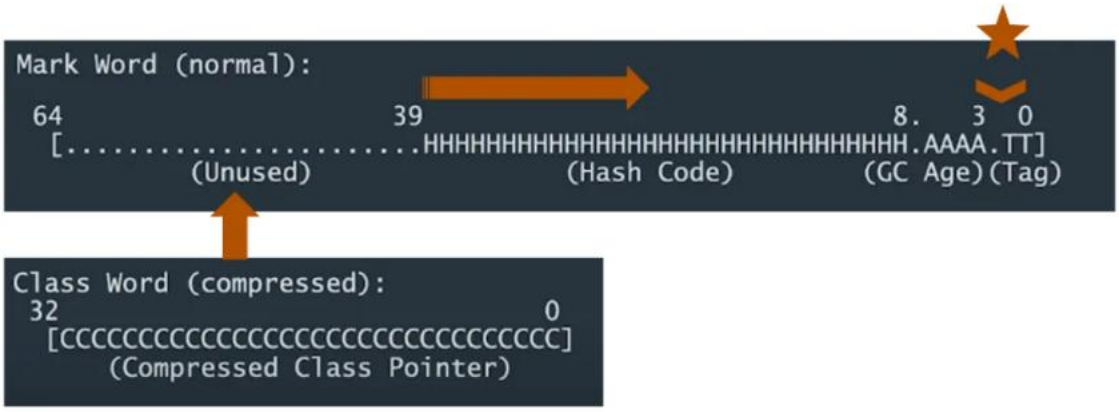
元数据压缩

Java 对象的存储会有额外的开销，即对象头，它用于存储与该对象相关的元数据信息。伴随存活对象的增多，以及大量小对象的存在，对象头占用空间问题也会愈发严重。本特性在 Aarch64 平台下将对象头从 128/96 bits 降到 64bits。

功能描述

在 64 位 Hotspot 中，Java 对象具有 128 位的对象头：一个 64 位的 MarkWord 字和一个 64 位的类指针。对象的平均大小通常为 5-6 个字，其中两个字始终由对象头占用。该特性利用了正常情况下 MarkWord 高位未被使用的特点，将类指针移动到 MarkWord 的高位，并通过缩短 HashCode 和修改标志位等方式，将对象头压缩至 64 位。

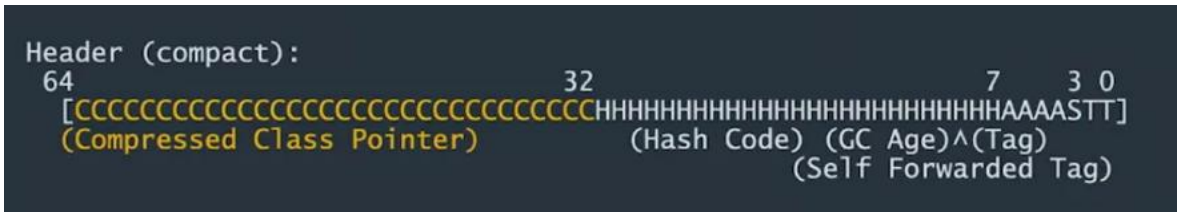
修改前的对象头：



该特性针对对象头做了以下修改:

- 将类指针部分移到 MarkWord 的高位，类指针仅支持压缩类指针（基本可以满足绝大数应用）。
- HashCode 减少到 25 位，这对于超大 Hashtable 的性能可能会存在劣化。
- 增加了 1 位 Self Forwarded Tag，用来表示 GC 复制阶段失败的情况（原始是将 MarkWord 替换成指向自己的指针）。

修改后对象头如下:



因此我们就可以显著降低内存压力，实现:

- 减少堆使用量
- 更高的对象分配率
- 降低 GC 活性
- 对象更紧密地排布将带来更好的缓存局部性

应用场景

本方案适用场景较为通用，对于内存占用的减少根据应用程序的实际数据情况而变化。本特性针对小对象分配多的场景下能够在端到端体现出较好的效果，在大数据部分场景下有收益。

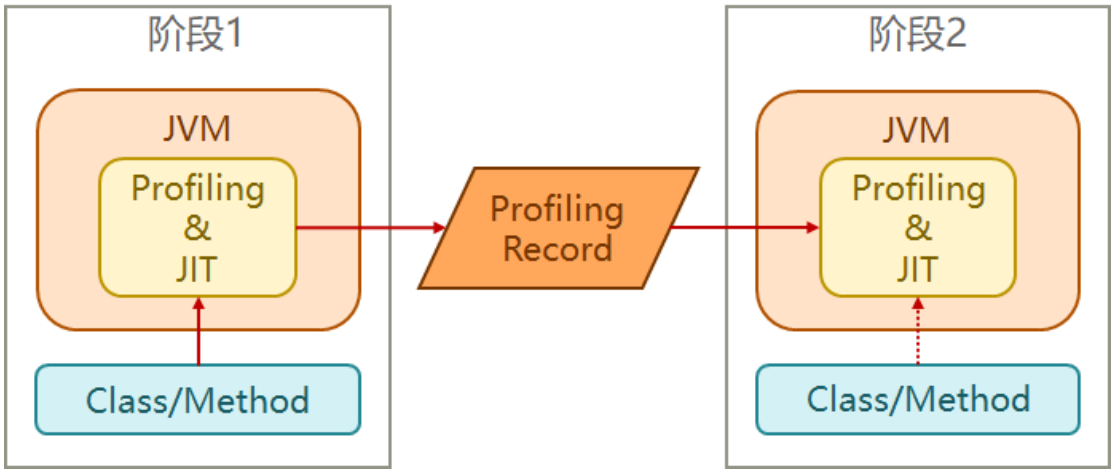
JIT 预热增强

本特性旨在使能 Java 进程启动后快速抵达峰值性能。Java 进程从拉起到最终到达峰值，存在一个较为漫长的过程。其中解释器->采样->编译 C1->进一步采样->编译 C2 的采样与在线 JIT 编译流程是应用预热过程的重要一环。本特性通过压缩该部分流程，有效加速 VM 启动与预热，助力应用快速达峰。

功能描述

毕昇 JDK JProfilecache 方案将应用程序的发布分成了两个阶段，分别是：

- **记录阶段**：在程序运行结束时，把热点方法的 Profiling 信息(主要是方法调用次数，回边次数)及该方法所属类(包括父类)输出到指定文件中。
- **预编译**：后续启动的时候，JVM 读取热点方法所在类并进行预加载，同时会把热点方法加入到编译队列进行提前编译，使得在用户请求到来之前，跨过 Profiling 阶段，就把热点方法编译成为性能较高的 Native Code，减少热点方法的预热开销。



应用场景

适用于对启动速度有要求的云原生场景、涉及反复拉起 driver/executor 进程的 Spark 大数据启动敏感场景。本特性在此类场景下能够有效加速 VM 启动与应用预热，实现性能快速达峰。

AI 编译器

ANNC（Accelerated Neural Network Compiler）是专注于加速神经网络计算的 AI 编译器，聚焦于通过计算图优化，高性能融合算子生成和高效的代码生成和优化能力，加速推荐

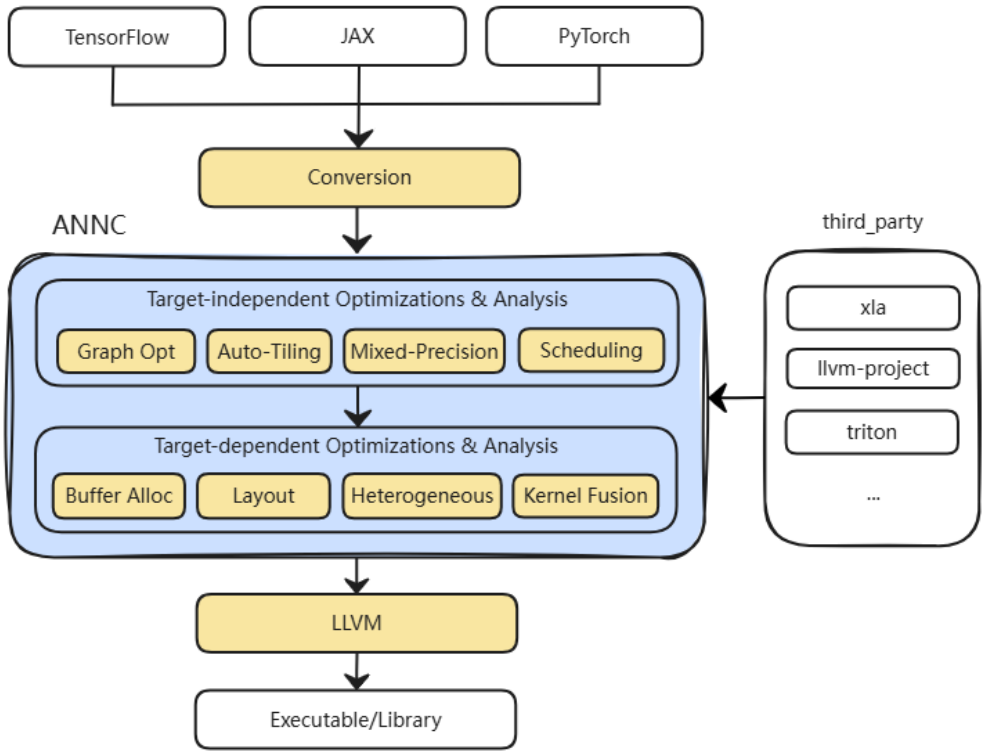
和大语言等模型的推理性能，并且从架构设计角度支持业界主流开源推理框架和不同硬件后端的接入，提高软件的可扩展性。

功能描述

计算图优化是通过优化神经网络的计算流程，从算法角度减少冗余操作、混合精度改写和自动子图调度优化计算负载和提高缓存利用率，从硬件架构角度优化张量数据布局、算子融合和转换、子图切分调度，进一步优化负载，充分利用硬件资源。

针对推荐模型 Embedding 层瓶颈突出、热点占比高的特点，ANNC 通过提供融合算子、常量折叠等特性，降低独立算子执行带来的内存搬移操作，减少不必要的访存，从而优化推理性能。

高性能融合算子库生成和对接包括前端计算图模式识别和转换，高性能算子库查询和对接，以及算子库自动生成三部分，从汇编指令层面上通过数据预取、模型并行和新型指令集应用等优化手段，减少访存，提高并行计算效率。



ANNC 旨在通过图编译优化和高性能算子生成和对接，提高 AI 推理速度和降低功耗，达到提高用户单位成本的推理效率的目的，同时通过软件兼容性和易用性设计减少用户的运营成本和环境影响。

应用场景

当前版本的 AI 编译器主要聚焦于图编译优化和算子库选择调用，面向主流搜索推荐系统中的粗排、精排模型能够取得较大收益，尤其是面向特征处理复杂的嵌入层网络，图编译优化能够取得更好的效果。

同时，ANNC 也预留了大语言模型等未来场景的性能优化途径，对接 LLM 推理框架，结合现有的图算融合、内存布局优化技术，以及 GEMM 等核心的性能优化手段，减小推理时延，提高推理吞吐。

安全

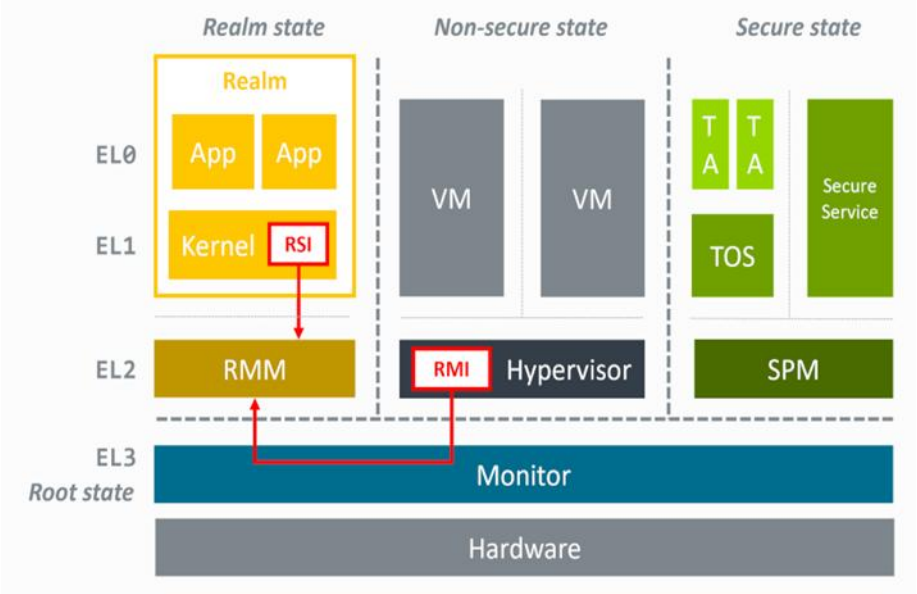
CCA 机密计算

ARM CCA（Confidential Computing Architecture）是 ARM v9 新引入的机密计算架构规范，旨在为下一代计算设备定义标准化的机密计算解决方案。CCA 引入了全新的 Realm 域作为可信执行环境，来保护正在使用中的数据和代码的机密性和完整性，即使面对拥有特权的基础设施软件或云服务提供商，也能得到有效保护。

openEuler 基于 ARM CCA 机密计算架构规范，实现了 OS 相关组件（KVM、QEMU、libvirt、Guest kernel）对 CCA 的支持，提供了原生支持 Realm 机密虚机的社区版本，满足基于 Realm 机密虚机保护使用中数据的安全诉求，同时提供了兼容传统应用生态及虚机管理软件的易用性。

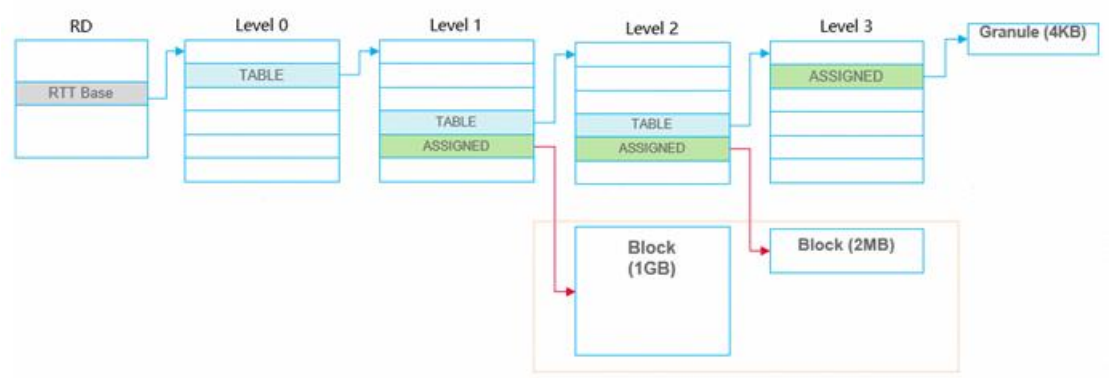
功能描述

ARM CCA 通过以下核心组件协同工作，构建一种隔离的、受保护的执行空间，在代码执行和数据访问方面与正常世界完全隔离，成为 Realm 机密域。



- **Realm 机密域**：Realm 是 CCA 的核心抽象，它是一种与正常世界（Non-secure）和安全世界（Secure，原来的 Trustzone）并行的新类型执行环境。Realm 是硬件隔离的，专为托管敏感代码和数据而设计。它独立于主机操作系统和 Hypervisor，它们可以管理 Realm 但无法访问其内部内容。
- **动态管理**：Hypervisor 可以应客户要求动态创建 Realm，并为其分配内存和 CPU 资源。但在 Realm 初始化后，Hypervisor 会将其控制权移交给一个受保护的安全虚拟化模块 RMM（Realm Management Monitor），此后 Hypervisor 便无法访问 Realm 内的秘密。
- **内存管理**：CCA 扩展了系统内存管理单元（MMU），使其能够识别和隔离 Realm 内存。任何从 Realm 外部（包括 Hypervisor）发起的访问尝试都会被硬件阻断，从而确保数据的机密性。
- **远程证明**：每个支持 CCA 的处理器都有一个基于硬件的唯一身份标识。当 Realm 启动时，它可以生成一份由硬件密码学签名的证明报告（Attestation Token）。用户可以获得这份报告，并验证其签名及组件度量值，从而确信他们的工作负载正在一个真实的、未被篡改的 ARM CCA 环境中运行。

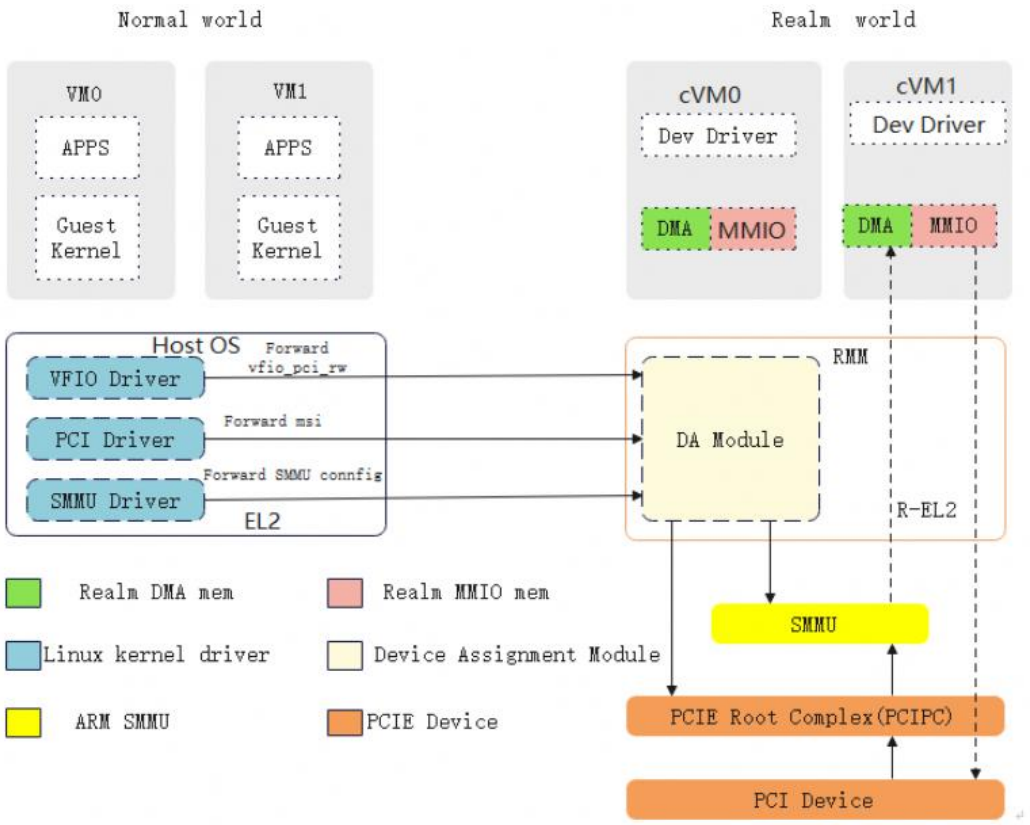
CCA 机密虚机默认启动流程中，RMM 组件通常以 4K 固定粒度来管理受保护内存的状态，并基于该粒度创建 stage2 页表，该流程存在页表遍历成本高、TLB 压力大、RMM 接口调用频繁等关键性能开销。基于 1G/2M 大页创建 CCA 机密虚机 stage2 页表可以显著降低前述性能开销，openEuler 支持根据 CCA 机密虚机 IPA 地址对齐及物理页连续情况选择创建 1G or 2M 页映射：



CCA 基于 1G/2M 创建 stage2 页表存在如下优势：

- **大页**（1GB/2MB）可在更高层级完成映射，缩短 CCA 机密虚机 stage2 页表遍历路径，降低页表访问延迟。
- **提高 TLB 命中率：**大页占用更少的 TLB 项，从而减少 CCA 机密虚机 TLB miss 和页表查找开销。
- **加速 Realm 启动与内存初始化：**批量映射大块物理内存可显著缩短 CCA 机密虚机创建时间。

CCA 机密虚机基于华为鲲鹏处理器通过预埋在 PCIe Root Complex 里的 PCIE 保护组件（PCIPC）实现 PCIe 设备机密直通，在机密计算场景下，支持使能 PCIPC 保护的 PCIe、设备直通 Realm 域，数据免中转免拷贝，以此保证设备数据链路的机密性。CCA 机密设备直通架构图如下所示：



CCA PCIe 设备机密直通存在以下优势:

- 安全隔离

使能 CCA 能力后 PCIPC 保护的安全设备仅支持 Realm 侧访问，Host 侧软件无法访问。

- 高性能

Realm 机密虚拟机设备直通，相比业界加解密方案，数据面无损耗。

- 易用性

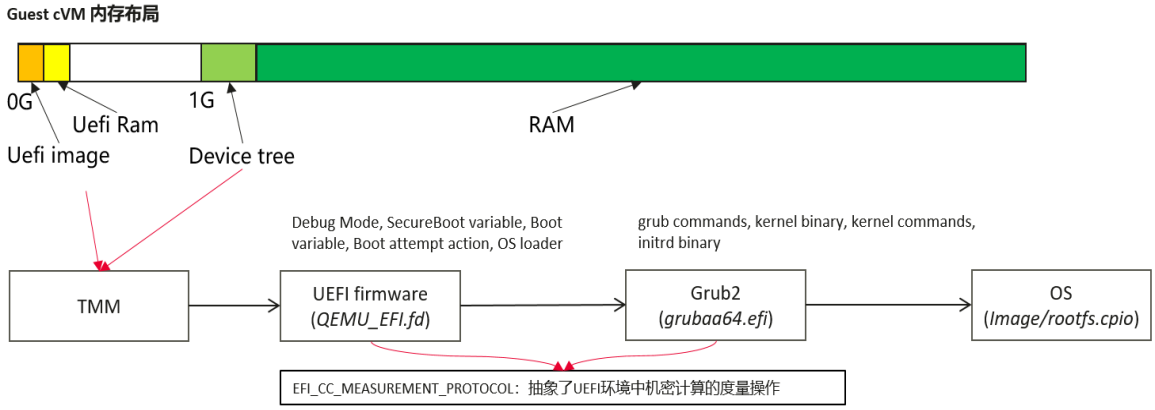
兼容现有开源 OS，无需修改开源 OS 内核驱动代码。

应用场景

机密虚机主要应用于云计算环境，为核心工作负载提供高级别安全隔离。它使得客户能在不受信的公有云上处理敏感数据（如金融记录、医疗健康信息或知识产权），并确保其机密性和完整性连云服务商也无法窥探。这有效满足了数据主权、监管合规和跨机构隐私数据协作（如联合建模与分析）的关键需求。

virtCCA 特性增强

功能描述



当前 virtCCA 架构在启动方式上存在特定约束：其仅支持 **kernel 与 rootfs 分离的启动模式**（即内核镜像与根文件系统分别挂载）。然而，在主流云平台环境中，虚拟机的启动流程普遍依赖 **GRUB 引导机制**，这要求将 UEFI 固件（如 EDK2）、内核（Kernel）及初始内存文件系统（initramfs）整合至单一磁盘镜像（如 QCOW2 格式）中。功能要点包括：

1. 单镜像封装

(1) 将 EDK2 固件、GRUB 引导程序、内核（Kernel）及 initramfs 整合至单一 QCOW2 磁盘镜像，形成完整启动栈。

(2) GRUB 通过配置文件（grub.cfg）定位内核路径，要求内核与 initramfs 必须位于同一文件系统（如 EXT4/XFS）。

2. 安全信任链传递

(1) Secure Boot 机制：EDK2 验证 GRUB 及内核的数字签名，确保启动组件未被篡改。

(2) 硬件资源协同：依赖 UEFI 运行时服务枚举硬件设备，为虚拟机管理程序（如 KVM）提供虚拟化资源池

3. 云原生优化

(1) 支持快照克隆、根文件系统动态扩容（依赖 initramfs 中的 cloud-init 工具）特性。

应用场景

云环境中采用 UEFI（Unified Extensible Firmware Interface）启动模式，已成为现代虚拟化架构的核心技术，virtCCA 架构支持 UEFI 启动，扩展了机密虚机的应用场景：

1. 快速实例启动与弹性伸缩

UEFI 通过**并行硬件初始化**（如 CPU、内存、存储设备同步检测）显著缩短启动时间。

2. 大容量磁盘支持

UEFI 依赖 **GPT 分区表**，突破传统 MBR 的 2TB 磁盘限制，支持**百 TB 级云盘**（如阿里云 ESSD 云盘），满足大数据存储（如 HDFS）、AI 训练等场景需求。

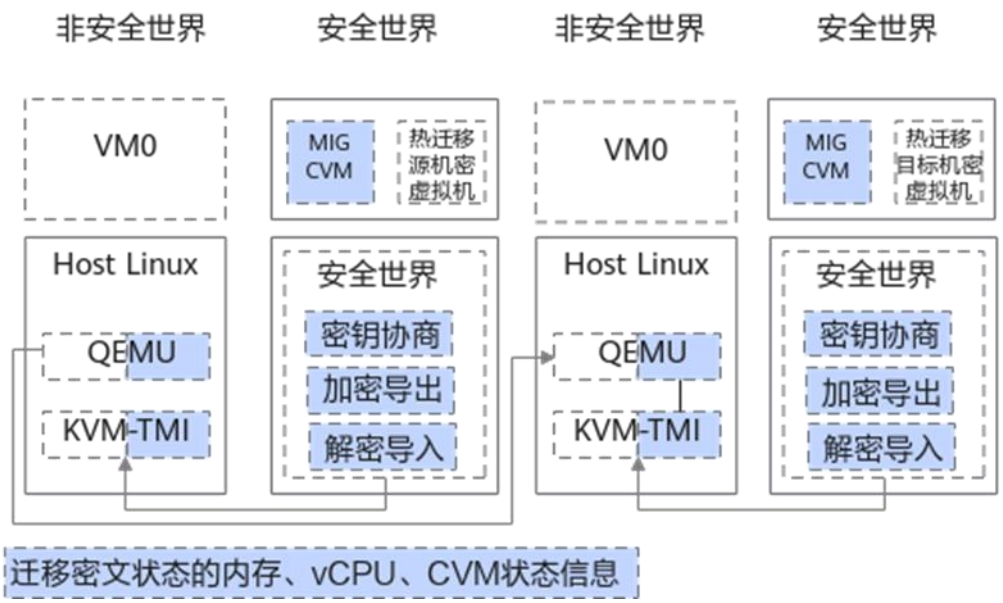
3. 自动化运维与批量部署

镜像标准化：云服务商提供的 UEFI 兼容镜像默认启用 GPT 分区，简化用户部署流程。

virtCCA 机密虚机热迁移

功能描述

机密虚拟机热迁移是指在机密虚机（CVM）业务运行不中断的情况下，将其从一个机密计算环境安全地迁移至另一个经过验证的机密计算环境，并确保机密虚机中的敏感数据在迁移前、中、后始终处于加密或隔离保护之下。



virtCCA 机密云主机热迁移通过迁移管理虚机 MigCVM 和安全世界 Hypervisor TMM 组件实现热迁移过程中机密虚机状态的机密性和完整性，包括如下功能要点：

- 迁移源平台需要对目标平台进行身份验证，TMM 组件需要对 MigCVM 进行可信度量；
- 身份验证通过后，源平台和目标平台 MigCVM 完成迁移密钥协商；
- 迁移源平台和目标平台建立加密安全会话，确保迁移数据的安全；
- MigCVM 对迁移状态进行管理和跟踪，确保迁移过程状态安全，管理异常状态；

- TMM 组件负责导出、导入 virtCCA 机密虚机内存状态及 VCPU 状态;
- TMM 组件基于迁移密钥对机密虚机状态数据进行加密，并对状态数据进行完整性校验。

应用场景

主流云主机管理平台要求云主机支持热迁移能力，通过热迁移能力实现在服务器运维场景下将云主机从一台服务器迁移至另一台服务器，热迁移过程中保证客户业务的连续性。VirtCCA 机密云主机热迁移主要包括以下应用场景:

- 硬件、系统维护与升级

物理服务器进行固件升级、硬件更换或系统更新时，可将 CVM 虚拟机实时并安全地迁移到其他主机，业务零中断。

- 负载均衡与性能优化

当某台主机 CPU、内存负载过高时，可以将指定资源消耗较大的 CVM 虚拟机到负载较低的主机，灵活调度，确保性能整体最优或局部最优。

Kuasar 机密容器

功能描述

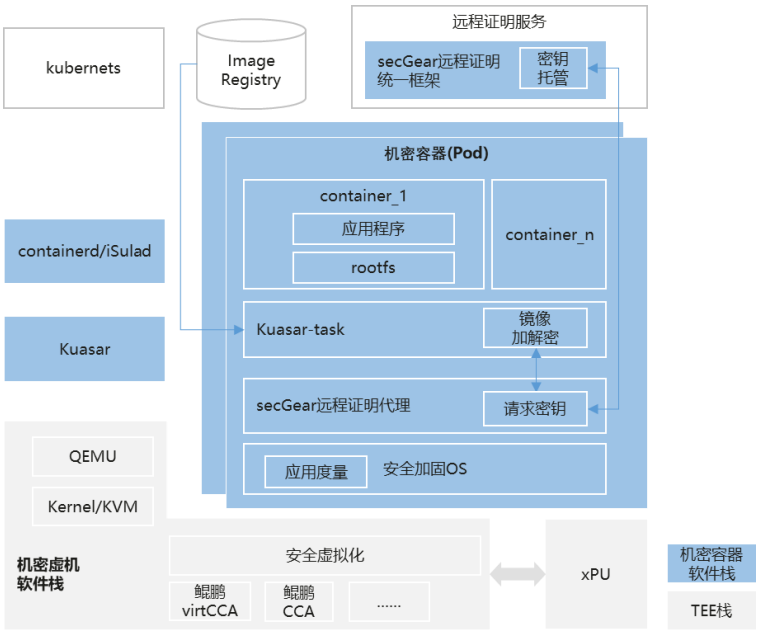
Kuasar 统一容器运行时在支持安全容器的基础上添加了对机密容器的支持。用户可以通过配置 iSulad 的运行时参数，完成对 Kuasar 机密容器的纳管。

当前 Kuasar 机密容器使用 iSulad+Kuasar 方案，提高了启动速度，极大降低了内存底噪。一方面 Sandbox API 的实现，使得创建容器不再单独创建 pause 容器，节省了准备 pause 容器镜像快照的时间；另一方面得益于 1:N 的管理模型，Sandboxer 进程常驻，从而节省了冷启动 Shim 进程的时间，这使得容器的启动速度大大提升，带来与 Pod 数成正比的内存收益。最后，Kuasar 使用 rust 实现，相比 golang，内存更安全，语言本身也带来了一些内存收益。

支持功能特性:

- 支持 iSulad 容器引擎对接 Kuasar 机密容器运行时，兼容 Kubernetes 云原生生态。
- 支持基于 virtCCA 的机密硬件，允许用户在鲲鹏 virtCCA 可信执行环境中部署机密容器。

- 支持 secGear 远程证明统一框架，遵循 RFC9334 RATS 标准架构，允许在机密计算环境中运行的容器向外部的受信任服务证明其可信性。
- 支持在机密容器内部拉取并解密容器镜像，保护容器镜像的机密性和完整性。



应用场景

Kuasar 机密容器在满足客户数据安全的诉求下，兼容云原生生态，确保用户的机密应用具备高可用性、弹性伸缩、快速交付 等云原生优势，在 AI 保护、数据可信流通、隐私保护等机密计算的业务中有广泛的应用场景。

Global Trust Authority 远程证明

功能描述

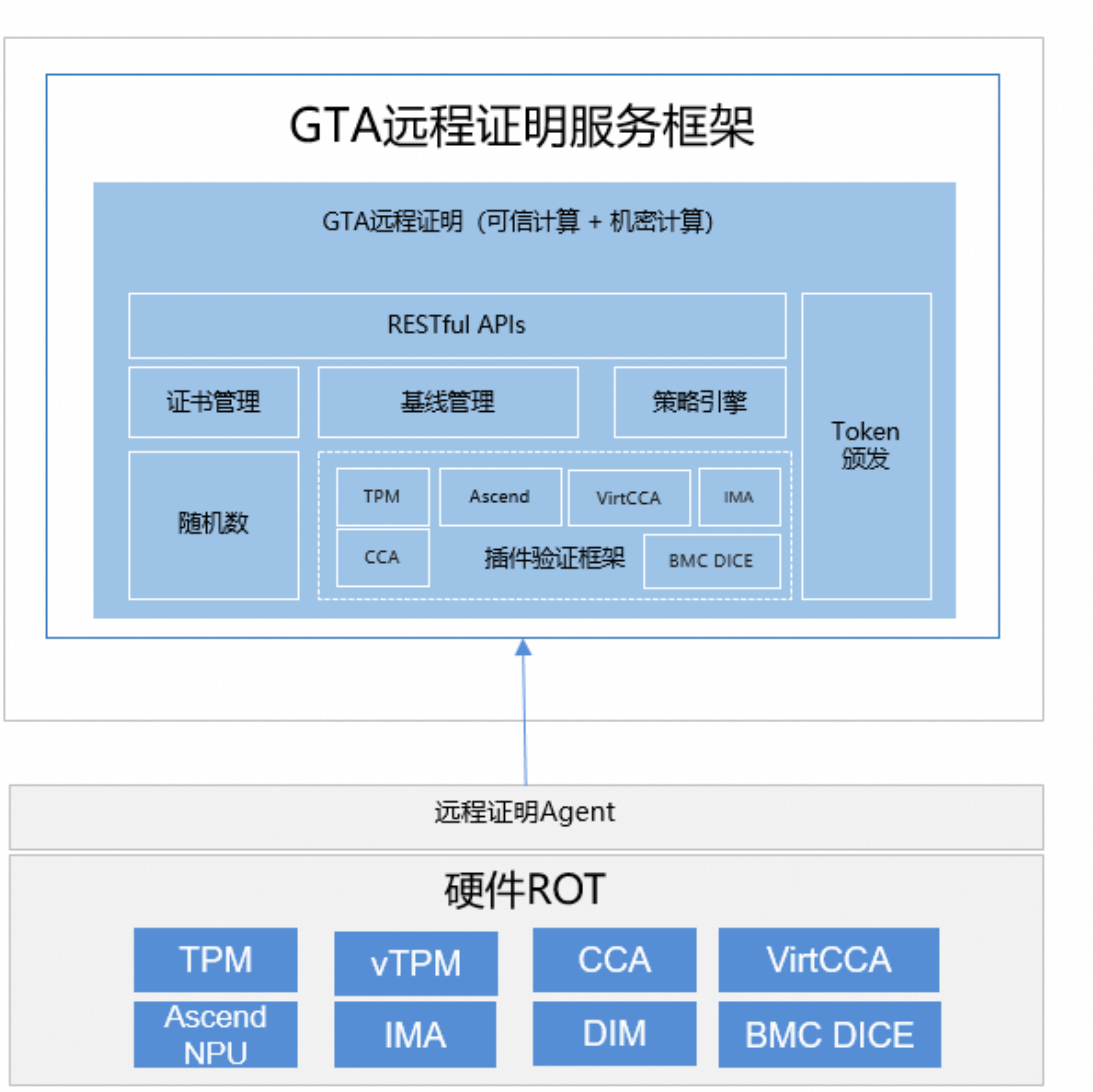
GTA 远程证明服务组件支持 TPM/vTPM、BMC DICE，Ascend NPU，iTrustee, VirtCCA、CCA 及 IMA/DIM 的远程证明，分为客户端和服务端。

- 服务端提供了远程证明服务框架兼容可信计算及机密计算，支持证书、策略等的增删改查，Quote 验证，随机数，JWT Token 生成等能力。
- 客户端支持采集本地 TPM 证据，并可与服务端交互，验证 Quote。

本组件在安全性及易用性上也提供了多种能力。

安全性上支持数据库完整性保护、数据链路加密，验证防重放，SQL 防注入，用户隔离，密钥轮换机制等一系列差异化安全竞争力。

易用性上支持护照模式及背调模式。客户端支持定时上报，响应挑战等多种验证模式。客户端及服务端支持 rpm 包及 docker 安装部署。



应用场景

远程证明是开启机密计算及可信计算的前置条件。只有当运行环境在密码学意义上被严格证明安全可信后方可进行后续运算。所有涉及机密计算的端到端解决方案都应将远程证明作为整体解决方案中的核心一环，一旦运行环境被证明不可信，则应立即中止后续任务。在 AI 模型保护，用户隐私保护，密钥管理等多场景均有广泛应用。

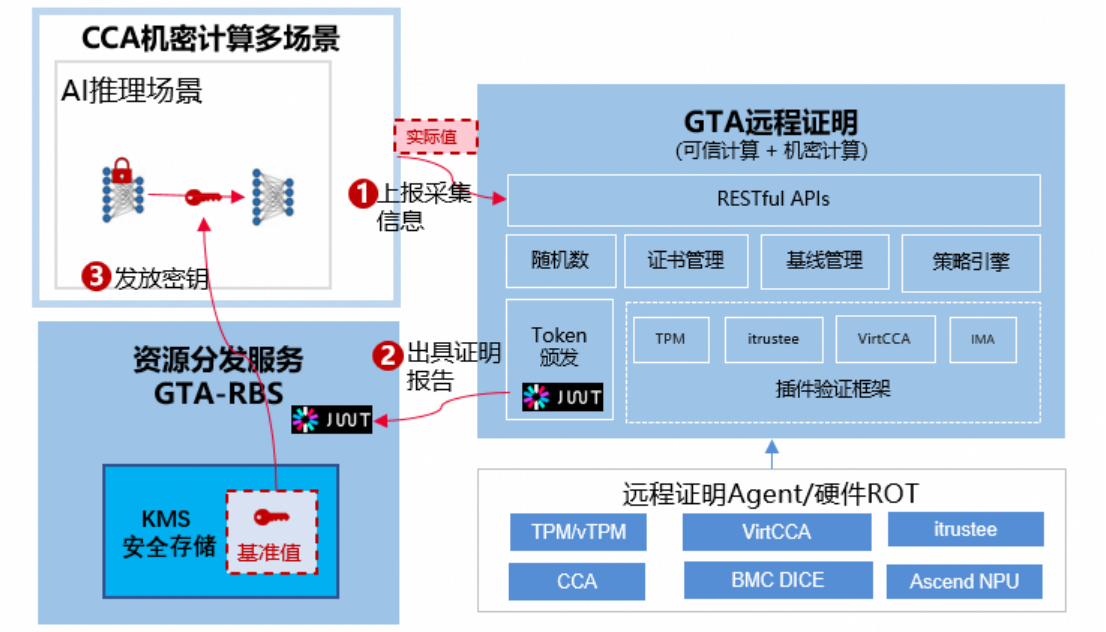
Global Trust Authority - Resource Broker Service

功能描述

Global Trust Authority RBS 资源分发服务是与 GTA 远程证明接口强耦合的配套服务。RBS 提供资源（如密钥）导入导出及其安全存储的能力，并可对该资源绑定安全环境基准值。如将 id1 的资源映射为机密环境 CCA 基准值 RIM 为 0x1234...。实际使用时，机密环境获取资源时，RBS 会验证 GTA 远程证明的 JWT 报告，当且仅当机密环境实际值与预设的基准值完全匹配时，才分发资源到机密环境中。 GTA 远程证明 + 机密计算/可信计算 TEE + RBS 资源分发服务，符合 rfc9334 对 Verifier，Attester，Relying Party 的定义，既支持背调模式，也支持护照模式。三者联用，可支持绝大部分机密计算的使用场景，也符合业界对机密计算的通用用法。

应用场景

以 AI 推理场景对私有模型的保护为例，模型 Owner 生成密钥本地加密模型后部署于机密环境如 CCA 机密虚机，并将密钥导入 RBS 安全存储，密钥与安全环境基准值绑定，如可定义 CCA REM0~3, RIM 等寄存器基准值 。运行时，通过 GTA 远程证明实时验证机密环境是否符合预定义的 REM 等寄存器基准值。RBS 验证通过后，将密钥以安全方式发送至机密虚机中，用于后续解密模型。事实上，资源分发服务在机密计算中与远程证明联用，可解决机密虚机内所有密钥来源，该密钥最常应用于落盘加解密,也可用于传输等其他加解密场景。



secScanner：操作系统安全加固组件

secScanner 是一款 Linux 操作系统安全扫描工具，旨在为操作系统提供安全管理、漏洞扫描、rootkit 入侵检测等功能。

功能描述

secScanner 基于“3 核心能力+3 公共能力”的技术架构，建设全面有效、实时自动、灵活易用的主动防护能力。

- **安全管理**：操作系统已有安全能力的使能度管理，针对操作系统安全配置复杂、运维人员安全意识不足、不同场景下安全配置需求不同等问题，安全管理功能可支持多基线选择，亦可根据需求定制安全基线，通过配置文件中预设的参数细粒度控制安全配置项，一键进行安全检测、安全加固、配置还原的功能。
- **漏洞扫描**：全面和及时应对操作系统组件已知 CVE 威胁，针对操作系统组件存在已知的 CVE 漏洞，潜在攻击者利用已知的攻击路径发起攻击。漏洞扫描功能支持从 openEuler 安全中心下载安全公告、CVE 公告保存在本地漏洞数据库，对系统组件进行全量扫描或者重点组件轻量化定向扫描，报告系统当前存在风险的组件，并给出升级建议。

- **入侵检测**：泛化性应对操作系统未知威胁，针对攻击者通过各种未知手段对系统进行入侵，入侵检测功能调用了 secDetector 工具对系统潜在的恶意 rootkit 模块进行检测，并给出清理建议。

应用场景

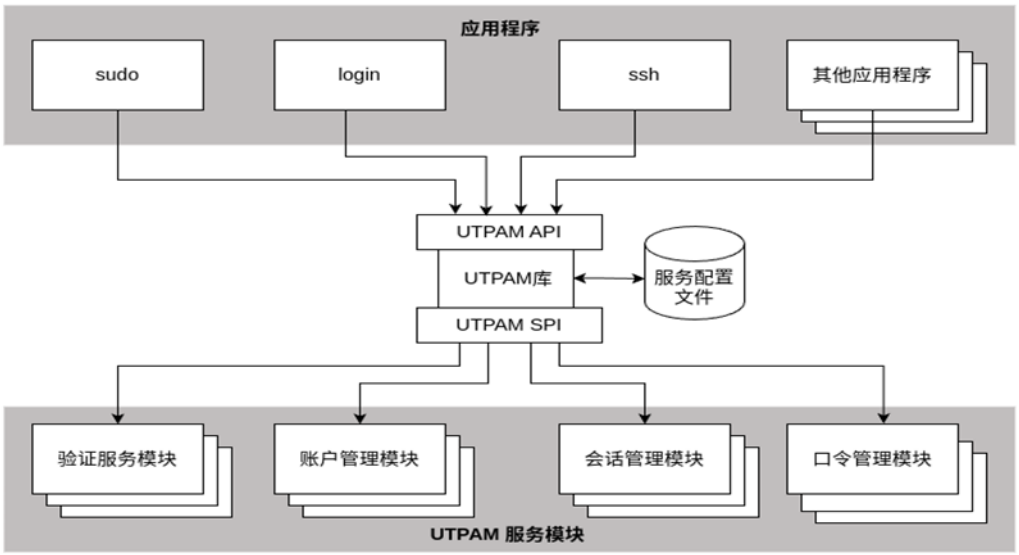
secScanner 安全加固工具主要应用于服务器安全运维场景。能够提供自动化的安全运维，一键检测当前系统中的薄弱配置项，并提供按照预置基线进行配置项加固的能力。有效防止运维人员安全意识不足产生的安全风险，减少运维人力，保护系统安全。

utpam：基于 Rust 开发的身份认证模块

utpam 是一种用于 Linux 系统的认证框架。它允许系统管理员定义系统中的不同服务的认证机制，并且可以根据需要组合多种认证方式。utpam 通过提供一个统一的接口来处理认证相关的工作，简化了应用程序对用户身份验证的过程。

功能描述

下图为 utpam 工具的整体系统结构，libutpam 是 utpam 的核心库，它根据应用程序提供的配置文件进行认证的初始化，配置文件默认位于/etc/utpam 目录下，文件以服务名称命名。utpam 向上为应用程序提供 API 接口选择认证类型，认证类型共有四类，分别为用户验证、账户管理、会话管理和口令管理；utpam 向下为开发者提供与 API 对应的 SPI 接口来实现具体的认证过程。



应用场景

UTPAM 是针对 Linux PAM 的 Rust 扩展套件，提供可插拔的认证模块与工具库，可在登录、鉴权与会话管理等阶段按需加载。典型场景包括：

安全加固：在特定阶段默认拒绝或严格校验，阻断不合规的认证路径。

特权管控：对 root/高权限账户执行快速放行或额外检查，降低运维摩擦。

审计与提示：在认证流程输出上下文信息（用户、TTY、服务、来源），提升可观测性。

异构集成：通过稳定的 C 接口与头文件接入传统 PAM/C 系统，同时复用 Rust 公共库。

适用人群与需求：需要分阶段控制认证行为、增强审计提示、对特权账户施策，以及在现有 PAM/C 环境引入更安全可靠的 Rust 实现的团队与系统。

cu-concrete 安全加固工具

功能描述

cu-concrete 安全加固工具平台是面向企业级打造的安全加固框架，主要用于自动化检测和修复系统层面的安全隐患，重点解决信创替代场景下批量加固效率低、安全基线适配难、传统加固脚本耦合度高、参数硬编码、运维成本高等问题，平台采用分层模块化设计，整合 CLI 命令行、TUI 可视化和 WEB 网页三种交互能力，内置 55 项标准化加固项，支持参数化配置，无需修改代码即可适配不同安全基线，具备开放式框架能力，可按照规范自主开发并自动集成新增加固模块，实现安全能力持续演进，同时支持按云池、按列表批量下发加固任务，能够实时监控任务执行状态，支持自动化配置生成和配置脚本微调，可留存、查看历史执行日志，实现从任务下发、执行监控到结果查看的全流程可追溯管理，平台整体采用配置化、开放式设计理念，所有加固策略通过 YAML 文件实现参数化管理，依托统一开发与目录规范保障模块扩展能力，并通过文件化轻量化存储实现加固状态持久化。

应用场景

cu-concrete 安全加固工具平台主要应用于信创服务器国产化改造相关项目，适用于存在大规模异构服务器集群的企业运维场景，可针对多类型服务器环境开展标准化、批量化的安全加固与合规整改工作，能够适配不同安全基线的合规适配需求，有效解决传统加固方式整改周期长、人工运维成本高、基线适配困难的问题，可支撑企业根据自身安全需求持续拓

展加固能力，为大规模异构服务器集群提供稳定、合规、可审计的安全防护体系，满足常态化安全加固、批量运维整改、合规审计追溯的企业级使用需求。

cu-scanner 漏洞扫描软件

功能描述

cu-scanner 是一款专为网络安全领域打造的工具，其核心功能是依据操作 CVE（Common Vulnerabilities and Exposures，通用漏洞披露）与安全公告的文件或数据接口，对相关信息进行收集、整理和唯一化处理，进而快速生成 OVAL（Open Vulnerability and Assessment Language，开放漏洞评估语言）格式的 XML 文件以及 SCAF（Security Content Automation Framework，安全内容自动化框架）格式的 JSON 文件。

该工具能够帮助安全人员更高效地处理漏洞信息，为漏洞评估、安全扫描等工作提供标准化的数据支持，有助于提升网络安全防护的准确性和及时性。无论是面对大量的本地文件还是实时的数据接口，cu-scanner 都能稳定、快速地完成数据处理与格式转换，满足不同场景下的安全工作需求。

应用场景

漏洞评估、披露: 安全工具可以使用 OVAL 定义来自动化评估系统是否存在已知漏洞，从而帮助组织识别潜在的安全风险。

合规性检查: OVAL 可以用于检查系统是否符合特定的安全标准和政策，例如 PCI-DSS、HIPPA 等合规性要求。

配置管理: 通过使用 OVAL 定义，组织可以评估系统配置是否符合最佳实践，从而减少安全漏洞的风险。

安全审计: 安全审计人员可以利用 OVAL 来系统性的检查和记录系统的安全状态，确保所有安全措施得到有效实施。

补丁管理: OVAL 可以帮助识别需要应用补丁的系统，确保系统保持最新和安全。

自动化响应: 在安全事件响应过程中，OVAL 可以用于快速评估受影响的系统状态，帮助安全团队采取适当的应对措施。

安全工具集成: 许多安全工具(如漏洞扫描器、合规性检查工具等)支持 OVAL 标准，能够通过 OVAL 定义进行自动化的安全评估和报告。

safeguard 审计观测工具

功能描述

safeguard 是基于 KRSI/eBPF+LSM 的 Linux 安全审计与访问控制工具，面向主机和容器环境提供文件、网络、挂载、进程等关键行为的监控、审计与阻断能力。该工具支持按进程名、父进程名、UID/GID、容器范围等上下文配置安全策略，可在监控模式下记录风险行为，也可在阻断模式下执行访问控制；同时支持白名单策略生成，帮助用户构建细粒度的最小权限运行环境。

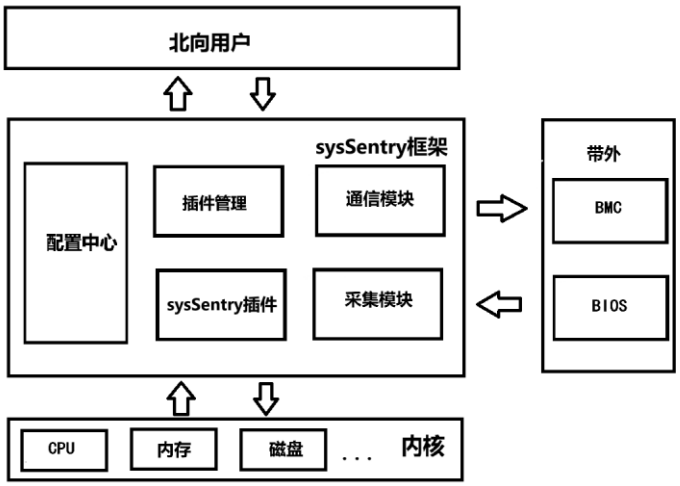
应用场景

适用于服务器、云原生节点及容器运行环境的安全加固与运行时防护。典型场景包括敏感文件访问保护、异常网络访问控制、容器挂载行为限制、进程执行白名单控制、业务运行行为审计、供应链构建环境加固，以及对 SSRF、容器逃逸、非授权访问等风险行为的检测和拦截，提升系统侧的安全可观测性和主动防护能力。

可靠性

sysSentry 系统级故障管理框架

sysSentry 主要提供故障巡检框架，该框架通过提供统一的北向故障上报接口以及南向提供支持不同巡检/诊断能力的插件，支持对系统中 CPU、内存、磁盘、NPU 等硬件故障进行巡检和诊断。



sysSentry 功能设计如下：

统一告警/事件通知服务：通过提供一个统一的告警服务，接收各个插件上报的故障信息，并由该通知服务进行统一转发，各个业务订阅服务可以根据需要进行不同故障的消息订阅。

统一日志服务：通过提供统一的日志服务，支持各个插件的故障信息进行汇总记录，提升问题定位效率。

故障诊断/巡检框架：该框架支持以插件化的方式进行各项巡检任务以及诊断任务的开发和配置，不同插件支持独立启动、停止、状态查询、结果查询以及启动方式设置，并且支持 C/C++、Python、Shell 等不同编程语言的插件。

轻量级数据采集服务：该服务支持通过内核接口、BIOS、BMC 等接口，查询硬件的各个状态信息，供各个插件进行分析和使用，并且支持适配底层不同的架构、版本以及数据采集服务。

慢 IO/慢盘检测插件

功能描述

慢 IO 检测基于滑动窗口对系统中一段周期内不同硬盘的 io 时延数据进行分析，当整个窗口中异常周期的数量超过一定数量时，则认为此时该盘发生慢 io 事件。

目前支持两种类型的慢 io 检测插件：基于平均时延计算异常阈值的平均阈值插件 avg_block_io，和基于 AI 算法聚类计算阈值的 AI 阈值插件 ai_block_io；两种插件均支持最多 10 个 io 阶段：blk-throttle、wbt、iocost、get_tag、plug、deadline、bfq、kyber、hctx、driver。

目前支持检测的硬盘类型有 NVMe SSD，SATA SSD，SATA HDD。

应用场景

1. 硬件故障：由于硬盘损坏、老化原因等导致硬盘出现慢 IO 情况。
2. IO 栈异常：由于内核 IO 栈配置、处理异常等导致出现慢 IO 情况。
3. 业务高负载：由于业务负载导致出现 IO 资源使用超限的情况。

磁盘寿命检测和健康监控插件

功能描述

磁盘寿命检测和监控插件，使用 ipmitool 工具，通过轮巡方式获取 bmc 上现有的所有磁盘告警，按照配置文件参数筛选需要的告警信息上报。在直通场景下，通过 ipmi 接口查询获取问题磁盘的盘符以及对应物理盘 SN 号，在 raid 场景下，通过相关的 raid 工具来获取磁盘盘符和物理盘 SN 号的对应关系，帮助快速预警硬盘问题。

应用场景

仅支持鲲鹏平台，华为自研 nvme 盘，硬盘故障，温度过高，IO 性能下降，寿命过低等问题，及时预警，保证系统可靠可用。

超算内存支持 Zone 粒度内存回收

功能描述

开启 Zone Reclaim 后，在直接回收前，新增了唤醒kswapd逻辑。在空闲内存不足时，同步唤醒kswapd，进行后台的内存异步回收，将空闲内存回收到 high 水线，避免频繁在缺页处理中进行内存回收，影响内存分配效率，提升了Zone Reclaim 特性的效率。

应用场景

在一般的2P服务器架构中，每个计算节点包含多个DDR内存节点，单个内存节点配置为一定数量的内容。这种内存节点多但单节点内存容量有限的设计，在内存密集型应用场景下容易因单节点内存不足而触发OOM-killer机制，导致用户态进程被强制终止。

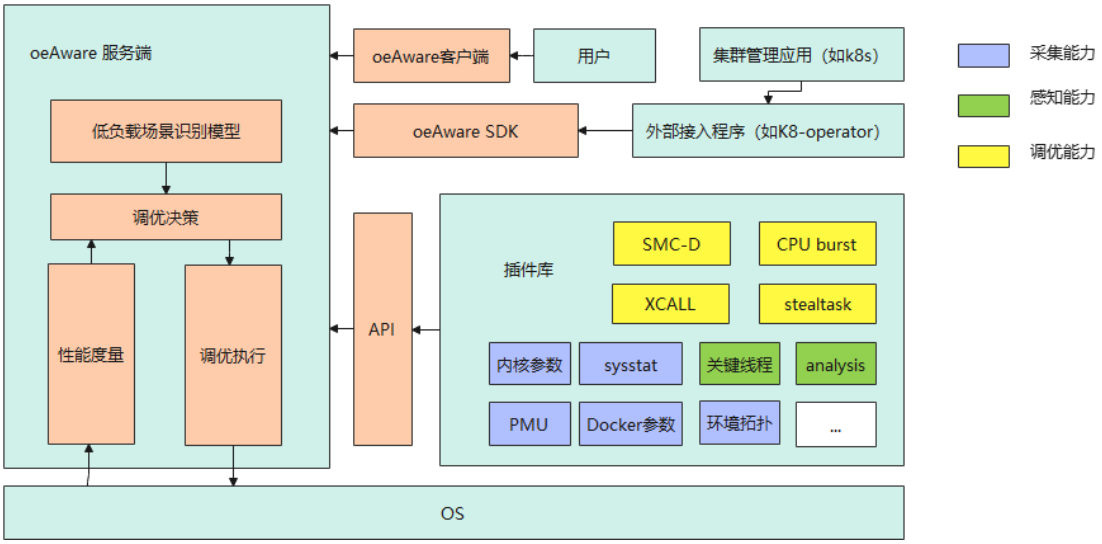
为缓解这一问题，通常建议启用Zone Reclaim机制。该机制能够在单节点内存不足时，优先回收本节点的文件页缓存（File Page Cache），而非跨节点占用难以回收的匿名页（Anonymous Page）。这种本地化回收策略可有效缓解单节点内存不足的情况，提升内存的局部性。

开启本特性后，通过异步的内存回收，避免的同步的内存回收开销，特性了内存申请的效率，从而提升性能。

性能调优

oeAware 采集、调优插件等功能增强

oeAware 是在 openEuler 上实现低负载采集感知调优的框架，目标是动态感知系统行为后智能使能系统的调优特性。传统调优特性都以独立运行且静态打开关闭为主，oeAware 将调优拆分为采集、感知和调优三层，每层通过订阅方式关联，各层采用插件式开发尽可能复用。



功能描述

oeAware 的每个插件都是按 oeAware 标准接口开发的动态库，包含若干个实例，每个实例可以是一个独立的采集、感知或调优功能集，每个实例包含若干个 topic，其中 topic 主

要用于提供采集或者感知的数据结果，这些数据结果可供其他插件或者外部应用进行调优或分析。

- SDK 提供的接口可以实现订阅插件的 topic，回调函数接收 oeAware 的数据，外部应用可以通过 SDK 开发定制化功能，例如完成集群各节点信息采集，分析本节点业务特征。
- PMU 信息采集插件：采集系统 PMU 性能记录。
- Docker 信息采集插件：采集当前环境 Docker 的一些参数信息。
- 系统信息采集插件：采集当前环境的内核参数、线程信息和一些资源信息（CPU、内存、IO、网络）等。
- 线程感知插件：感知关键线程信息。
- 评估插件：分析业务运行时系统的 NUMA 和网络信息，给用户推荐使用的调优方式。
- 系统调优插件：（1）stealtask ：优化 CPU 调优 （2）smc_tune（SMC-D）：基于内核共享内存通信特性，提高网络吞吐，降低时延 （3）xcall_tune ：跳过非关键流程的代码路径，优化 SYSCALL 的处理底噪。(4) soft_domain_tune：任务分域调度，多实例业务单个实例尽量在独立的调度域内调度，更加亲和。
- Docker 调优插件：利用 cpuburst 特性在突发负载下环境 CPU 性能瓶颈。

约束限制

- SMC-D：需要在服务端客户端建链前，完成使能 smc 加速。比较适用于长链接多的场景。
- Docker 调优：暂不适用于 K8s 容器场景。
- xcall_tune：内核配置选项 FAST_SYSCALL 打开。
- realtime_tune ：需要配合 Preempt-RT 内核使用。
- net_hard_irq_tune ：TCP 网络通信业务。
- soft_domain_tune ：内核配置选项 SOFT_DOMAIN 打开，容器大小不能超过一个 numa

应用场景

stealtask 适用于希望提高 CPU 利用率的场景，例如 Doris，该调优实例可以有效提高 CPU 利用，避免 CPU 空转。

XCALL 适用于应用程序的 SYSCALL 开销较大的场景，XCALL 提供一套跳过非关键流程的代码路径，来优化 SYSCALL 的处理底噪，这些被跳过的非关键流程会牺牲部分维测和安全功能。

SMC-D 特别适用于需要高吞吐量和低延迟的应用场景，如高性能计算（HPC）、大数据处理和云计算平台。通过直接内存访问（DMA），SMC-D 能够显著减少 CPU 负载并提升交互式工作负载的速率。

soft_domain_tune 适用于多实例业务，实例大小不超过一个 numa。实例间运行较独立，通过对调度范围进行分域，可以减少业务间的干扰。

cpuburst 适用于高负载容器场景，例如 Doris，能够缓解 CPU 限制带来的性能瓶颈。

realtime 适用于需要低时延、低抖动、有严格时延限制的场景，如工业制造行业。

net_hard_irq_tune 适用于业务性能受 TCP 网络影响较大的业务，例如 redis、nginx 等。

并行感知调度适用于跨 NUMA 访存对业务性能比较大且不好固定绑核的场景，例如 Spark。

容器潮汐调度适用于容器化部署的业务，例如容器化 mysql，当业务负载低时，在满足 QoS 的基础上，使用最少的 CPU 资源，减少 CPU 迁移、idle 切换、cache miss 和跨 NUMA 访问，增强资源的局部性。当业务负载高时，又可以突破绑核约束，使用更多 CPU 资源来提升 QoS 质量，提高 CPU 资源利用率。

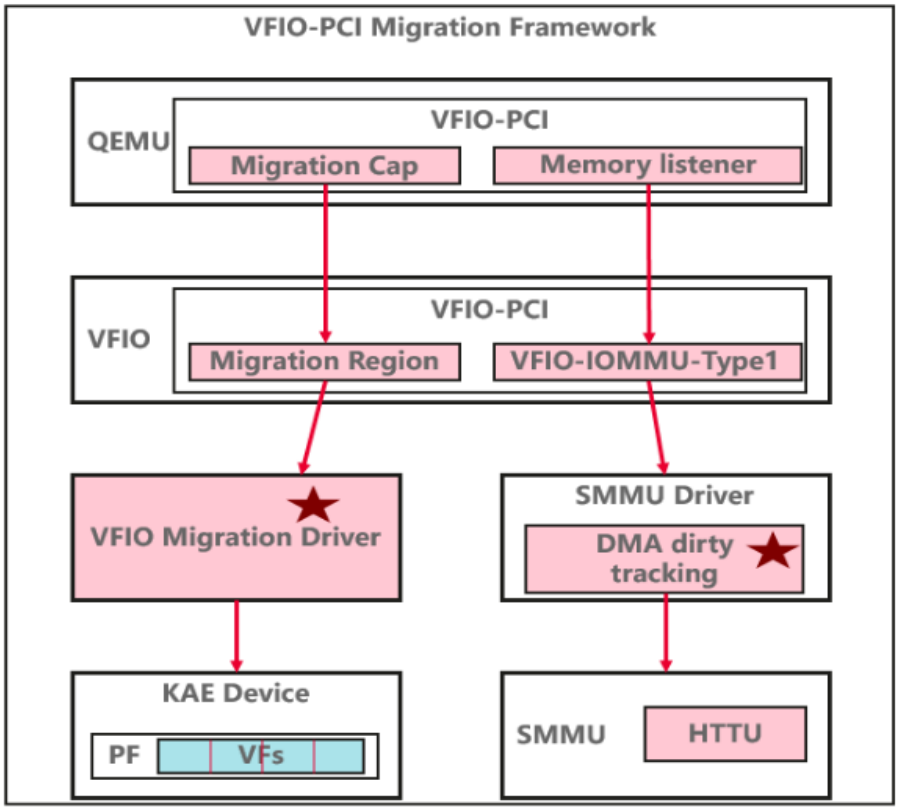
虚拟化

虚拟化支持 vKAE 直通设备热迁移

功能描述

KAE 是基于鲲鹏 920 新型号处理器提供的硬件加速解决方案，包括 HPRE、SEC、ZIP 设备，可用于加解密和压缩解压缩，能够显著降低处理器消耗，提高处理器效率。KAE 直通热迁移是指虚拟机在配置 KAE 直通设备时，进行热迁移的能力，可以为 KAE 设备的使用提供更强的灵活性和业务不中断的保障。

smmu 脏页跟踪是实现直通设备高效、可靠的热迁移的关键技术。在 ARM 架构中，通过纯软件方式进行脏页跟踪，会带来了较大的性能损耗。HTTU(Hardware Translation Table Udate)允许硬件自动更新 smmu 页表状态，在进行写操作时会自动置对应页表项的写权限位，热迁移时扫描页表的写权限位进行脏页统计。



应用场景

为虚拟机中使用 KAE 直通设备的场景提供热迁移支持，适用于对数据安全性和处理性能有较高要求的领域，如金融、云计算和大数据处理等，有效增强业务连续性与运行稳定性。

VMAnalyzer：轻量级虚拟化性能监控组件

VMAnalyzer 是一款轻量级的虚拟化监测分析工具，围绕如下两个方面进行设计：

虚拟化环境监测：通过实时监测虚拟机的 CPU、内存、磁盘 I/O 等关键性能指标，发现性能瓶颈。

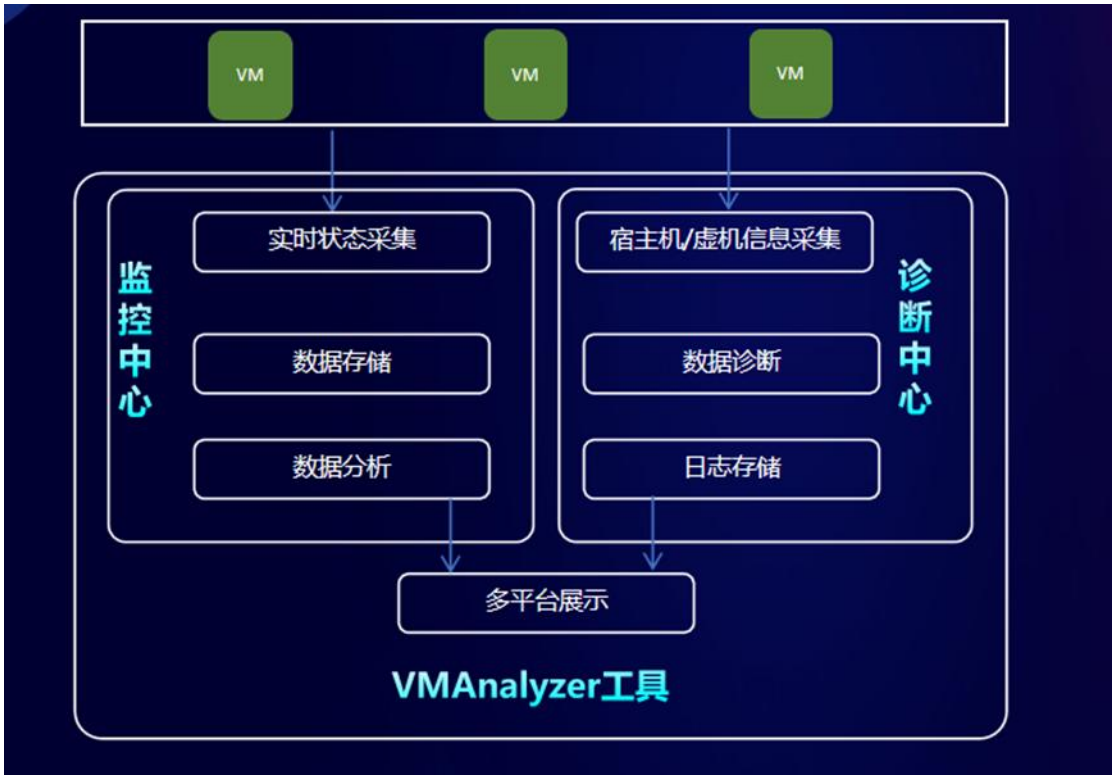
高可靠性保障：通过分析虚拟机的 qemu 进程、物理环境配置等信息提供高可靠性的虚拟机维护方案，预测并发现潜在的故障风险。

功能描述

VMAnalyzer 包括监控中心、诊断中心两部分，架构如下：

监控中心：实时采集虚机的数据，能够细粒度的分析虚拟机的运行状况，检测结果的可灵活通过 console、OPS、DW 等多平台展示各个云主机数据。

诊断中心：针对虚拟化层面的问题，通过执行命令对虚拟化层进行深度诊断，通过配置分析、状态分析，虚机进程消耗分析等，帮助用户分析虚拟化层面的各类问题，最终通过日志文件展示出来。



应用场景

VMAnalyzer 是一款轻量级的虚拟化性能监测分析工具，主要应用于云计算环境，能够细粒度的分析虚拟机的运行状况和性能，可轻松识别性能问题和瓶颈，可以帮助用户维护具有高性能和高可靠性的虚拟机。

GIC 超分优化

功能描述

GICv4.1 特性，由于硬件多条 VMOVP 指令需要串行同步执行限制，在超分&范围绑核等 vcpu 频繁迁移场景会出现性能劣化；

然而，客户场景普遍存在超分场景使得算力最大化，GICv4.1 作为一个虚拟化 IO 常用优化手段在超分场景下 1:2 场景下有性能劣化，导致客户在超分场景下不得不关闭 GICv4.1，无法解决超分这一类 IO 场景虚拟化损耗的关键痛点。为了提升该场景性能，设计通过优化 VMOVP 指令数减少该性能瓶颈，进一步提升整体性能。

使用方式：

新增/sys/module/kvm/parameters/enable_vmovp_elision 参数控制该功能，如需使用，需设置该参数为 Y。

应用场景

在鲲鹏 920 新型号机型上，可以优化 GICv4.1 开启场景，虚拟机同亲和组内频繁核间迁移场景性能。

vTimer 直通

功能描述

传统虚拟机 timer 中断需在中断到期时陷出，Hypervisor 侧注入中断后再陷入 Guest，针对 timer 定时时延敏感类业务可能存在较大影响。vTimer 直通通过避免 timer 中断注入过程中的陷入/陷出操作，从而降低中断注入时延，提升场景性能。

应用场景

在鲲鹏 920 新型号及之后机型上，GICv4.1 开启场景，对 GuestOS 的 timer 时延有极致要求，可通过此技术极大的提升 timer 时延精度

硬件

支持树莓派

Raspberry Pi（树莓派）是由 Raspberry Pi 基金会与 Broadcom 公司合作开发的一系列小型单板计算机。凭借其价格低、体积小、能耗低、高可编程性以及丰富的生态系统等特点，树莓派在工业自动化、机器人技术、物联网、教育以及业余爱好者项目等领域得到了广泛应用。树莓派 4B 和树莓派 5 作为树莓派产品线的经典代表，采用 ARM 架构的处理器。其中树莓派 4B 是极具性价比的普及型单板计算机，树莓派 5 凭借其显著的性能突破和扩展能力成为一款在高性能边缘计算领域颇具竞争力的创新产品。

功能描述

作为开源硬件领域的一个较为高阶的硬件产品，树莓派 4B 和树莓派 5 支持 Raspberry Pi OS、Ubuntu、openEuler 等多种 Linux 发行版，外设丰富，具有较强的视频编解码能力，以及板载网络等功能，完全可以作为独立计算机系统使用。

应用场景

树莓派 4B 和树莓派 5 凭借其强大的性能和丰富的扩展能力，广泛应用于多个领域：

1. 教育与学习：学习 Python 等编程语言、借助外设接口进行电子实验；
2. 多媒体与娱乐：作为媒体中心或游戏机；
3. 物联网和智能家居：作为传感器节点或智能家居中枢，用于环境监测、家庭自动化控制和边缘计算；
4. 服务器与网络应用：用于家庭服务器、轻量级 Web 服务及容器化应用；
5. 创客与 DIY 项目：用于机器人控制、3D 打印管理和无人机飞控；
6. 科研与开发：用于 AI 实验、嵌入式开发原型验证；
7. 工业与自动化：实现设备监控、人机界面和机器视觉。

海光架构特性增强

HYGON CCP 驱动支持 SM4-XTS/GCM

功能描述

本特性面向 Hygon CCP（Cryptographic Co-Processor）密码协处理器，升级其内核驱动以支持 SM4 算法的高级模式 SM4-XTS 和 SM4-GCM。SM4-XTS 为可调整密码本模式，适用于块设备加密、磁盘加密等场景，要求 32 字节密钥与 16 字节 IV；SM4-GCM 为带认证的计数器模式，支持 AAD（附加认证数据）与认证 Tag，可同时保障数据机密性与完整性。新增算法以 xts-sm4-ccp、gcm-sm4-ccp 形式注册到 Linux Kernel Crypto API，并保留 xts-sm4-cis 路径以支持 CIS 扩展。测试方面，通过内核模块 ccp_test.ko 直接调用 crypto_skcipher / crypto_aead API，同时基于 libkcapi 提供用户态 AF_ALG 接口测试，测试向量分别引用 GB/T 17964-2021（SM4-XTS）与 RFC 8998（SM4-GCM）。

应用场景

块设备与磁盘加密：为服务器本地盘、云盘提供 SM4-XTS 硬件加速加密，满足数据静态安全需求。

数据库与金融核心业务：利用 SM4-GCM 的 AEAD 能力，为交易数据、日志、消

息队列提供机密性与完整性双重保护。

容器与云原生环境: 为 Kubernetes 持久化卷、容器镜像层加密提供国密算法支持。

合规与信创替代: 满足政府、金融、电信等行业对国产密码算法 SM4 的合规要求，支撑信创服务器国产化改造。

分布式存储

fastblock 分布式块存储

功能描述

fastblock 是一个专为高性能、极低延迟设计的开源分布式块存储系统。它旨在解决传统分布式存储（如 Ceph）在 NVMe SSD 硬件环境下 CPU 消耗大、单卷延迟高、可用性差等瓶颈。

- 极速 IO 路径：核心基于 SPDK（存储性能开发套件）编写，采用用户态 NVMe 驱动与无锁队列技术，规避了内核态切换，实现极致的单卷读写性能与超低延迟。
- 零拷贝网络：利用 RDMA 技术进行数据传输，支持内核旁路与零拷贝，网络通信无需 CPU 深度干预。
- 强一致性复制：采用 Multi-Raft 协议进行数据复制，既保证了数据的强一致性与高可靠性，又避免了传统主从同步复制在集群抖动时导致的 IO 阻塞。
- 高效元数据管理：利用 Go 语言与 etcd 实现轻量、易定制的 Monitor 集群，确保元数据视图在客户端和存储节点（OSD）之间高度一致。

应用场景

fastblock 适用于对存储吞吐量、IOPS 和响应时间有严苛要求的现代化数据中心与云基础设施:

- 高性能虚拟化与云平台：通过 SPDK vhost-blk 接口无缝对接 QEMU/KVM 虚拟机，为公有云或私有云平台提供亚毫秒级低延迟、高并发的系统盘与数据盘服务。
- 容器化云原生数据库：通过 CSI（容器存储接口）与 Kubernetes 对接，为 MySQL、PostgreSQL、Redis 等对 IO 极其敏感的数据库和状态集（StatefulSet）提供高性能、高可用的持久化卷支持。
- 高性能计算 (HPC) 与裸金属服务：使用 NBD（网络块设备）形式将远程块设备直接

映射给裸金属服务器，支撑 AI 模型训练、大数据分析等需要高吞吐吞吐量的存算分离业务。

- 高可用金融级核心业务：依托 Multi-Raft 的强一致性优势与 RDMA 低延迟网络，满足金融交易、在线支付等核心系统对数据零丢失和超快响应时间的严苛需求。

7. 著作权说明

8. 商标

9. 附录

附录 1：搭建开发环境

环境准备	地址
下载安装 openEuler	https://openeuler.org/zh/download/
开发环境准备	https://atomgit.com/openeuler/community/blob/master/zh/contributors/prepare-environment.md

构建软件包	https://atomgit.com/openeuler/community/blob/master/zh/contributors/package-install.md

附录 2：安全处理流程和安全批露信息

社区安全问题披露	地址
安全处理流程	https://atomgit.com/openeuler/security-committee/blob/master/docs/zh/vulnerability-management-process/security-process.md
安全披露信息	https://atomgit.com/openeuler/security-committee/blob/master/docs/zh/vulnerability-management-process/security-disclosure.md
安全保障策略总纲	https://atomgit.com/openeuler/security-committee/blob/master/security-strategy-overview.md

附录 3：64K 内核 openEuler 自孵化特性兼容性规格限制

组件	需求描述	64k 兼容情况
kernel	ext4: use iomap for regular file's buffered IO path and enable large folio	不兼容，通过 mount 参数限制，默认不使用
	添加 clear_free_list_pages sysctl 特性	不兼容，config 默认关闭
	打印脏页特性	不兼容，config 默认关闭
	内存可靠性分级	不兼容，config 默认关闭
	Per-memcg swap control	不兼容，config 默认关闭
	提供支持大页内存的共享内存映射工具	不兼容，config 默认关闭
虚拟化	虚拟化场景支持虚拟机内存分级压缩能力	不兼容，64k 无法使用
	DPU 支持内核态 vDPA 系统盘及支持数据盘热扩容	不兼容，默认防呆
	openEuler 24.03 支持 vdpa 需求	不兼容，默认防呆
secDetector	为 OS 设计的内构入侵检测系统，旨在为关键	不兼容，默认安装防呆

	信息基础设施提供入侵检测及响应能力	
dim	动态完整性度量特性能够检测到运行态的篡改和注入等攻击引起的内存代码段变化	不兼容，默认安装防呆
sysmonitor	控 OS 系统运行过程中的异常，将监控到的异常记录到系统日志	不兼容，默认安装防呆
sysmaster	全新 1 号进程实现方案，秒级故障自愈，保障系统全天在线	不兼容，默认安装防呆