openEuler 24.03 LTS SP3 Technical White Paper

1 Introduction

The openEuler open source community is incubated and operated by the OpenAtom Foundation.

openEuler is a digital infrastructure OS that fits into any server, cloud computing, edge computing, and embedded deployment. This secure, stable, and easy-to-use open source OS is compatible with multiple computing architectures. openEuler suits operational technology (OT) applications and enables the convergence of OT and information and communications technology (ICT).

The openEuler open source community is a portal available to global developers, with the goal of building an open, diversified, and architecture-inclusive software ecosystem for all digital infrastructure scenarios. It has a rich history of helping enterprises develop their software, hardware, and applications.

The openEuler open source community was officially established on December 31, 2019, with the original focus of innovating diversified computing architectures.

On March 30, 2020, the Long Term Support (LTS) version openEuler 20.03 was officially released, which was a new Linux distribution with independent technology evolution.

On September 30, 2020, the innovative openEuler 20.09 version was released through the collaboration efforts of multiple companies, teams, and independent developers in the openEuler community. The release of openEuler 20.09 marked a milestone not only in the growth of the openEuler community, but also in the history of open sourced software in China.

On March 31, 2021, the innovative kernel version openEuler 21.03 was released. This version is enhanced in line with Linux kernel 5.10 and also incorporates multiple new features, such as live kernel upgrade and tiered memory expansion. These features improve multi-core performance and deliver the computing power of one thousand cores.

On September 30, 2021, openEuler 21.09 was released. This premium version is designed to supercharge all scenarios, including edge and embedded devices. It enhances server and cloud computing features, and incorporates key technologies including cloud-native CPU scheduling algorithms for hybrid service deployments and KubeOS for containers.

On March 30, 2022, openEuler 22.03 LTS was released based on Linux kernel 5.10. Designed to meet all server, cloud, edge computing, and embedded workloads, openEuler 22.03 LTS is an all-scenario digital infrastructure OS that unleashes premium computing power and resource utilization.

On September 30, 2022, openEuler 22.09 was released to further enhance all-scenario innovations.

On December 30, 2022, openEuler 22.03 LTS SP1 was released, which is designed for hitless porting with best-of-breed tools.

On March 30, 2023, openEuler 23.03 was released. Running on Linux kernel 6.1, it streamlines technical readiness for Linux kernel 6.x and facilitates innovations for hardware adaptation and other technologies.

On June 30, 2023, openEuler 22.03 LTS SP2 was released, which enhances scenario-specific features and increases system performance to a higher level.

On September 30, 2023, the openEuler 23.09 innovation version was released based on Linux kernel 6.4. It provides a series of brand-new features to further enhance developer and user experience.

On November 30, 2023, openEuler 20.03 LTS SP4 was released, an enhanced extension of openEuler 20.03 LTS. openEuler 20.03 LTS SP4 provides excellent support for server, cloud native, and edge computing scenarios.

On December 30, 2023, openEuler 22.03 LTS SP3 was released. Designed to improve developer efficiency, it features for server, cloud native, edge computing, and embedded scenarios.

On May 30, 2024, openEuler 24.03 LTS was released. This version is built on Linux kernel 6.6 and brings new features for server, cloud, edge computing, AI, and embedded deployments to deliver enhanced developer and user experience.

On June 30, 2024, openEuler 22.03 LTS SP4 was released. Designed to improve developer efficiency, it further extends features for server, cloud native, edge computing, and embedded scenarios.

On September 30, 2024, openEuler 24.09 was released. It is an innovation version built based on Linux kernel 6.6 and brings more advanced features and functions.

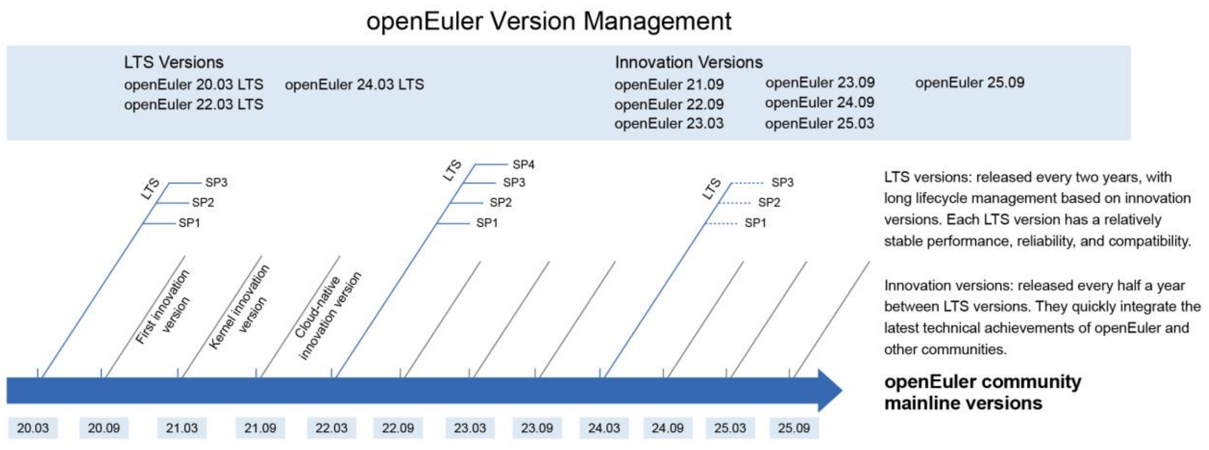
On December 30, 2024, openEuler 24.03 LTS SP1 was released. This enhanced and extended version of openEuler 24.03 LTS is developed on Linux kernel 6.6 and designed for server, cloud, edge computing, and embedded deployments. It offers new features and enhanced functions that streamline processes across a range of domains.

On March 30, 2025, openEuler 25.03 was released. It is an innovation version designed based on Linux kernel 6.6 and is suited for server, cloud, edge, and embedded scenarios. It provides a variety of new features and functions and brings brand-new experience to developers and users in diverse industries.

On June 30, 2025, openEuler 24.03 LTS SP2 was released. It is an enhanced version of 24.03 LTS based on the Linux 6.6 kernel. It is designed for server, cloud, AI, and embedded scenarios, introducing new functions and features covering kernel optimization, heterogeneous collaborative inference, high-density many-core computing, confidential containers, and multi-core and multi-instance hybrid deployment. It offers enhanced experience for developers and users across various industries.

On September 30, 2025, openEuler 25.09 was released. It is an innovation version based on the Linux 6.6 kernel. It offers new features and enhanced functionality for server, cloud, AI, and embedded scenarios. Key improvements include kernel optimization, heterogeneous collaborative inference, high-density many-core computing, confidential containers, confidential VMs, multi-core and multi-instance hybrid deployment, compilers, trusted computing, and cryptographic acceleration. This latest release enhances the experience for developers and users across a wide range of fields.

More recently, on December 30, 2025, the first openEuler release supporting SuperPoDs was officially launched. openEuler 24.03 LTS SP3, an enhanced and extended version of openEuler 24.03 LTS based on the Linux 6.6 kernel, is designed for server, cloud, and AI scenarios. It introduces a wide range of new features and capabilities, including kernel optimization, heterogeneous collaborative inference, intelligent diagnosis, confidential virtual machines (cVMs), compiler enhancements, RISC-V architecture optimizations, an intelligent developer desktop, security hardening, UnifiedBus-powered SuperPoDs, identity authentication, and virtualization. This release delivers a significantly improved experience for developers and users across a broader range of industries.

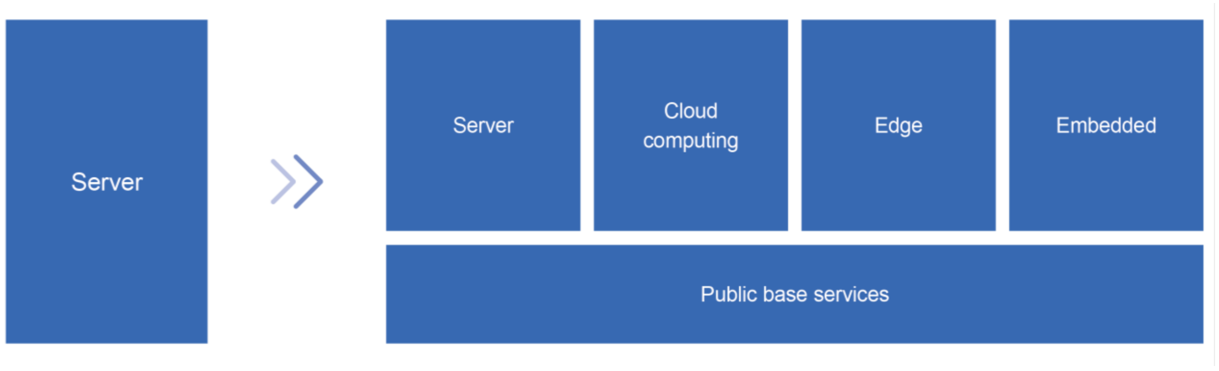

openEuler Version Management
LTS Versions
openEuler 20.03 LTS
openEuler 22.03 LTS
openEuler 24.03 LTS
Innovation Versions
openEuler 21.09
openEuler 22.09
openEuler 23.03
openEuler 23.09
openEuler 24.09
openEuler 25.03
openEuler 25.09
LTS
SP1
SP2
SP3
SP4
First innovation version
Kernel innovation version
Cloud-native innovation version
LTS versions: released every two years, with long lifecycle management based on innovation versions. Each LTS version has a relatively stable performance, reliability, and compatibility.
Innovation versions: released every half a year between LTS versions. They quickly integrate the latest technical achievements of openEuler and other communities.
openEuler community mainline versions
20.03
20.09
21.03
21.09
22.03
22.09
23.03
23.09
24.03
24.09
25.03
25.09

openEuler releases LTS versions to provide enhanced specifications and a secure, stable, and reliable OS for enterprise users.

openEuler is built on tried-and-tested technologies. Its innovation versions, released every half year, quickly integrate the latest technical achievements of openEuler and other communities. The innovative tech is first verified in the openEuler open source community as a single open source project, and then these features are added to each new release, enabling community developers to obtain the source code.

Technical capabilities are first tested in the open source community, and continuously incorporated into each openEuler release. In addition, each release is built on feedback given by community users to bridge the gap between innovation and the community, as well as improve existing technologies. openEuler is both a release platform and incubator of new technologies, working in a symbiotic relationship that drives the evolution of new versions.

Innovative Platform for All Scenarios



openEuler supports multiple processor architectures (x86, Arm, RISC-V, and LoongArch), as part of a focus to continuously improve the ecosystem of diversified computing power.

The openEuler community is home to an increasing number of special interest groups (SIGs), which are dedicated teams that help extend the OS features from server to cloud computing, edge computing, and embedded scenarios. openEuler is built to be used in any scenario.

The OS is a perfect choice for ecosystem partners, users, and developers who plan to enhance scenario-specific capabilities. By creating a unified OS that supports multiple devices, openEuler hopes to enable a single application development for all scenarios.

Open and Transparent Software Supply Chain

The process of building an open source OS relies on supply chain aggregation and optimization. To ensure reliable open source software or a large-scale commercial OS, openEuler comprises a complete lifecycle management that covers building, verification, and distribution. The brand regularly reviews its software dependencies based on user scenarios, organizes the upstream community addresses of all the software packages, and verifies its source code by comparing it to that of the upstream communities. The build, runtime dependencies, and upstream communities of the open source software form a closed loop, realizing a complete, transparent software supply chain management.

2 Platform Architecture

System Framework

openEuler is an innovative open source OS platform built on kernel innovations and a solid cloud base to cover all scenarios. It is built on the latest trends of interconnect buses and storage media, and offers a distributed, real-time acceleration engine and base services. It provides competitive advantages in edge and embedded scenarios, and is the first step to building an all-scenario digital infrastructure OS.

openEuler 24.03 LTS SP3 runs on Linux kernel 6.6 and provides POSIX-compliant APIs and OS releases for server, cloud native, edge, and embedded environments. It is a solid foundation for intelligent collaboration across hybrid and heterogeneous deployments. openEuler 24.03 LTS SP3 is equipped with a distributed soft bus and K3s edge-cloud collaboration framework, among other premium features, making it a perfect choice for collaboration over digital infrastructure and everything connected models.

In the future, the openEuler open source community will continue to innovate, aiming to promote the ecosystem and consolidate the digital infrastructure.

Cloud base:

- **KubeOS for containers**: In cloud native scenarios, the OS is deployed and maintained in containers, allowing the OS to be managed based on Kubernetes, just as service containers.
- **Secure containers**: Compared with the traditional Docker+QEMU solution, the iSulad+shimv2+StratoVirt secure container solution reduces the memory overhead and boot time by 40%.
- **Confidential containers**: The iSulad+Kuasar+secGear confidential container solution offers privacy protection while remaining compatible with the cloud-native ecosystem.
- **Confidential computing**: Based on the Arm Confidential Compute Architecture (Arm CCA), this is the first community release to support CCA confidential VMs, providing confidentiality and integrity protection for data and code, even against privileged infrastructure software or cloud service providers.

All scenarios:

- **Edge computing**: openEuler 24.03 LTS SP3 Edge is released for edge computing scenarios. It integrates the Kubernetes edge-cloud collaboration framework to provide unified management, provisioning of edge and cloud applications, and other capabilities.

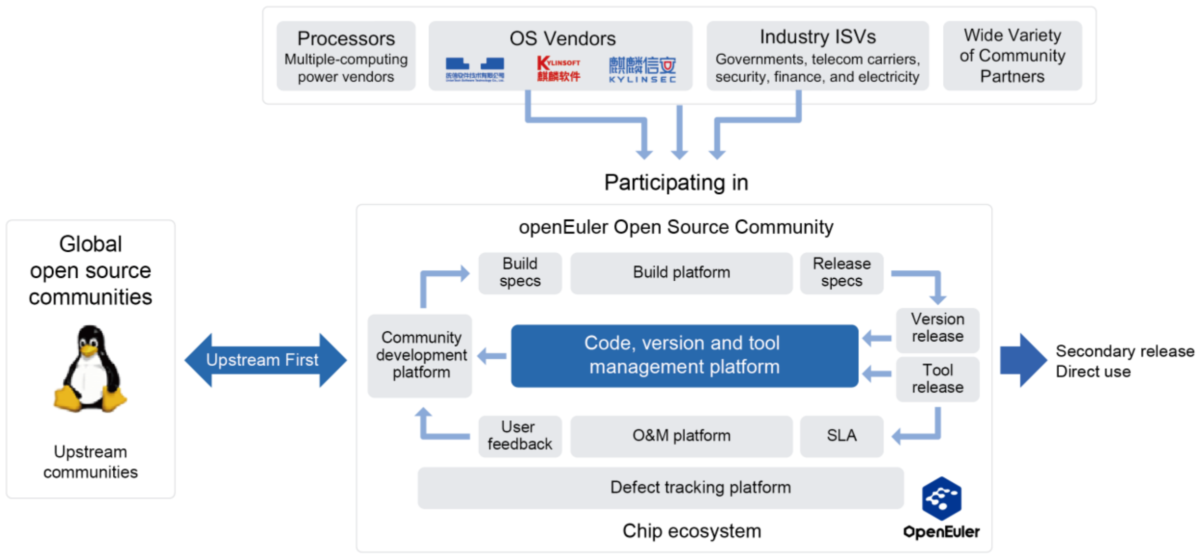
- **Embedded**: openEuler 24.03 LTS SP3 Embedded features an image smaller than 5 MB and a boot time of under 5 seconds. Its elastic virtualization base and mixed-criticality (MICA) deployment framework support concurrent deployment of multiple cores and instances. Cross-OS communication provides an efficient shared-memory mechanism for communication between different OSs.
- **AI-native OS**: The OS enables AI software stacks with out-of-the-box availability. Heterogeneous convergence of memory, scheduling, and training/inference resources reduces AI development costs and improves efficiency. The intelligent interaction platform of the OS streamlines development and administration.
- **SuperPoD OS**: The OS extends the existing framework to support SuperPoDs by building a hierarchical SuperPoD system software stack. It provides SuperPoD capabilities and standardized interfaces, enabling ecosystem development and value realization.

Flourishing community ecosystem:

- **Desktop environments**: UKUI, DDE, Kiran-desktop, and GNOME.
- **openEuler DevKit**: Supports OS migration, compatibility assessment, and various development tools such as secPaver which simplifies security configuration.
- **DevStation**: It offers an intelligent developer workstation that delivers an out-of-the-box, efficient, and secure development environment, streamlining the entire process from coding and deployment to building, packaging, and release. Additionally, it integrates a Model Context Protocol (MCP) AI engine to quickly invoke the community toolchain, improving efficiency from infrastructure setup to application development. It also integrates an intelligent CVE remediation system that significantly improves the response speed and efficiency of security vulnerability fixes.

Platform Framework

The openEuler open source community partners with upstream and downstream communities to advance the evolution of openEuler versions.


Processors
Multiple-computing power vendors
OS Vendors
Industry ISVs
Governments, telecom carriers, security, finance, and electricity
Wide Variety of Community Partners
Participating in
Global open source communities
Upstream communities
Upstream First
openEuler Open Source Community
Build specs
Build platform
Release specs
Version release
Community development platform
Code, version and tool management platform
Tool release
User feedback
O&M platform
SLA
Defect tracking platform
Chip ecosystem
OpenEuler
Secondary release
Direct use

Hardware Support

The openEuler open source community works with multiple vendors to build a vibrant southbound ecosystem. With participation of major chip vendors including Intel and AMD, all openEuler versions support x86, Arm, LoongArch, and RISC-V CPU architectures, and a wide range of CPU chips, such as Loongson 3 series, Zhaoxin KaiXian and KaiSheng, Intel Sierra Forest and Granite Rapids, and AMD EPYC 4/5. openEuler can run on servers from multiple hardware vendors and is compatible with NIC, RAID, Fibre Channel, GPU & AI, DPU, SSD, and security cards.

openEuler supports the following CPU architectures:

Hardware Type	x86_64	AArch64	LoongArch	RISC-V
CPU	Intel, AMD, Hygon, Zhaoxin	Kunpeng, Phytium	Loongson	Sophgo, THead, etc.

Visit https://www.openeuler.org/en/compatibility/ to see the full hardware list.

3 Operating Environments

Servers

To install openEuler on a physical machine, check that the physical machine meets the compatibility and hardware requirements.

For a full list, visit https://openeuler.org/en/compatibility/.

Item	Configuration Requirement
Architecture	AArch64, x86_64, RISC-V, LoongArch64
Memory	≥ 4 GB
Drive	≥ 20 GB

VMs

Verify VM compatibility when installing openEuler.

Hosts running on openEuler 24.03 LTS SP3 support the following software packages:

- libvirt-9.10.0-24.oe2403sp3
- libvirt-client-9.10.0-24.oe2403sp3
- libvirt-daemon-9.10.0-24.oe2403sp3
- qemu-8.2.0-57.oe2403sp3
- qemu-img-8.2.0-57.oe2403sp3

openEuler 24.03 LTS SP3 is compatible with the following guest OSs for VMs:

Guest OS	Architecture
CentOS 6	x86_64
CentOS 7	AArch64
CentOS 7	x86_64

Guest OS	Architecture
CentOS 8	AArch64
CentOS 8	x86_64
Windows Server 2016	x86_64
Windows Server 2019	x86_64
Item	Configuration Requirement
Architecture	AArch64, x86_64
CPU	≥ 2 CPUs
Memory	≥ 4 GB
Drive	≥ 20 GB

Edge Devices

To install openEuler on an edge device, check that the edge device meets the compatibility and minimum hardware requirements.

Item	Configuration Requirement
Architecture	AArch64, x86_64
Memory	≥ 4 GB
Drive	≥ 20 GB

Embedded Devices

To install openEuler Embedded on an embedded device, check that the embedded device meets the compatibility and minimum hardware requirements.

Item	Configuration Requirement
Architecture	AArch64, AArch32, x86_64
Memory	≥ 512 MB
Drive	≥ 256 MB

4 Scenario-specific Innovations

AI

AI is redefining OSs by powering intelligent development, deployment, and O&M. openEuler supports general-purpose architectures like Arm, x86, and RISC-V, and next-gen AI processors like NVIDIA and Ascend. Further, openEuler is equipped with extensive AI capabilities that have made it a preferred choice for diversified computing power.

OS for AI

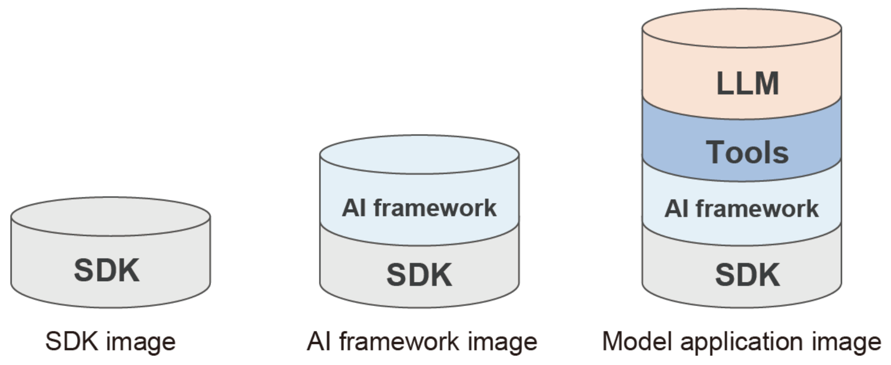
Ready-to-Use Availability

openEuler offers an efficient development and runtime environment that containerizes software stacks of AI platforms with out-of-the-box availability. It also provides various AI frameworks to facilitate AI development.

Feature Description

openEuler supports TensorFlow, PyTorch, and MindSpore frameworks and software development kits (SDKs) of major computing architectures, such as Compute Architecture for Neural Networks (CANN) and Compute Unified Architecture (CUDA), to make it easy to develop and run AI applications.

Environment setup is further simplified by containerizing software stacks. openEuler provides three types of container images:



- **SDK images**: Use openEuler as the base image and install the SDK of a computing architecture, for example, Ascend CANN and NVIDIA CUDA.
- **AI framework images**: Use an SDK image as the base and install AI framework software, such as PyTorch and TensorFlow. Users can use an AI framework image to quickly build a distributed AI framework, such as Ray.
- **Model application images**: Provide a complete set of toolchains and model applications.

For details, see openEuler AI Container Image User Guide.

Application Scenarios

openEuler uses AI container images to simplify deployment of runtime environments. Users can select the container image that best suits their requirements and complete the deployment in a few simple steps.

- **SDK images**: Users can develop and debug Ascend CANN or NVIDIA CUDA applications using an SDK image, which provides a compute acceleration toolkit and a development environment. These containers offer an easy way to perform high-performance computing (HPC) tasks, such as large-scale data processing and parallel computing.

- **AI framework images**: This type of containers is designed to support AI model development, training, and inference.
- **Model application images**: Such an image contains a complete AI software stack and purpose-built models for model inference and fine-tuning.

sysHAX

Feature Description

The sysHAX large language model (LLM) heterogeneous acceleration runtime enhances model inference performance in single-server, multi-xPU setups by optimizing Kunpeng + xPU (GPU/NPU) resource synergy.

- **CPU inference acceleration**: Improves CPU throughput via NUMA-aware scheduling, parallelized matrix operations, and SVE-optimized inference operators.
- **Heterogeneous converged scheduling**: When GPUs are fully loaded, the prefill phase of an inference request can be executed on GPUs while the decode phase is handled by CPUs.

Application Scenarios

sysHAX is used to optimize Transformer models, including DeepSeek and Qwen. Its CPU inference acceleration capability has been adapted for DeepSeek 7B/14B/32B and Qwen2.5/Qwen3 series models on mainstream hardware platforms such as NVIDIA and Ascend 310P. sysHAX fits into the following application scenario:

Data centers: sysHAX assigns inference tasks to CPUs to fully utilize CPU resources and increase the concurrency and throughput of LLMs.

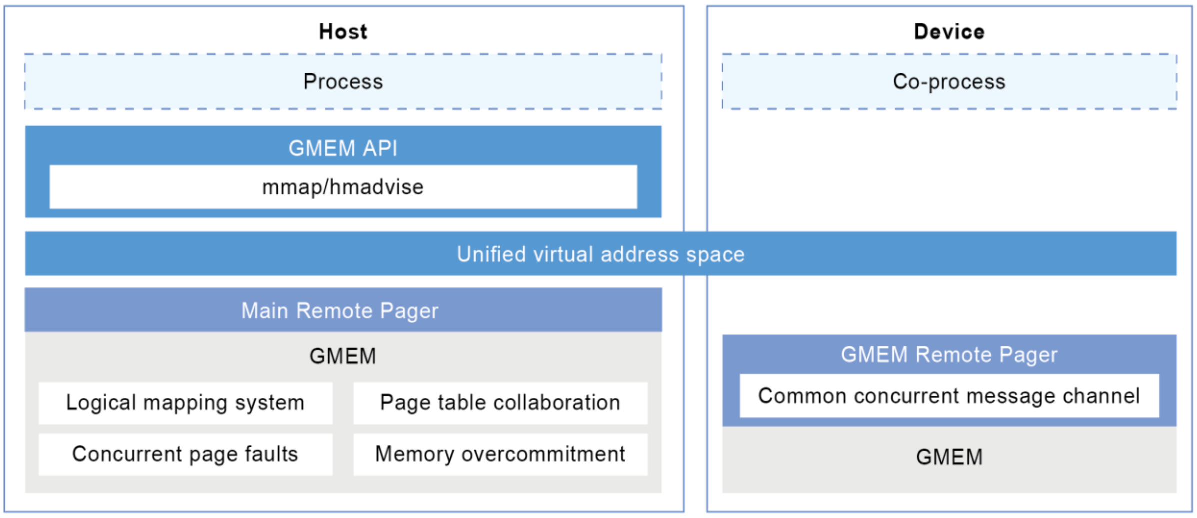
GMEM

In the post-Moore era, there have been breakthroughs in GPUs, TPUs, FPGAs, and other dedicated heterogeneous accelerators. Similar to CPUs, these devices increase computing speeds by storing data in local memory (such as LPDDR SDRAM or HBM), but such design catalyzes more complicated memory systems. Modern memory systems have the following defects:

- Memory management is split between CPUs and accelerators. Explicit data migration makes it difficult to balance the usability and performance of accelerators' memory.
- The high bandwidth memory (HBM) available on accelerators is often insufficient for foundation models. Manual swapping is only feasible in limited scenarios and typically results in significant performance degradation.
- A large number of invalid data migrations occur in search & recommendation and big data scenarios, and no efficient memory pooling solution is available.

Heterogeneous Memory Management (HMM) is a Linux feature that is plagued by issues of poor programming, performance, and portability, while also relying heavily on manual tuning. As such, it is unfavored by most OS communities, and has fueled demand for an efficient solution for heterogeneous accelerators. Generalized Memory Management (GMEM) is one new option, which offers a centralized management mechanism for heterogeneous memory connections. GMEM APIs are compatible with native Linux APIs, and feature high usability, performance, and portability. After an accelerator calls GMEM APIs to connect its memory to the unified address space, the accelerator automatically obtains the programming optimization capability for heterogeneous memory, and does not need to execute the memory management framework multiple times. This greatly reduces development and maintenance costs. Developers can apply for and release a unified set of APIs to achieve heterogeneous memory programming without memory migrations. If the HBM of an accelerator is insufficient, GMEM can use the CPU memory as the accelerator cache to transparently over-allocate the HBM without manual swapping. GMEM offers an efficient memory pooling solution thanks to a shared memory pool that eliminates the need for duplicate migrations.

Feature Description



GMEM enhances memory management in the Linux kernel. Its logical mapping system masks the differences between the ways how the CPU and accelerator access memory addresses. The Remote Pager memory message interaction framework provides the device access abstraction layer. In the unified address space, GMEM automatically migrates data to the OS or accelerator when data is to be accessed or paged.

- GMEM

  GMEM leverages the computing power of both CPUs and accelerators. It combines the two independent address spaces of the OS and accelerators into a unified virtual address space, to realize unified memory management and transparent memory access.

  GMEM uses a collection of new logical page tables to manage the unified virtual address space, ensuring data consistency between the page tables of different CPUs and micro architectures. Based on the memory access consistency mechanism of the logical page tables, the target memory space can be migrated between the host and accelerator using a kernel page fault handling procedure. If the accelerator's memory space is insufficient, the accelerator can borrow memory from the host and reclaim its cold memory to achieve memory overcommitment. This practice removes issues in which model parameters are restricted by the accelerator's memory, and reduces the cost of foundation model training.

  GMEM high-level APIs in the kernel allow the accelerator driver to directly use memory management functions by registering the MMU functions defined in the GMEM specification. With memory management functions, the accelerator can create logical page tables and perform memory overcommitment. The logical page tables decouple the high-layer logic of memory management from the CPU's hardware layer, so as to abstract the high-layer memory management logic that can be reused by different accelerators. This design only requires the accelerator to register bottom-layer functions, and does not need high-layer logic for the unified address space. The OS automatically manages data migration across different storage media, such as DRAM and HBM.

- Remote Pager

  Remote Pager is a memory message interaction framework that adopts message channel, process management, memory swap, and memory prefetch modules for the collaboration between the host and accelerator. It is enabled by the independent driver **remote_pager.ko**. The Remote Pager abstraction layer simplifies device adaptation by enabling third-party accelerators to easily access the GMEM system.

- User APIs

  To allocate the unified virtual memory, users can directly use the mmap system call. GMEM adds the flag (**MMAP_PEER_SHARED**) of allocating the unified virtual memory to mmap.

  The libgmem user-space library provides the hmadvise API of memory prefetch semantics that helps optimize the accelerator memory access. For details, visit https://atomgit.com/openeuler/libgmem/blob/master/README.md.

- Constraints

  Hardware: Ascend NPUs are required.

  Software: GMEM supports only 2 MB huge pages. Therefore, transparent OS huge pages must be enabled on hosts and NPUs to use GMEM. The heterogeneous memory obtained using **MAP_PEER_SHARED** cannot be inherited during fork.

For details about how to use GMEM, visit:

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

Application Scenarios

- Heterogeneous unified memory programming

  To simplify heterogeneous memory programming, GMEM can allocate a unified virtual memory to CPUs and accelerators, which then share the same pointer. For example, an NPU memory management system can be connected to GMEM with just 100 lines of modified code in the NPU driver, a huge reduction on the original 4,000 lines in the memory management framework.

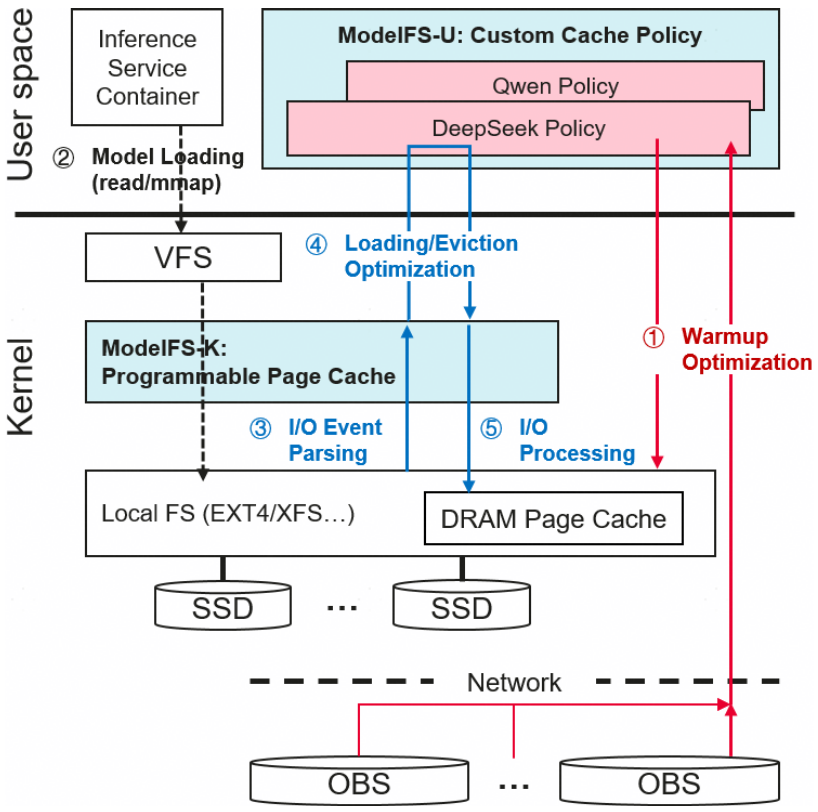
- Automatic memory overcommitment of an accelerator

  When a GMEM API is used to allocate memory to an application, the application is not limited by the physical memory capacity of the accelerator. Instead, the application can transparently over-allocate memory until the CPU's DRAM capacity is exhausted. GMEM swaps cold device memory pages out to the CPU memory to achieve high-performance and easy-to-start training and inference. This design makes GMEM deliver 1.6 times the performance of NVIDIA-UVM in ultra-large model training when the overcommitment ratio is 200%. (Test results compared the Ascend 910 NPU and NVIDIA A100 GPU under the same HBM conditions.)

ModelFS

ModelFS addresses the model loading bottleneck during the startup phase of large language model (LLM) inference. While existing approaches often sacrifice compatibility for performance, real-world industrial deployment demands seamless integration with existing ecosystems. ModelFS bridges this gap by achieving state-of-the-art loading performance through optimized file system cache policies, without compromising compatibility. Specifically, it introduces a non-intrusive, flexible, and lightweight programmable page cache framework in the kernel, enabling users to customize file system page cache policies. Building on this framework, we provide a reference implementation optimized to accelerate model loading.

Feature Description



- ModelFS-K

  The kernel module of ModelFS provides a programmable framework for the file system page cache. Its core design is a stacked file system that can be mounted on top of an existing file system. ModelFS-K redirects the logic of the underlying file system to user-space **prefetch()** and **evict()** functions via user-space procedure calls (UPCs).

- ModelFS-U

  The user-space module of ModelFS provides the runtime environment for cache policies. It offers a VFS-like programming framework in user space, allowing model implementers and I/O optimizers to customize cache policies based on the I/O characteristics of model loading. Users need to implement the **init()**, **exit()**, **prefetch()**, and **evict()** functions. ModelFS-U registers these functions with ModelFS-K and is responsible for parsing I/O events as well as performing asynchronous data prefetching and eviction at runtime.

Application Scenarios

ModelFS can accelerate I/O operations across multiple types of storage backends, including local file systems, distributed file systems, and object storage service (OBS). The following are two typical use cases.

- Remote OBS weight prefetching

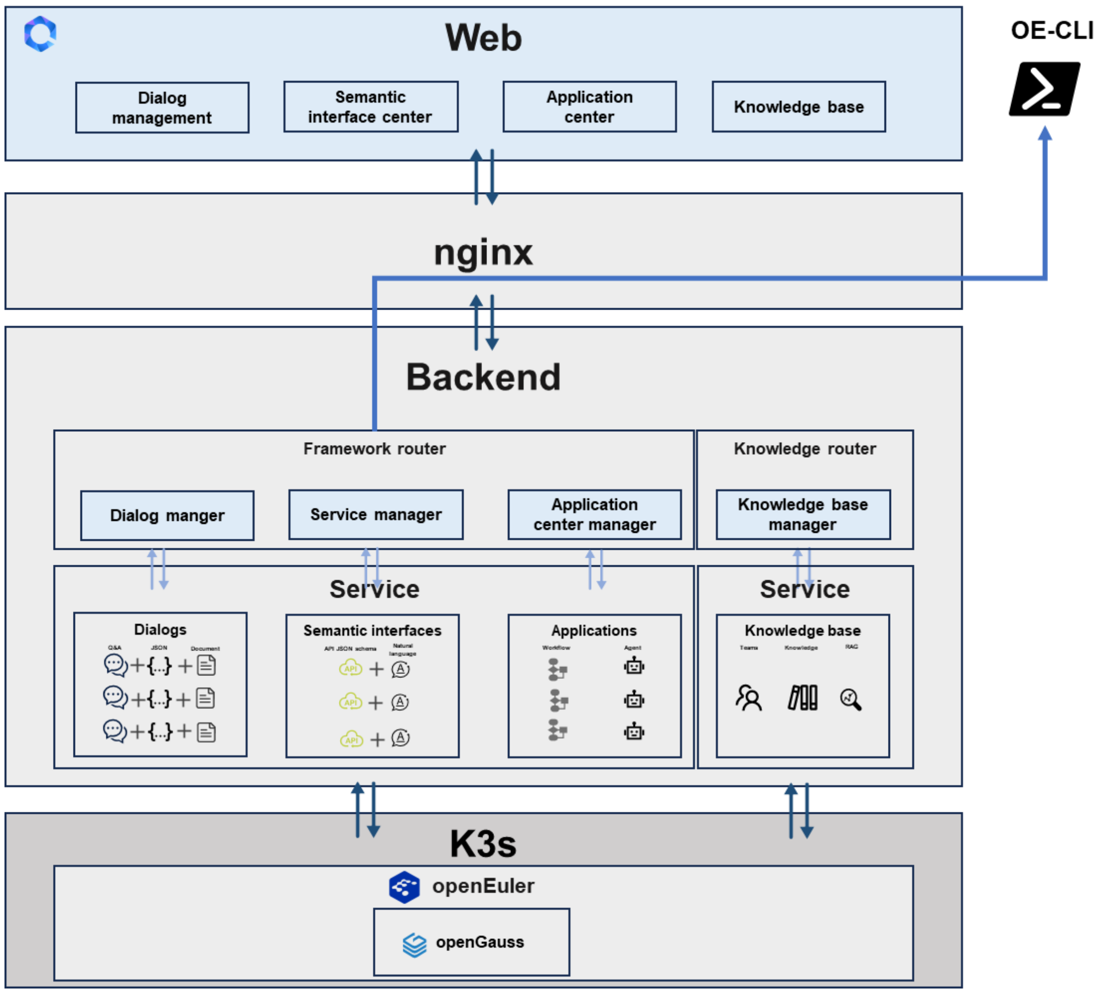
  When model weights are stored in remote OBS, the inference system typically prefetches hotspot weights to local storage in advance due to the low bandwidth of OBS. ModelFS can accelerate weight prefetching by implementing a cache policy that aggregates I/Os across multiple OBS buckets.

- Local SSD weight loading acceleration

When model weights are stored on local SSDs, the performance bottleneck often arises from low bandwidth utilization. ModelFS addresses this issue with an I/O template-based mechanism. During the first run, it traces the I/O loading behavior of each inference service. In subsequent inference startups, it accurately predicts future I/O behavior and fully leverages SSD bandwidth and XPU affinity to accelerate prefetching and eviction.

AI for OS

AI makes openEuler more intelligent. openEuler Intelligence is an AI-powered Q&A platform built on openEuler data. It enables workflow orchestration through semantic interfaces and agent building through MCP. Additionally, it integrates some system services to further improve the intelligence of openEuler.



Intelligent Q&A

Feature Description

The openEuler Intelligence system is accessible via web or shell.

- **Web**: Through the visual web entry, users can easily build and use the knowledge base, run automated tests on the knowledge base, and view test evaluation results. They can also register OpenAPI APIs, build and use workflow applications based on them, register and activate MCP, and build and use MCP-based agent applications. The web entry helps beginners access openEuler knowledge and leverage openEuler's AI capabilities effortlessly.

- **Shell**: Through the shell entry, users can trigger intelligent Q&A or pre-integrated agent applications within the agent framework. The shell entry allows O&M personnel to interact with openEuler in an OS-affinity and intelligent manner, reducing O&M costs.

Intelligent Planning, Scheduling, and Recommendation

- **Intelligent planning**: The agent applications of openEuler Intelligence can plan steps in real time based on user input and available tools, continuing until the user's objective is achieved or the maximum number of steps is reached.
- **Intelligent scheduling**: openEuler Intelligence allows users to define multiple workflows within a workflow application. When a query request is made, openEuler Intelligence automatically extracts the relevant parameters and selects the most suitable workflow to execute the query task.
- **Intelligent recommendation**: Based on users' query requests and workflow execution results, openEuler Intelligence recommends workflows that may be useful in future tasks, increasing the likelihood of task completion and making applications easier to use.

Workflow Applications

- **Semantic interfaces**: A semantic interface contains natural language comments. openEuler Intelligence supports two methods for registering semantic interfaces. The first method allows users to register APIs with openEuler Intelligence as OpenAPI (3.0 or later) YAML files. When writing an OpenAPI YAML file, users can add natural language comments to the APIs. When these APIs are called, the foundation model selects the right APIs and sets their parameters based on the comments. The second method allows users to register functions with openEuler Intelligence as Python code, and this method provides a more user-friendly option for using openEuler Intelligence. The semantic interfaces generated in both methods can be orchestrated into workflows in a visualized manner.
- **Workflow orchestration and invocation**: openEuler Intelligence enables users to visually connect built-in semantic interfaces and user-registered interfaces to create workflows. Users can debug these workflows, and then release and use them as applications. When workflows are debugged and executed, intermediate results are displayed to help lower debugging costs and enhance the overall user experience.

Agent Applications

- **MCP registration, installation, and activation**: MCP is a mainstream AI-related protocol. It uses SDKs to encapsulate complex and diverse services with natural semantic information, allowing AI to easily invoke tools and services reconstructed based on MCP.
- **Agent building and use**: openEuler Intelligence allows building agents based on MCP and various foundation models. These agents can decompose a user's objective into phased tasks using the configured model information and the user-provided objective. MCP tools are then used to complete each task until the user's objective is achieved.

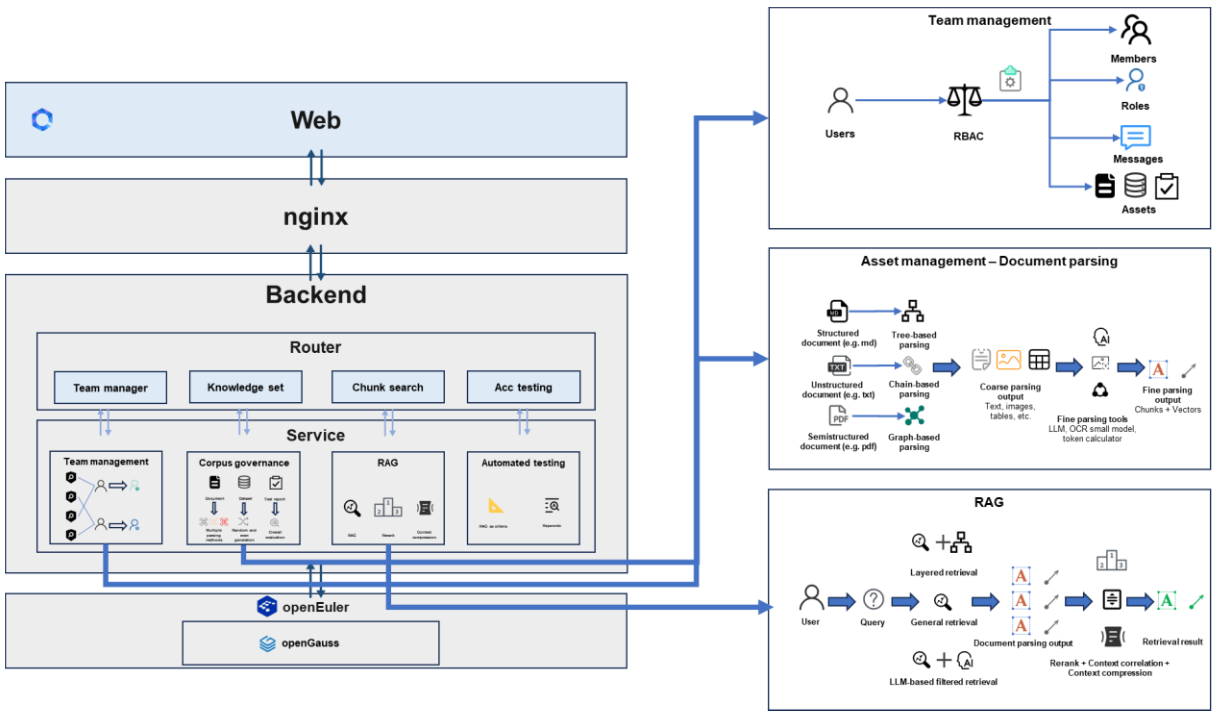
RAG

Retrieval-augmented generation (RAG) is a technique for enhancing the long-term memory capability of large language models (LLMs). Used by the openEuler Intelligence system, RAG is essential to reducing model training costs and has the following highlights:

- **Pre-processing for retrieval**: When a user's query request lacks sufficient information, it can be rewritten based on historical context and inferred intent to enhance retrieval accuracy.

- **Knowledge indexing**: For documents of various formats and content types, openEuler Intelligence offers document parsing capabilities such as summarization, text feature extraction, tree-structured parsing, and optical character recognition (OCR). These capabilities help generate segment-level feature indexes to increase hit rates.
- **Retrieval enhancement**: openEuler Intelligence supports eight retrieval methods. Among them, the methods using dynamic keyword weighting and LLM-based filtering significantly boost out-of-the-box accuracy.
- **Post-processing for retrieval**: For fragments with missing context, a uniform randomized context completion method is provided to enhance the completeness of fragment information and reduce hallucination in model fitting. For fragments (sets) with the maximum number of tokens, various methods are provided to enable intelligent Q&A for scenarios with limited resources. These methods include the reranking method based on the Jaccard distance, token compression method that removes common words and applies random discarding, and token truncation method based on binary search.

These capabilities enable RAG for the openEuler Intelligence system to accommodate to more document formats and content types, and enhance Q&A services with minimal impact on system load.



Team Management

Team management is a fundamental capability of the RAG technology in openEuler Intelligence. It leverages role-based access control (RBAC) to manage team members' access to roles, users, and assets, thereby enhancing the overall usability of the knowledge base.

- **Roles**: This includes editing the atomic permissions of roles, adding, deleting, and modifying roles, and assigning roles to members. Atomizing these capabilities is central to instantiating team permissions.
- **Members**: This includes processing invitations and requests, as well as adding, deleting, modifying, and querying members. Atomizing these capabilities is central to managing team members and assigning permissions.

- **Assets**: This includes adding, deleting, modifying, and querying the asset library, as well as the documents, datasets, and testing results within it. Atomizing these capabilities is central to enabling on-demand access to asset data.

Corpus Governance

Corpus governance is one of the basic RAG capabilities in the openEuler Intelligence system. It imports corpuses into the knowledge base in a supported format using context location extraction, text summarization, and OCR, increasing the retrieval hit rate.

- **Context location extraction**: The relative location relationship within a document is retained. Specifically, the global and local relative offsets of each segment are recorded to support context completion.
- **Text summarization**: Sliding windows and LLMs are used to generate text summaries for complex documents or segments, supplying foundational data for multi-level retrieval.
- **OCR augmentation**: For documents containing both images and text, summaries are generated based on image text and surrounding context to provide foundational data for answering image-related questions.
- **Document source tracing**: During answer generation, openEuler Intelligence displays a note in the lower right corner of the corresponding sentence based on the returned segment and related document information. Users can click the note to view the document name and abstract, enhancing the reliability of question answering.

Automated Testing

Automated testing is a core RAG capability in openEuler Intelligence. It detects shortcomings in the knowledge base and retrieval augmentation algorithms through automated dataset generation and evaluation.

- **Dataset generation**: After a user selects a parsed document, openEuler Intelligence randomly samples a portion of the parsing results and sends them to an LLM. The LLM then generates and filters Q&A pairs to create a high-quality test dataset that contains queries, reference answers, and original segments.
- **Test evaluation**: Automated tests are conducted using the user-generated dataset and the predefined parameters such as the retrieval augmentation algorithm, LLM, and top-k fragments. Tests score are calculated based on the queries, reference answers, original fragments, associated fragments, and model fitting results in the test dataset. After that, the test results are evaluated using metrics such as the precision rate, recall rate, fidelity value, explainability, longest common substring score, edit distance score, and Jaccard similarity coefficient.

Application Scenarios

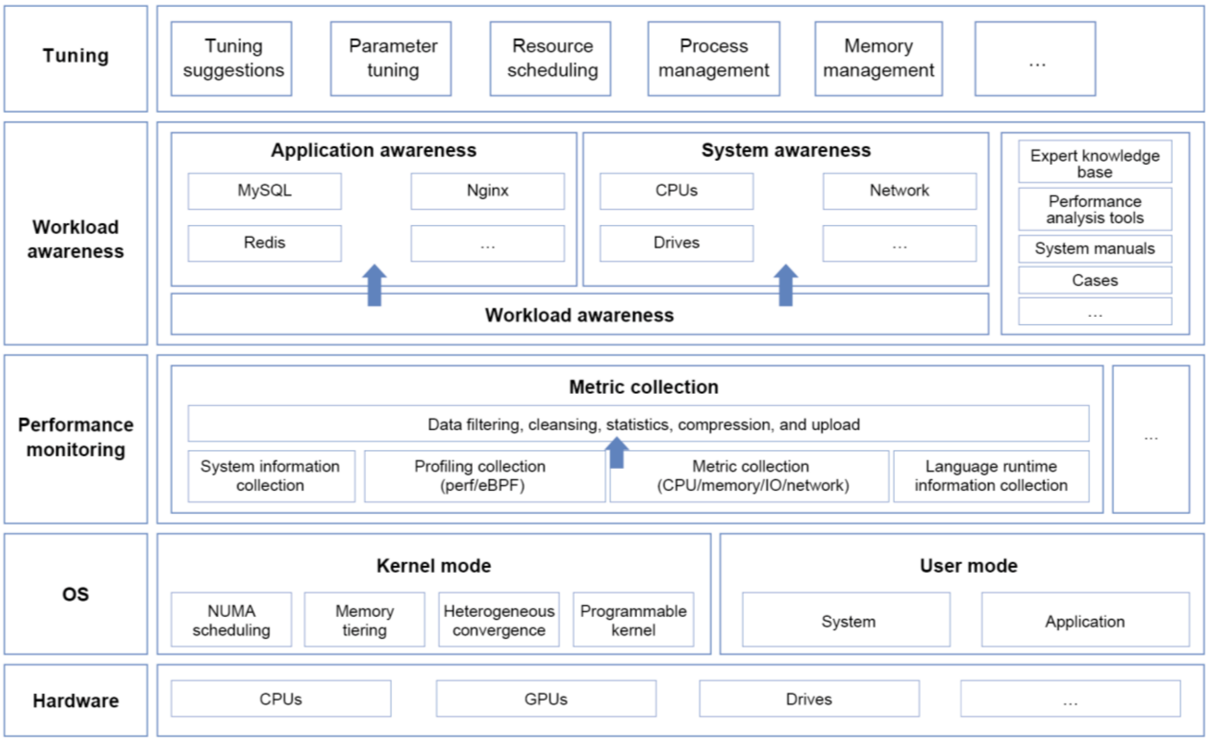
- Common openEuler users can build customized smart assistants using local files or documents from the openEuler community, such as white papers.
- openEuler developers can build custom workflows or agent applications using the web client, reducing repetitive work in certain scenarios, such as code audits.
- openEuler O&M personnel can interact with the OS using natural language through the shell, reducing the effort required to construct and modify complex commands.

  For details, see openEuler Intelligence.

Intelligent Tuning

Feature Description

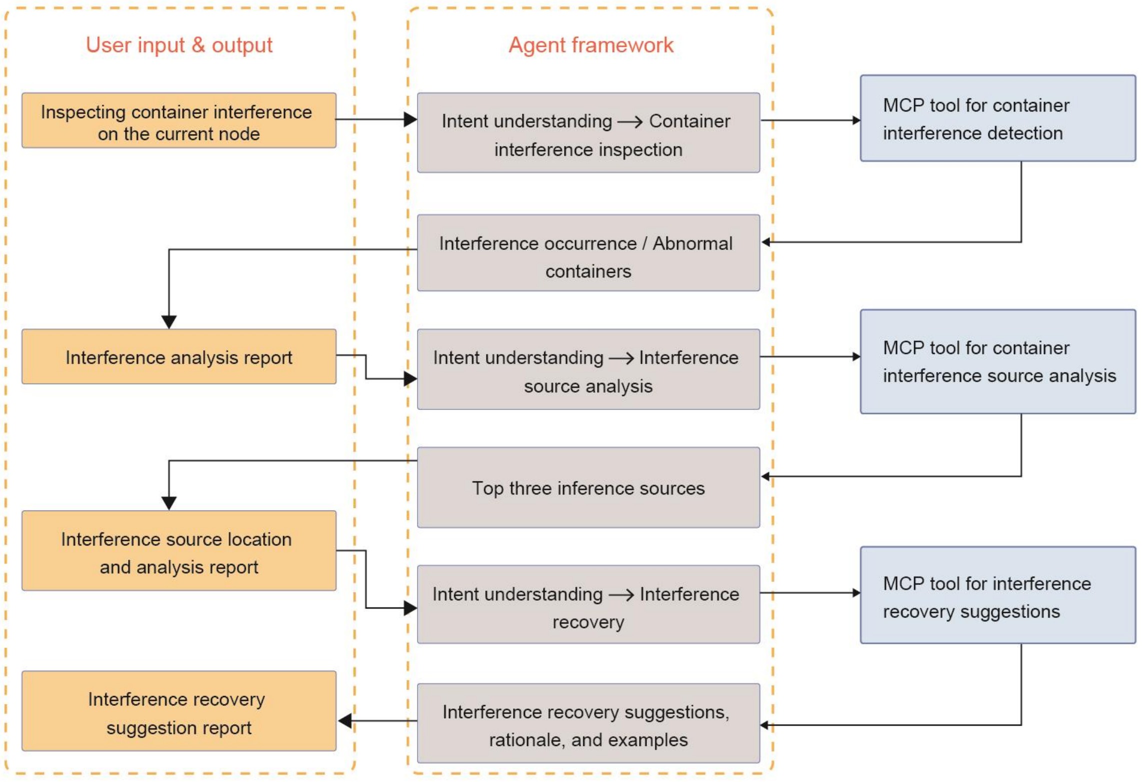
The openEuler Intelligence system supports the intelligent shell entry. Through this entry, users can interact with openEuler Intelligence using a natural language and perform heuristic tuning operations such as performance data collection, system performance analysis, and system performance tuning. The MCP protocol can be used to detect tuning intents.



Application Scenarios

- **Gaining insights from key performance metrics**: Users can learn about the system performance status based on collected performance metrics like CPU, I/O, drive, network, microarchitecture, and application.
- **Analyzing system performance**: Performance analysis reports are generated, making it easier to locate performance bottlenecks across the entire system and in individual applications.
- **Receiving performance tuning suggestions**: Based on current system performance, openEuler Intelligence can recommend appropriate application or system parameters, which are automatically applied. It can also generate a one-click performance tuning script to optimize both the system and specific applications.

Intelligent Diagnosis


User input & output
Agent framework
Inspecting container interference on the current node
Intent understanding → Container interference inspection
MCP tool for container interference detection
Interference occurrence / Abnormal containers
Interference analysis report
Intent understanding → Interference source analysis
MCP tool for container interference source analysis
Top three inference sources
Interference source location and analysis report
Intent understanding → Interference recovery
MCP tool for interference recovery suggestions
Interference recovery suggestion report
Interference recovery suggestions, rationale, and examples

Feature Description

- **Interference detection**: The container interference detection tool monitors nodes over a specified period, identifies interference, and reports the affected containers along with the corresponding metrics.
- **Interference source analysis**: The container interference source analysis tool identifies all affected containers on a node and reports the top three suspected interference source containers, along with the associated metrics (such as CPU run delay) for each affected container.
- **Interference recovery suggestion**: Based on the results of interference source analysis, the interference recovery suggestion tool analyzes the interference and generates a report with recommended recovery actions, including the rationale and example commands for each suggestion.

Application Scenarios

The intelligent diagnosis interface in openEuler 24.03 LTS SP3 provides three key functions: container interference detection, interference source analysis, and interference recovery suggestion generation.

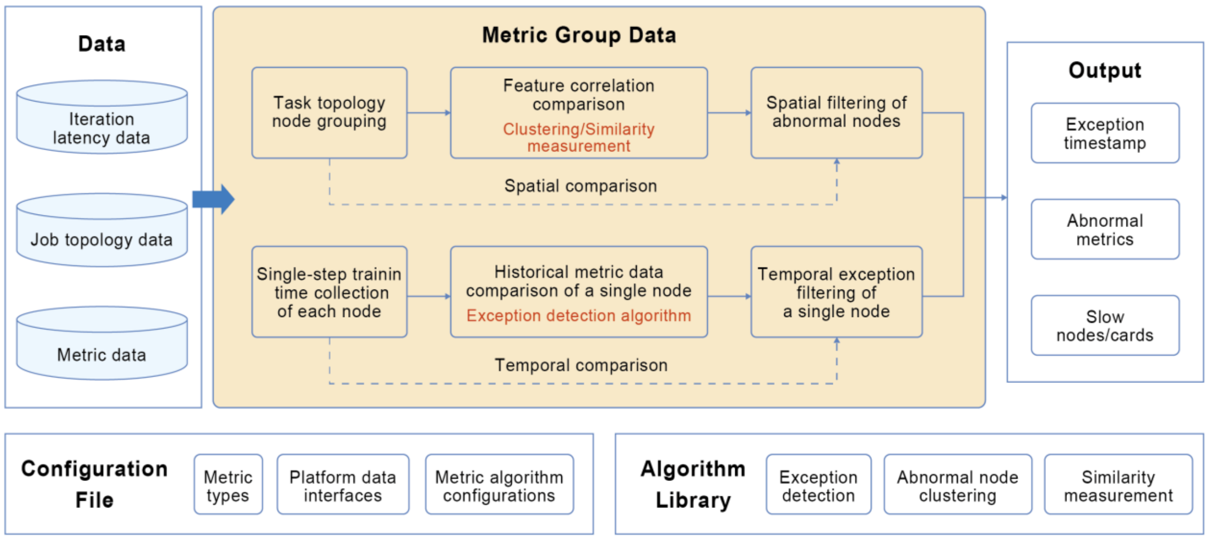
- Container interference detection identifies interference in monitored container metrics on a single node and reports the affected containers.
- Interference source analysis determines the root causes based on detection results and node monitoring metrics, outputting the top three contributing metrics for each affected container.

- Interference recovery suggestion generation produces a report with recommended recovery actions based on the results of interference source analysis.

AI Cluster Slow-Node Demarcation

Performance degradation during AI cluster training is inevitable and often results from a wide range of complex factors. Existing solutions rely on log analysis after performance degradation occurs. However, it can take 3 to 4 days from log collection to root cause diagnosis and issue resolution on the live network. To address these pain points, an online slow node detection solution is offered. This solution allows for real-time monitoring of key system metrics and uses model- and data-driven algorithms to analyze the observed data and pinpoint slow or degraded nodes. This facilitates system self-healing and fault rectification by O&M personnel.

Feature Description


Data
Iteration latency data
Job topology data
Metric data
Metric Group Data
Task topology node grouping
Feature correlation comparison
Clustering/Similarity measurement
Spatial filtering of abnormal nodes
Spatial comparison
Single-step trainin time collection of each node
Historical metric data comparison of a single node
Exception detection algorithm
Temporal exception filtering of a single node
Temporal comparison
Output
Exception timestamp
Abnormal metrics
Slow nodes/cards
Configuration File
Metric types
Platform data interfaces
Metric algorithm configurations
Algorithm Library
Exception detection
Abnormal node clustering
Similarity measurement

Grouped metric comparison helps detect slow nodes and cards in AI cluster training. This technology is built on Systrace and includes a configuration file, an algorithm library, and a slow node analysis mechanism based on both time and space dimensions. It outputs the exception timestamp, abnormal metrics, and IP addresses of slow nodes and cards. This technology enhances overall system stability and reliability.

- **Configuration file**: Contains the types of metrics to be observed, configuration parameters for the metric algorithms, and data interfaces, which are used to initialize the slow node detection algorithms.
- **Algorithm library**: Includes common time series exception detection algorithms, such as Streaming Peaks-over-Threshold (SPOT), k-sigma, abnormal node clustering, and similarity measurement.
- **Data**: Metric data collected from each node is represented by a time sequence.
- **Grouped metric comparison**: Supports spatial filtering of abnormal nodes and temporal exception filtering of a single node. Spatial filtering identifies abnormal nodes based on the exception clustering algorithm, while temporal exception filtering determines whether a node is abnormal based on the historical data of the node.

Application Scenarios

Systrace provides capabilities for detecting slow nodes, displaying alarms, and writing exception information to drives.

- **AI model training**: This feature quickly detects slow nodes for training jobs in large-scale AI clusters, facilitating system self-healing and fault rectification by O&M personnel.
- **AI model inference**: This feature identifies performance degradation across multiple instances of the same model. By comparing the resource usage of multiple instances, it quickly locates underperforming instances to support inference task scheduling and improve resource utilization.

Intelligent Container Images

Feature Description

The openEuler Intelligence system can invoke environment resources through a natural language, assist in pulling container images for local physical resources, and establish a development environment suitable for debugging on existing compute devices.

This system supports three types of containers, and container images have been released on Docker Hub. Users can manually pull and run these container images.

- **SDK layer**: encapsulates only the component libraries that enable AI hardware resources, such as CUDA and CANN.
- **SDKs + training/inference frameworks**: accommodates TensorFlow, PyTorch, and other frameworks (for example, tensorflow2.15.0-cuda12.2.0 and pytorch2.1.0.a1-cann7.0.RC1) in addition to the SDK layer.
- **SDKs + training/inference frameworks + LLMs**: encapsulates several models (for example, llama2-7b and chatglm2-13b) based on the second type of containers.

The following table lists the container images supported by the openEuler Intelligence system:

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

Registry	Repository	Image Name	Tag
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

Application Scenarios

- **Common openEuler operations**: Simplify the process of building a deep learning development environment while saving physical resources. For example, set up an Ascend development environment on openEuler.
- **openEuler development**: Developers familiarize themselves with the openEuler AI software stack to reduce the trial-and-error cost of installing components.

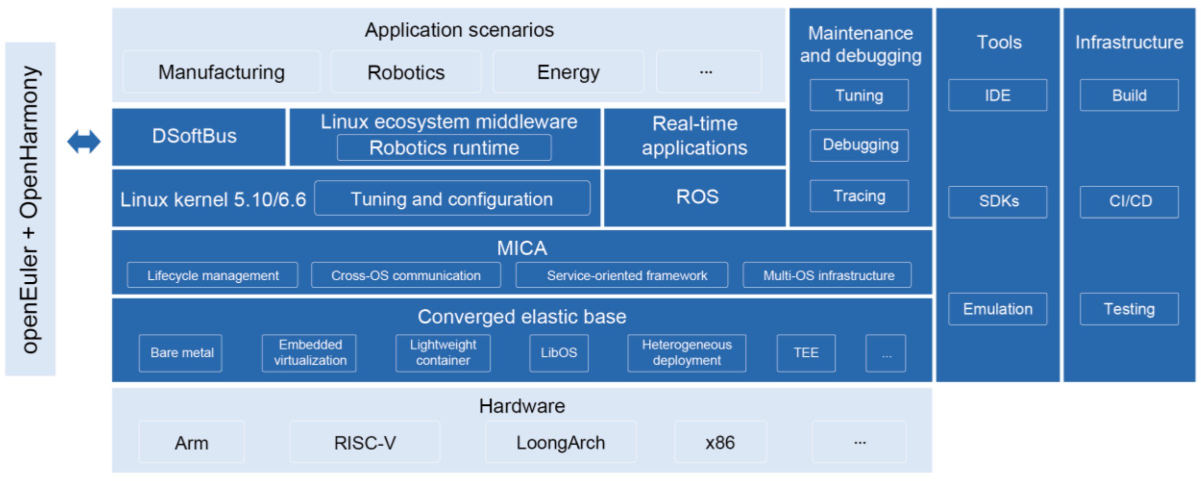
Embedded

openEuler 24.03 LTS SP3 is suited for embedded applications, offering significant progress in southbound and northbound ecosystems, technical features, infrastructure, and implementation over previous generations.

openEuler Embedded provides a closed loop framework often found in operational technology (OT) applications such as manufacturing and robotics, whereby innovations help optimize its embedded system software stack and ecosystem. openEuler Embedded enhances its software package ecosystem by incorporating the oeBridge feature, which supports online software installation from an openEuler mirror site. When building Yocto images, oeBridge can be used to install openEuler RPM packages for easy image customization. openEuler Embedded also supports the oeDeploy feature for quick deployment of AI and cloud-native software stacks. Kernel support in openEuler is enhanced by optimizing the meta-openEuler kernel configuration and the oeAware real-time tuning feature. These updates help control interference and improve real-time system responsiveness.

Future versions of openEuler Embedded will integrate contributions from ecosystem partners, users, and community developers, increase support for chip architectures such as LoongArch and more southbound hardware, and optimize industrial middleware, embedded AI, embedded edge, and simulation system capabilities.

System Architecture



Southbound Ecosystem

openEuler Embedded Linux supports mainstream processor architectures like AArch64, x86_64, AArch32, and RISC-V, and will extend support to LoongArch in the future. openEuler 24.03 and later versions have a rich southbound ecosystem and support chips from Raspberry Pi, HiSilicon, Rockchip, Renesas, TI, Phytium, StarFive, and Allwinner.

Embedded Virtualization Base

openEuler Embedded uses an elastic virtualization base that enables multiple OSs to run on a system-on-a-chip (SoC). The base incorporates a series of technologies including bare metal, embedded virtualization, lightweight containers, LibOS, trusted execution environment (TEE), and heterogeneous deployment.

- The bare metal hybrid deployment solution runs on OpenAMP to manage peripherals by partition at a high performance level; however, it delivers poor isolation and flexibility. This solution supports the hybrid deployment of UniProton/Zephyr/RT-Thread and openEuler Embedded Linux.
- Partitioning-based virtualization is an industrial-grade hardware partition virtualization solution that runs on Jailhouse. It offers superior performance and isolation but inferior flexibility. This solution supports the hybrid deployment of UniProton/Zephyr/FreeRTOS and openEuler Embedded Linux or of OpenHarmony and openEuler Embedded Linux.
- Real-time virtualization is available as two community hypervisors, ZVM (for real-time VM monitoring) and Rust-Shyper (for Type-I embedded VM monitoring).

MICA Deployment Framework

The MICA deployment framework is a unified environment that masks the differences between technologies that comprise the embedded elastic virtualization base. The multi-core capability of hardware combines the universal Linux OS and a dedicated real-time operating system (RTOS) to make full use of all OSs.

The MICA deployment framework covers lifecycle management, cross-OS communication, service-oriented framework, and multi-OS infrastructure.

- Lifecycle management provides operations to load, start, suspend, and stop the client OS.
- Cross-OS communication uses a set of communication mechanisms between different OSs based on shared memory.
- Service-oriented framework enables different OSs to provide their own services. For example, Linux provides common file system and network services, and the RTOS provides real-time control and computing.
- Multi-OS infrastructure integrates OSs through a series of mechanisms, covering resource expression and allocation and unified build.
- The MICA deployment framework provides the following functions:
- Lifecycle management and cross-OS communication for openEuler Embedded Linux and the RTOS (Zephyr or UniProton) in bare metal mode
- Lifecycle management and cross-OS communication for openEuler Embedded Linux and the RTOS (FreeRTOS or Zephyr) in partitioning-based virtualization mode

Northbound Ecosystem

- **Northbound software packages**: Over 700 common embedded software packages can be built using openEuler.
- **Soft real-time kernel**: This capability helps respond to soft real-time interrupts within microseconds.
- **DSoftBus**: The distributed soft bus system (DSoftBus) of openEuler Embedded integrates the DSoftBus and point-to-point authentication module of OpenHarmony. It implements interconnection between openEuler-based embedded devices and OpenHarmony-based devices as well as between openEuler-based embedded devices.
- **Embedded containers and edges**: With iSula containers, openEuler and other OS containers can be deployed on embedded devices to simplify application porting and deployment. Embedded container images can be compressed to 5 MB, and can be easily deployed into the OS on another container.

UniProton

UniProton is an RTOS that features ultra-low latency and flexible MICA deployments. It is suited for industrial control because it supports both microcontroller units and multi-core CPUs. UniProton provides the following capabilities:

- Compatible with processor architectures like Cortex-M, AArch64, x86_64, and riscv64, and supports M4, RK3568, RK3588, x86_64, Hi3093, Raspberry Pi 4B, Kunpeng 920, Ascend 310, and Allwinner D1s.
- Connects with openEuler Embedded Linux on Raspberry Pi 4B, Hi3093, RK3588, and x86_64 devices in bare metal mode.
- Can be debugged using GDB on openEuler Embedded Linux.

Application Scenarios

openEuler Embedded helps supercharge computing performance in a wide range of industries and fields, including industrial and power control, robotics, aerospace, automobiles, and healthcare.

SuperPoD Innovation

The exponential growth in computing power requirements, coupled with breakthroughs in high-speed interconnect technologies, is driving the evolution of hardware from standalone nodes to

SuperPoDs. Going forward, the core requirements for data, resources, and services will exhibit the following characteristics:

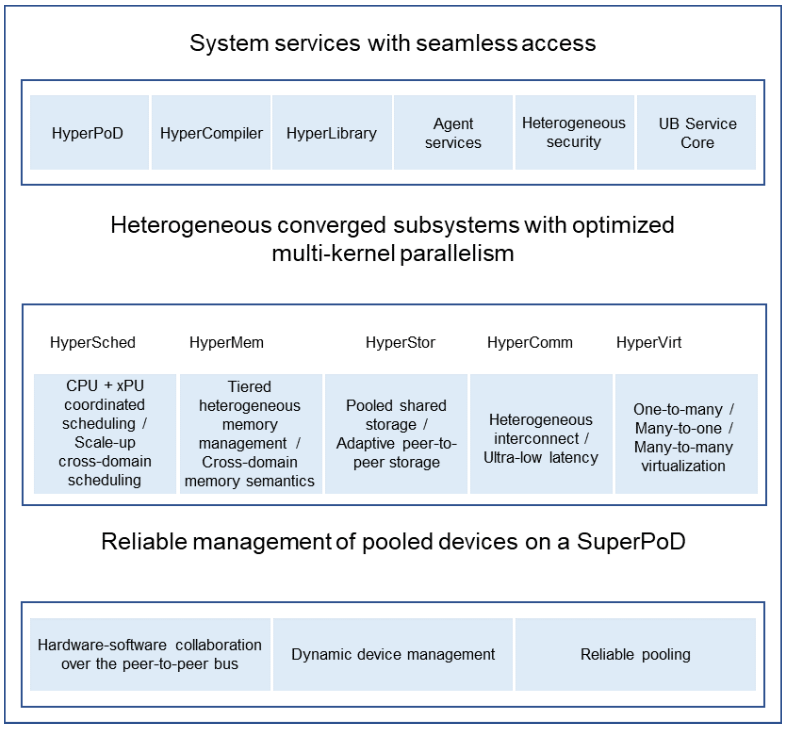
- **Resource pooling**: Compute, memory, interconnect, and storage resources can be pooled to support many-to-many collaboration.
- **Scale expansion**: On-demand, flexible expansion is supported, ensuring high utilization. A unified addressing mechanism is used across the entire system, reducing static transfer and dynamic latency.
- **Long-term stability and reliability**: The system's long-term stable runtime is improved tenfold. It supports automatic recovery, is easy to deploy, and adapts to various equipment room environments. The system can also be expanded from modules to full cabinets.

To meet the requirements for SuperPoD scheduling, pooling, communication, and virtualization, openEuler has been upgraded to support SuperPoDs, unleashing heterogeneous computing.

SuperPoD OS Architecture

During the implementation of SuperPoD applications, the industry faces three core challenges: streamlining the development process and enabling smooth migration of services with zero modifications, accurately matching diversified computing power supply and achieving on-demand, efficient resource release, and managing increasing system complexity while ensuring high application availability. The openEuler heterogeneous convergence system addresses the preceding challenges as follows:

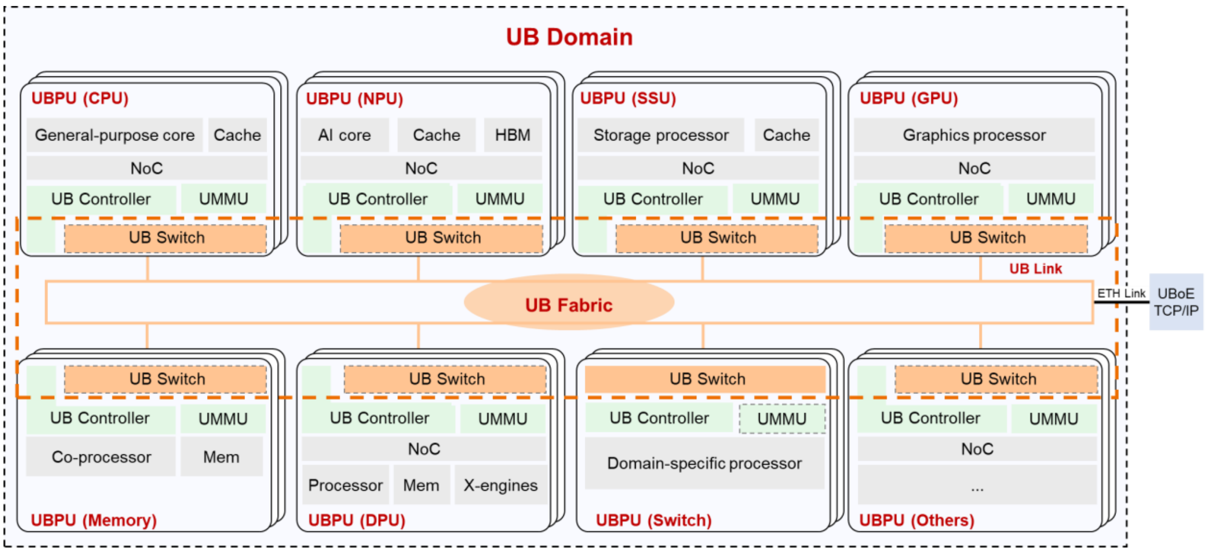
- It introduces advanced system services to streamline service migration, enabling zero-modification deployment and greatly reducing development and migration costs.
- It strengthens the core subsystems for heterogeneous convergence, providing communication and virtualization capabilities that accurately match computing power requirements and maximize resource utilization.
- It enhances device pooling to support flexible SuperPoD expansion. Reliable pooling prevents fault propagation and ensures stable service operation.


System services with seamless access
HyperPoD
HyperCompiler
HyperLibrary
Agent services
Heterogeneous security
UB Service Core
Heterogeneous converged subsystems with optimized multi-kernel parallelism
HyperSched
HyperMem
HyperStor
HyperComm
HyperVirt
CPU + xPU coordinated scheduling / Scale-up cross-domain scheduling
Tiered heterogeneous memory management / Cross-domain memory semantics
Pooled shared storage / Adaptive peer-to-peer storage
Heterogeneous interconnect / Ultra-low latency
One-to-many / Many-to-one / Many-to-many virtualization
Reliable management of pooled devices on a SuperPoD
Hardware-software collaboration over the peer-to-peer bus
Dynamic device management
Reliable pooling

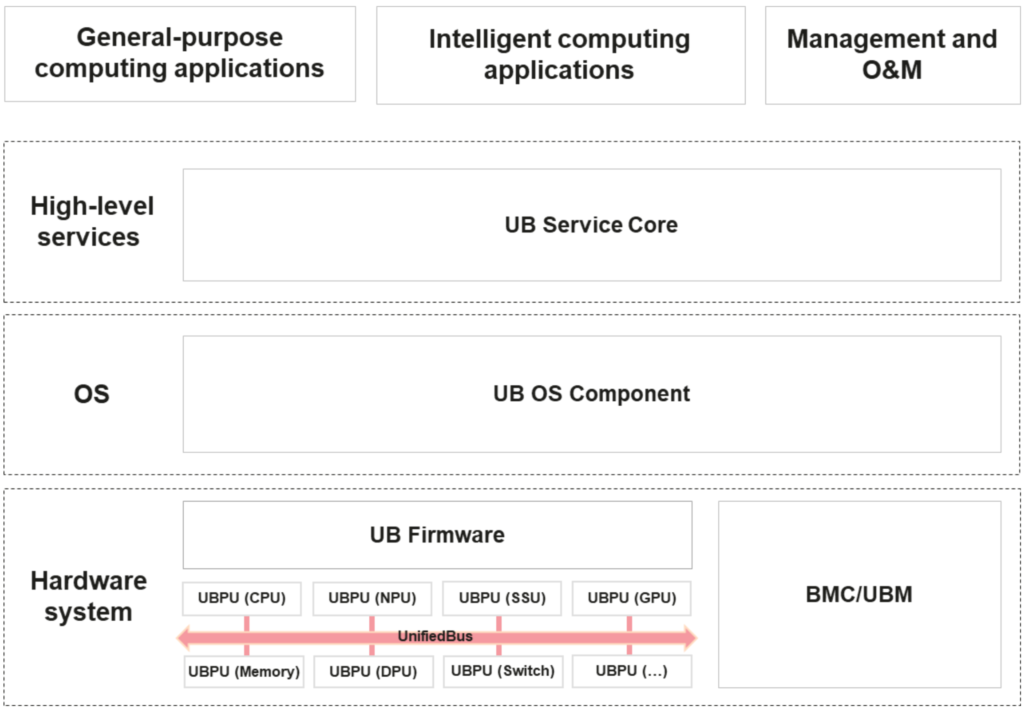
OS Implementation for SuperPoDs via UnifiedBus

UnifiedBus Architecture

SuperPoDs are the core of UnifiedBus-powered computing systems. They redefine computing systems, break the boundaries of computing hosts, and enable the expansion of intelligent and general-purpose computing capabilities, thereby achieving substantial performance gains. UnifiedBus represents the target computing system architecture for the future AI era. The following figure shows the UnifiedBus architecture.



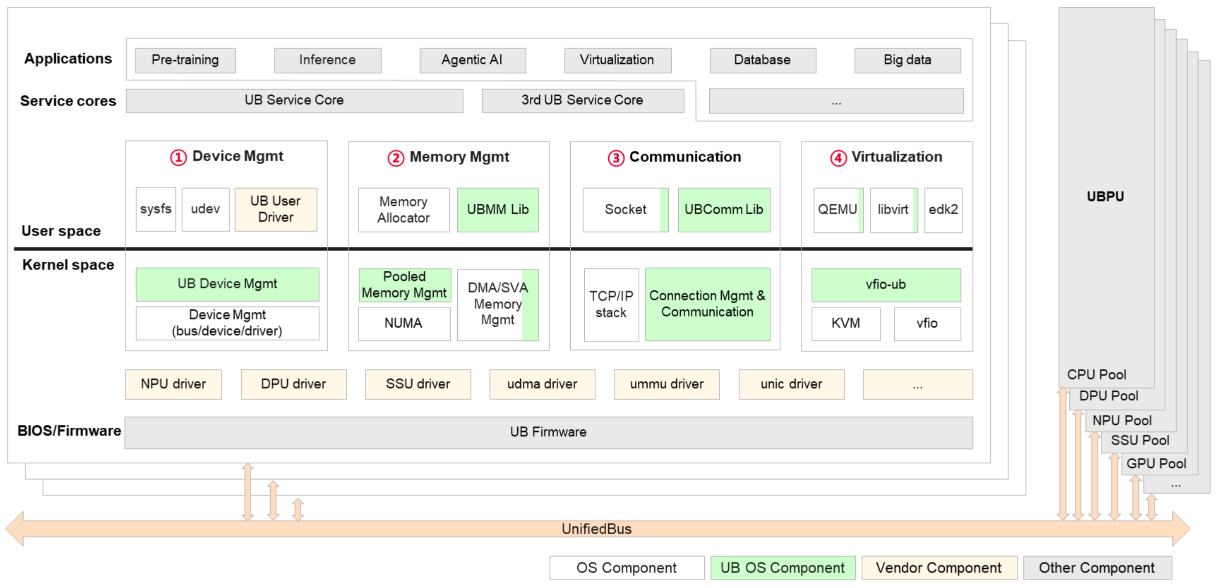
Within a SuperPoD, UnifiedBus provides an ultra-low-latency, unified multi-protocol interconnect. It fully pools and connects computing resources via peer-to-peer links, enabling flexible resource composition, ultra-large-scale networking, and high system availability. The following figure shows the overall reference architecture of a UnifiedBus-powered computing system.



This reference consists of four layers:

- **The UnifiedBus hardware implements composable computing for SuperPoDs.**
  - **UnifiedBus**: It forms the foundation of a UnifiedBus-powered SuperPoD computing system. UnifiedBus provides bus-level unified interconnect, peer-to-peer collaboration, full resource pooling, large-scale networking, and high availability for computing clusters.
  - **UB Fabric Manager (UBFM)**: It completes the UnifiedBus interconnect configuration, enables flexible and efficient aggregation of computing resources, optimizes computing interconnect SLAs (including latency, bandwidth, and reliability), and delivers flexible SuperPoD computing power.
- **UB OS Component enables UnifiedBus and devices, and provides unified abstraction and management.**
  - UB OS Component extends the Linux kernel to natively support SuperPoDs, providing unified abstraction and management of UnifiedBus devices.
  - It also maintains compatibility with POSIX interfaces to enable fast migration of existing applications.
- **UB Service Core enables peak SuperPoD performance.**
  - A UnifiedBus-powered computing system fully leverages its advantages in memory pooling, communication, and distributed operations. It simplifies UnifiedBus resource management and scheduling, and provides high-level UnifiedBus services (via UB Service Core) to support computing applications.

UB OS Component



UB OS Component enhances the original OS framework of memory management, communication, device management, and virtualization to provide improved support for UB. The feature enhancements include:

- **Device Mgmt**: It provides UB device management capabilities, allowing hot plugging and configuration of UB devices on compute nodes. It consists of the following modules:
  - **UB Device Mgmt**: Based on Linux's device management model (bus/device/driver), this module provides extended management for UB devices. It manages UB drivers, UB

firmware interactions, and UB device hot plugging. In addition, it provides device driver interfaces for developing UB device drivers.
    - **sysfs**: UB Device Mgmt generates UB device, driver, and bus information in sysfs. Users can view and use the information through sysfs.
    - **udev**: UB Device Mgmt generates uevents. Users can use the udev tool to obtain the hot plug status of UB devices and manage these devices accordingly.
    - **UB User Driver**: It is the user-space driver for UB devices. It exposes UB devices to user space, enabling applications and virtualization software to use these devices.
- **Memory Mgmt**: It enables memory-semantic access for memory borrowing and sharing across compute nodes and devices within UB Domain. It consists of the following modules:
    - **DMA/SVA Memory Mgmt**: Based on Linux's direct memory access (DMA) and shared virtual addressing (SVA) features, this module manages UB Memory Management Unit (UMMU). Drivers and other components can register host memory with UB devices using this module.
    - **Pooled Memory Mgmt**: Its functions include exporting local memory, importing remote memory, and maintaining shared memory consistency. Based on Linux's NUMA memory management framework, Pooled Memory Mgmt can bind remote memory to NUMA nodes. When it instead binds remote memory to memory devices (character devices created under **/dev**), the remote memory can be accessed through mmap calls.
    - **UBMM Lib**: It encapsulates the interfaces provided by Pooled Memory Mgmt in user space. UB Pooled Resource Manager (UBPRM, see Note 1) can call these interfaces to manage memory on both the Home and User sides.
    - **Memory Allocator**: It is compatible with common memory allocators including glibc, jemalloc, tcmalloc, and libnuma. After UBPRM borrows memory from other compute nodes and binds it to NUMA nodes through UBMM Lib, applications can allocate or release the borrowed memory using standard memory management interfaces such as malloc, free, and numa_alloc_onnode provided by these common allocators, without requiring any application modifications.
- **Communication**: It implements communication and remote procedure calls (RPCs) across compute nodes and UB devices. It consists of the following modules:
    - **Connection Mgmt & Communication**: They manage asynchronous communication connections and provide kernel-space asynchronous communication interfaces for other modules.
    - **UBComm Lib**: It provides interfaces for enabling asynchronous communication and Unified Remote Procedure Call (URPC) within UB. Applications can use these interfaces to perform inter-node communication and function invocation within a SuperPoD.
    - **Socket**: It is compatible with glibc and other libraries that provide socket interfaces. Applications on different nodes can communicate with each other through socket interfaces without modification or with minimal modification (see Note 2).
- **Virtualization**: It enables UB devices to be passed through to VMs. It consists of the following modules:
    - **QEMU**: It extends QEMU functions to virtualize UB Controller, UB devices, and UMMU.
    - **libvirt**: It extends libvirt functions to support UB device creation and deletion.
    - **vfio-ub**: It supports UB device virtualization based on Linux's Virtual Function I/O (VFIO) framework.

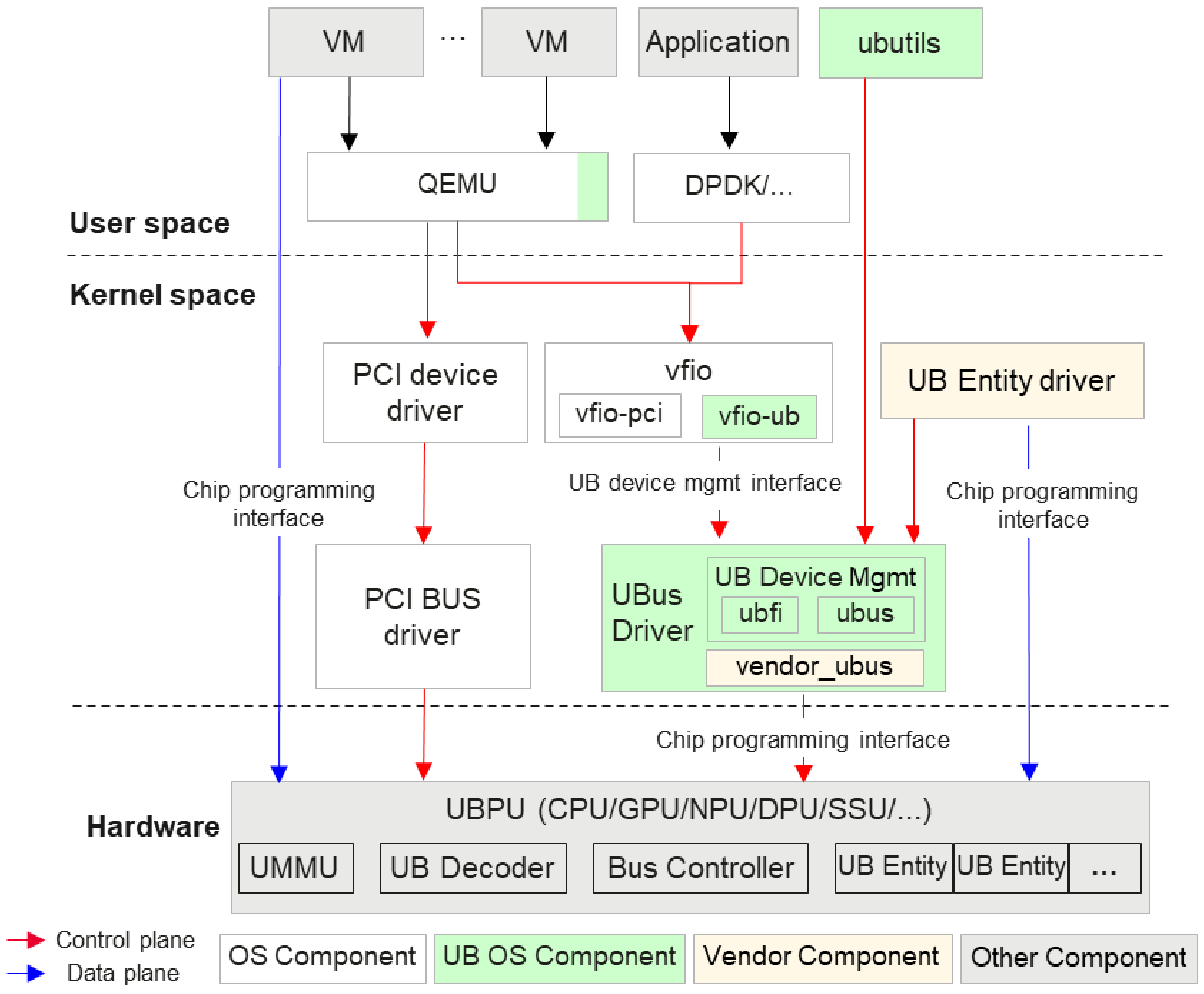
For details about the functions and code of the UnifiedBus components, see UnifiedBus™ (UB) Software Reference Design for Operating Systems.

Device Mgmt

UB device management involves UBus Driver, vfio-ub, and ubutils, which together provide a set of UB device service management interfaces. These services include device discovery, device registration, and interrupt enabling for UB device drivers; user-space passthrough to UB devices; and querying UB device information and configurations. Various UB devices can be registered with UB to perform their functions.

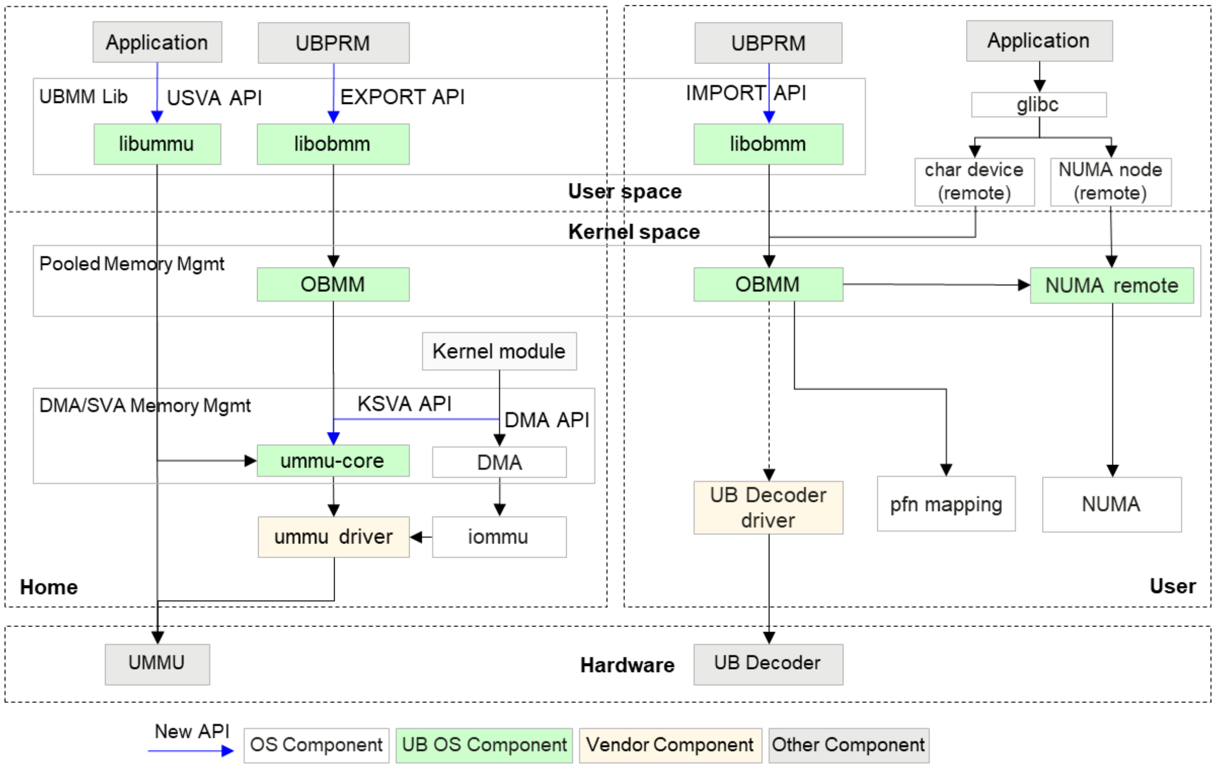
In the OS, UB device management functions, such as device discovery, registration, and driver loading, can be used in both single-server and cluster deployments.

- **Single-server scenarios**: Each server operates independently and exclusively manages its local UB devices. UBus Driver discovers and enables these devices, while vfio-ub enables user-space passthrough for direct device access.
- **Cluster scenarios**: In a SuperPoD, UBus Driver discovers and enables UB Entities allocated to compute nodes, supports the addition and removal of pooled devices, and works with vfio-ub to enable user-space passthrough for direct device access.



Memory Mgmt

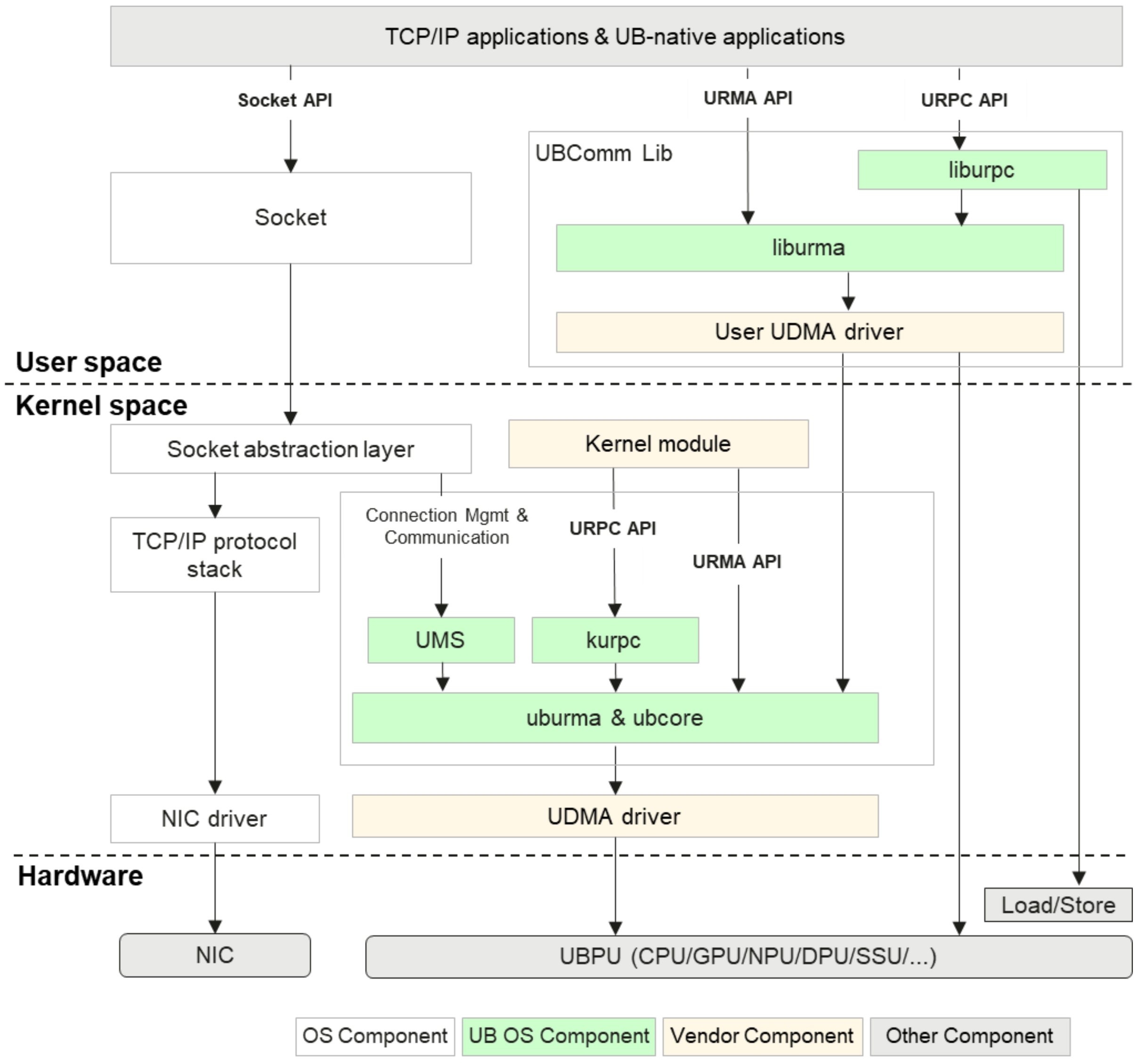
In general, existing bus architectures follow a host-device model, where the host (CPU) manages each device and three independent memory access paths exist: host-to-device, device-to-host, and device-to-device. UB streamlines this model. It categorizes UB nodes as Users and Homes in the memory access model. Users can access the memory resources of Homes either synchronously (via load/store) or asynchronously (via DMA).



Based on UnifiedBus capabilities, the Ownership-Based Memory Management (OBMM) module provides inter-node data path configuration and cross-node data consistency maintenance. After the configuration is complete, users can access the memory of another node from within a UnifiedBus cluster, enabling cross-node memory sharing, memory pooling, and inter-node memory borrowing.

Communication

UnifiedBus Memory Development Kit (UMDK) is a distributed communication software library that provides high-performance communication interfaces for inter-card communication within data center networks, SuperPoD environments, and servers, fully utilizing UB hardware capabilities.



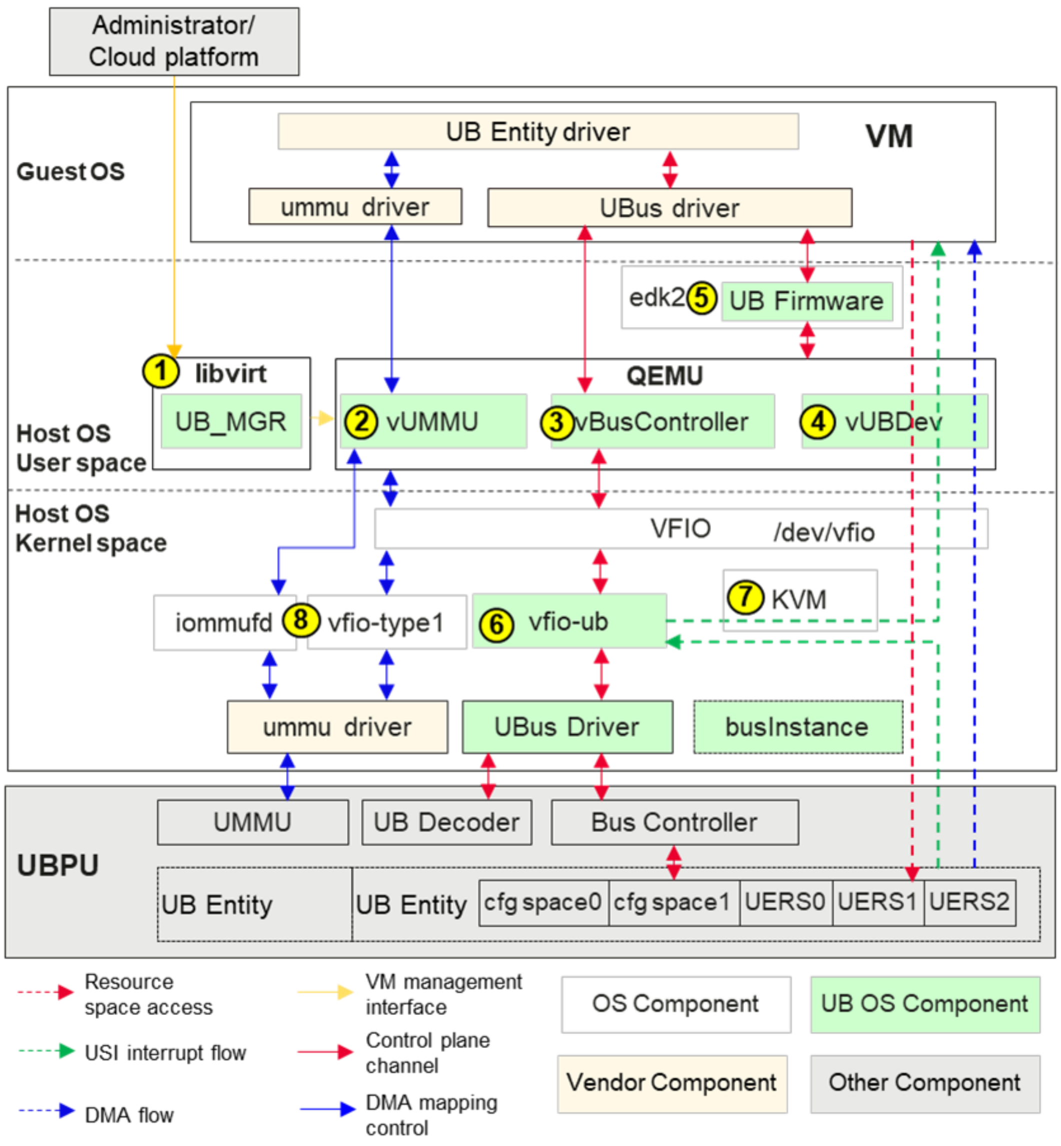
- **UB Memory-based Socket (UMS)**: It can connect to the Socket abstraction layer. It is a kernel network protocol stack compatible with the Socket programming interface in the northbound direction and transmits data over the UnifiedBus network in the southbound direction. It transparently accelerates TCP applications to improve performance.
- **Unified Remote Memory Access (URMA)**: It is a software library for communication over UnifiedBus. It masks hardware differences and provides remote memory access semantics, such as read/write, send/receive, and atomic operations. URMA serves as the foundation for UnifiedBus communication applications.
- **Unified Remote Procedure Call (URPC)**: It is the native RPC of UnifiedBus. It supports high-performance communication between hosts and devices, accelerating remote procedure calls.

Virtualization

In cloud computing scenarios, hardware resources on a physical machine are isolated and provisioned to VMs through virtualization, significantly enhancing resource utilization. As a next-generation high-speed interconnect, UB natively supports virtualization.

Mainstream device virtualization technologies include I/O full virtualization, I/O paravirtualization, and hardware-assisted I/O virtualization. Under traditional architectures, each server connects to the Top of Rack (TOR) switch through its own NIC, meaning its maximum supported bandwidth is fixed. This can lead to several issues in real-world applications, as illustrated below.

A UB Entity acts as an isolated functional unit within a UB device. Virtualization in UB supports direct device assignment to VMs, delivering native-level performance. Compared to traditional hardware-assisted I/O virtualization, UB device virtualization not only supports passthrough to local UB devices but also to pooled UB devices. The preceding problems can be solved by pooling NIC and DPU resources. All device resources are aggregated in a shared device pool, allowing servers to dynamically request NIC resources based on their actual network load. This approach improves overall resource utilization and helps prevent network bandwidth bottlenecks. In addition, UnifiedBus provides unique memory management and high-bandwidth communication features, enabling flexible memory overcommitment and fast live migration for VMs.



- **UBNative**: It is a high-performance device virtualization solution built on the UnifiedBus protocol. It simulates the UnifiedBus device model to enable seamless VM access to

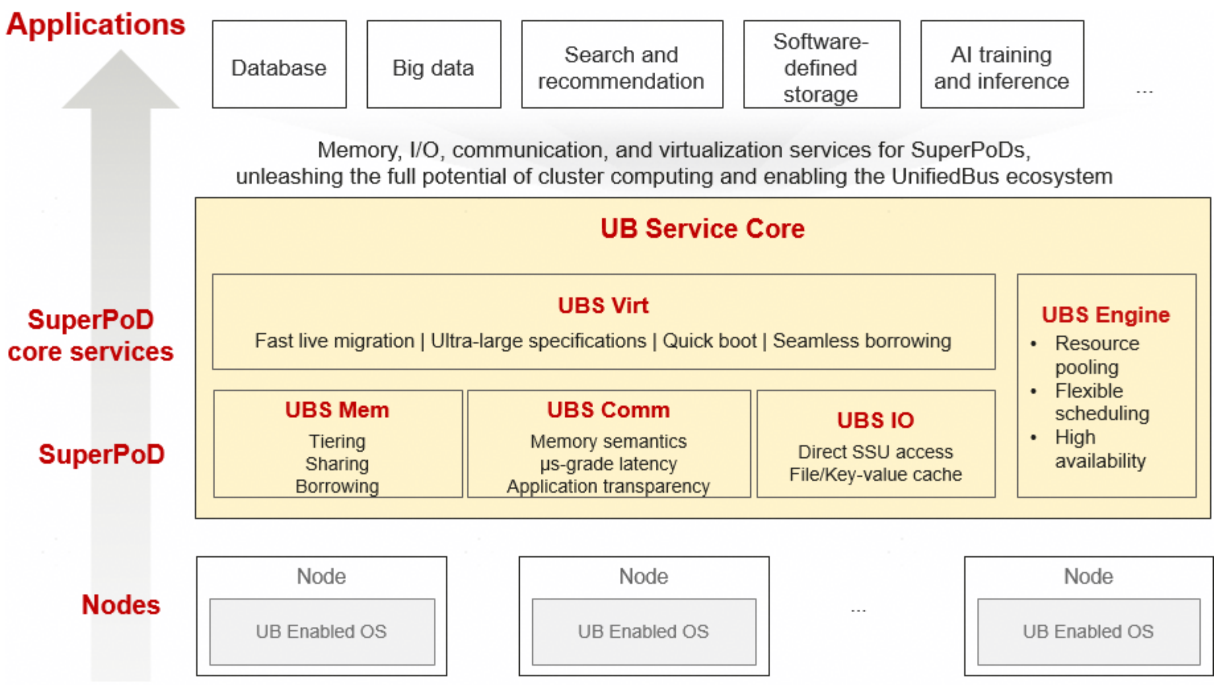
UnifiedBus devices. Based on Virtual Function I/O (VFIO) technology, UnifiedBus devices are passed through to VMs, ensuring native device performance in virtual environments. This achieves hardware-level efficiency and resource utilization, providing stable support for high-throughput, latency-sensitive scenarios.

- **Memory overcommitment**: As host memory capacity increases, memory accounts for a larger share of the total cost of ownership (TCO). Improving memory utilization in cloud scenarios is therefore becoming increasingly important. The memory overcommitment function is implemented based on the UnifiedBus-powered SuperPoD architecture. A memory reclamation solution for hugepage and passthrough VMs fully reclaims unused VM memory, improving virtual memory utilization and reducing cloud vendors' TCO.
- **Ultra-fast live migration**: URMA memory read/write semantics are used with the UnifiedBus high-performance network to migrate memory data. This reduces downtime and resource overhead, and improves the success rate of VMs under high memory pressure.

UB Service Core

For SuperPoDs, UnifiedBus builds the UB Service Core to encapsulate underlying capabilities and cluster topologies. This allows applications to use SuperPoD resources like local resources, streamlining development and maintaining compatibility with the existing ecosystem.

UB Service Core enables five cluster-level system services, leveraging the advantages of the peer-to-peer interconnect and accelerating applications by 30% to 50%. The following figure illustrates the reference architecture of UB Service Core.



UB Service Core consists of five components:

- **UB Service Core Engine (UBS Engine)**: It provides resource pooling and dynamic scheduling for memory and data processing units (DPUs), supports distributed automatic primary node election, and implements N–1 high availability. It serves as the core control-plane reference implementation for a UnifiedBus-powered computing system.
- **UB Service Core Memory (UBS Mem)**: **It s**upports unified memory programming and enables memory sharing and pooling within a UnifiedBus-powered SuperPoD.

- **UB Service Core Communication (UBS Comm)**: It is a high-performance, highly reliable communication protocol with ecosystem compatibility (user-space Socket/Verbs over UnifiedBus) for SuperPoDs.
- **UB Service Core IO (UBS IO)**: It provides high-level I/O services for the global data read/write cache system for SuperPoDs.
- **UB Service Core Virt (UBS Virt)**: It supports virtualization pooling, live migration policy decision-making, quick disaster recovery, and efficient VM/container communication, improving virtualization performance.

For details, see UnifiedBus™ (UB) Service Core Software Architecture Reference Design.

Note: openEuler 24.03 LTS SP3 currently includes UBS Engine and UBS Comm. Additional components are expected to be open sourced in 2026.

Application Scenarios

SuperPoDs, built on UnifiedBus, support 10 core service scenarios for AI and general-purpose computing. These include foundation model pre-training, central inference, post-training and reinforcement learning, multi-modal content understanding and generation, agentic AI, virtualization, big data, databases, software-defined storage, and HPC. It provides leading system capabilities, improving computing service performance and resource utilization.

For details, see UnifiedBus™ (UB) SuperPoD Reference Architecture White Paper.

5 Kernel Innovations

openEuler 24.03 LTS SP3 runs on Linux kernel 6.6 and inherits the competitive advantages of community versions and innovative features released in the openEuler community.

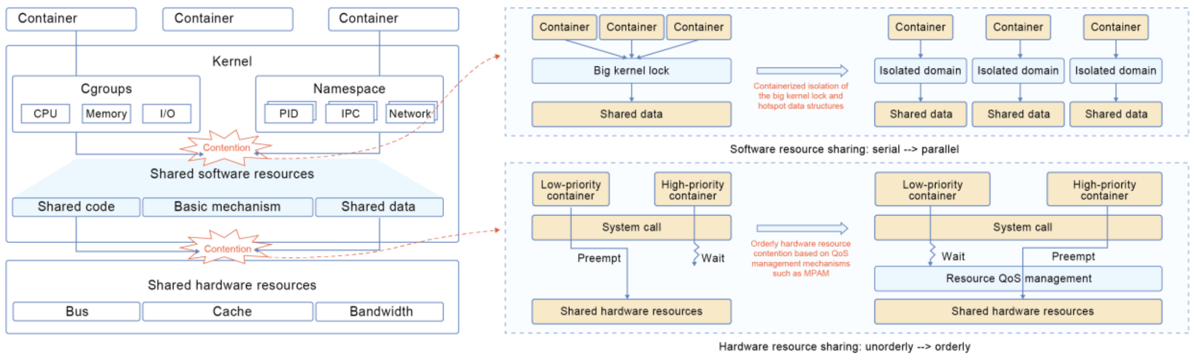
- **Programmable page cache for file systems**: A programmable page cache framework is implemented to address inefficient I/O during model loading in model inference scenarios. The framework is stacked on an existing file system in a transparent manner and forwards file system page faults to user space. This allows applications to implement custom page cache based on load characteristics in user space, significantly improving I/O efficiency for loading different models.
- **Cross-process zero-copy data transfer**: It provides efficient inter-process data transfer within a node. Applications can map pages from a specified virtual memory space in the source process to the virtual memory space of the destination process. It supports both page middle directory (PMD) hugepage mapping and page table entry (PTE) smallpage mapping. The mapped destination addresses are accessible with the same permissions as their corresponding source addresses.
- **Filesystem in Userspace (FUSE) over io_uring**: In the current FUSE architecture, communication between the user-space daemon and the kernel-space driver module via the character device **/dev/fuse** suffers from performance bottlenecks: (1) In multi-threaded environments, FUSE requests must be obtained through **/dev/fuse**, leading to lock contention. (2) I/O requests are transferred one by one, without support for batch processing, resulting in low efficiency for I/O-intensive tasks. (3) The user program initiating read/write requests and the backend FUSE thread handling file I/O often run on different CPU cores, causing frequent core switches and reduced efficiency. To address these bottlenecks, the FUSE over io_uring technology is employed, with the following solutions: (1) Replace **/dev/fuse** with the io_uring communication interface to improve concurrency. (2) Allow the backend FUSE service to flexibly use either io_uring or the traditional **/dev/fuse** character device for communication.

- **I/O QoS controller IOInflight for I/O hybrid deployments**: For details, see High-Density Many-Core Container Isolation.
- **epoll_wait asynchronous prefetch based on Xcall**: In performance optimization scenarios targeting specific system calls, dynamic Xcall leverages an interception mechanism to execute customized system calls. This allows users to intercept system calls at the granularity of application ELF files, without intrusive kernel modifications. Based on this mechanism, a kernel module for epoll-based asynchronous prefetch is implemented. It can achieve notable performance benefits in scenarios such as Redis.

6 Cloud Base

High-Density Many-Core Container Isolation

Server chips have evolved from multi-core to many-core architectures (typically exceeding 256 cores), posing new challenges to OSs. To boost rack-level computing density and reduce data center TCO, many-core servers have become the mainstream in the Internet industry. As cloud technologies and service scales advance, containerized deployment has also become the dominant model. Against this backdrop, serialization and synchronization overheads hinder system scalability, while interference and low resource utilization become increasingly prominent. These scalability issues in container deployments arise from contention for shared hardware and software resources.



Feature Description

Lightweight virtualization is used to partition resources by NUMA domain and enforce container-level resource isolation within each domain. This approach minimizes performance interference caused by hardware and software resource contention and enhances the container deployment scalability.

- **VM memory QoS control**: When multiple tenants' VMs are deployed on the same physical host, memory-intensive VMs may consume a large portion of the available memory bandwidth. This can lead to resource contention, preventing other VMs from obtaining sufficient bandwidth to meet their performance requirements. As a result, the overall system QoS may be degraded. Leveraging the MPAM feature of the Kunpeng processor and the OS-level resource control mechanisms, the system can perform fine-grained monitoring and dynamic control of memory bandwidth usage for up to 30 VMs. This capability enables configuration of upper and lower limits, as well as priority policies, to establish a memory bandwidth isolation and assurance framework in a multi-tenant environment.
  - **Memory bandwidth upper limit**: A maximum memory bandwidth threshold can be configured for each VM to prevent any single VM from consuming excessive bandwidth resources, thereby avoiding performance interference with other tenant VMs.

  - **Memory bandwidth lower limit**: A minimum bandwidth guarantee can be configured to maintain a baseline allocation even for lightly loaded VMs, enabling dynamic resource optimization and efficient bandwidth utilization.
  - **Priority-based scheduling policy**: The memory bandwidth priority of each VM can be configured based on service importance, ensuring stable bandwidth allocation for critical workloads and improving the availability and service quality of high-priority VMs.
- **NUMA affinity of virtual devices**: PCI devices possess NUMA affinity and can be directly accessed on the host. The OS scheduling mechanism optimizes task placement based on device affinity to minimize performance loss resulting from cross-NUMA access to PCI devices. In VM passthrough scenarios, PCI devices do not expose their NUMA node affinity to guest VMs. This feature expands the PCI device topology of VMs based on the PCI Expander Bridge (PXB). Within a VM, it displays the NUMA node where each virtual device is located. This helps the system optimize scheduling and allows users to deploy service applications according to the NUMA nodes of their virtual devices, thereby reducing performance loss caused by cross-NUMA resource access and improving the overall performance of VM service applications.
- **Lightweight virtualization**: Virtual device interrupt offload allows VirtIO device interrupts to be offloaded to hardware injection, reducing storage virtualization overhead. PMD queue load balancing distributes virtual drive I/O queues from a single reactor to multiple reactors, eliminating CPU bottlenecks.
- **CPU scheduling by domain**: CPUs are divided into domains by cluster for container deployment. Each container operates within an independent scheduling domain. This design isolates interference between containers, reduces cross-cluster cache synchronizations, and eases contention for hardware resources like cache and NUMA memory. It improves performance by more than 10% in high-concurrency Redis scenarios.
- **Interference isolation in file system block allocation**: Optimizations to the group lock and s_md_lock in the EXT4 block allocation and release processes enhance the scalability of EXT4 block allocation. When the target block group is occupied, allocation can switch to an idle block group to reduce CPU overhead caused by multiple containers competing for the same group. Leveraging multiple EXT4 block groups helps ease group lock contention. Apart from that, the global target of the streaming allocation is split to multiple targets, so that the contention for the global lock s_md_lock is alleviated and the file data is more aggregated. In a 64-container concurrency scenario, the OPS increases by over 5 times in mixed block allocation and release workloads and by over 10 times in single-block allocation workloads.
- **Efficient slab reclamation**: In this openEuler release, the read and write locks used for slab memory reclamation are replaced with the read-copy-update (RCU) lock-free mechanism. Memory reclamation across different slabs operates independently, significantly improving reclamation efficiency. In multi-container concurrency scenarios, system calls are accelerated.
- **TCP hash interference isolation**: In high-concurrency scenarios, lock contention in tcp_hashinfo bash and ehash and frequent ehash calculations lead to reduced bandwidth and increased latency. The spin lock of tcp_hashinfo bash and ehash can be replaced with read-copy update (RCU), and the ehash calculation method can be changed to lport increment. These changes reduce the query time and calculations and reduce the lock contention in the TCP connection hash.
- **Enhanced control group (cgroup) isolation**: Original atomic operations are replaced with percpu counters to avoid parent node contention across namespaces and eliminate rlimit count interference between containers. This mechanism addresses the linearity issue in the **will-it-scale/signal1** test case and triples concurrent throughput performance in a 64-container deployment. Memory cgroups are released in batches to avoid contention for the same parent node's counter caused by frequent small memory releases, enhancing memory count scalability. In the **tlb-flush2** test case, this improves throughput by 1.5 times with 64

containers. Leveraging eBPF's programmable kernel capabilities, a host information isolation and filtering approach is provided to present container-specific resource views. Compared with the peer LXCFS solution, this openEuler solution avoids the overhead of switching between the kernel space and user space, and eliminates the performance and reliability bottlenecks associated with the LXCFS process. It doubles the resource view throughput in a single container and achieves a 10-fold increase in a 64-container deployment.

- **Interference monitoring**: Interference falls into three categories by result: instruction execution failure, instruction slowdown, and increased instruction execution. Interference is monitored from the kernel perspective, with statistics collected on each typical category during runtime. The system supports online monitoring of schedule latency, throttling, softirq, hardirq, spin lock, mutual exclusion (mutex), and simultaneous multi-threading (SMT) interference while incurring less than 5% performance overhead.
- **Kunpeng memory and cache QoS control**: The memory bandwidth traffic and cache usage at each level can be configured based on the upper limit, lower limit, or priority-based policy. Each thread is assigned a specific isolation policy to suit specific service requirements. The usage of shared resources is monitored in real time at both service and thread levels, and reported to the control policy to enable feedback-driven control. In addition, the MPAM feature and the SMMU combine to enhance peripherals' I/O QoS. They support bandwidth isolation for peripherals and heterogeneous accelerators and allow for resource monitoring at the device level.
- **Dynamic QoS policy configuration**: This openEuler release provides a cluster-level MPAM QoS management plugin. Using the QoS interfaces provided by MPAM, the plugin automatically assigns priorities to nodes based on user-defined policies and sets MPAM QoS priorities for offline tasks according to user configurations. This ensures efficient resource utilization in hybrid deployment scenarios. When online services are busy, the last-level cache (LLC) and memory bandwidth allocated to offline tasks are automatically preempted. When online services are idle, these resources are released to enhance offline service performance.
- **I/O QoS controller IOInflight for I/O hybrid deployments**: In high-density hybrid deployments, I/O resource contention can cause the interference rate of online services to exceed 40%. Traditional blk-throttle static I/O throttling reduces interference but wastes drive bandwidth. To address this issue, the IOInflight controller is introduced. It employs a dynamic I/O control mechanism based on latency monitoring and queue depth to precisely preempt online services. The controller keeps the interference rate and memory overhead below 5%, ensuring low latency for online services, while providing offline workloads with up to twice the bandwidth compared to traditional hard-limit approaches.

Application Scenarios

In a many-core server environment, service containers are deployed in a high-density configuration. The openEuler solution minimizes interference between containers, increases deployment density, and enhances resource utilization.

In memory-intensive scenarios, this feature mitigates bandwidth contention among multiple VMs. It monitors and controls the memory bandwidth usage of each VM, ensuring that performance-critical services run reliably.

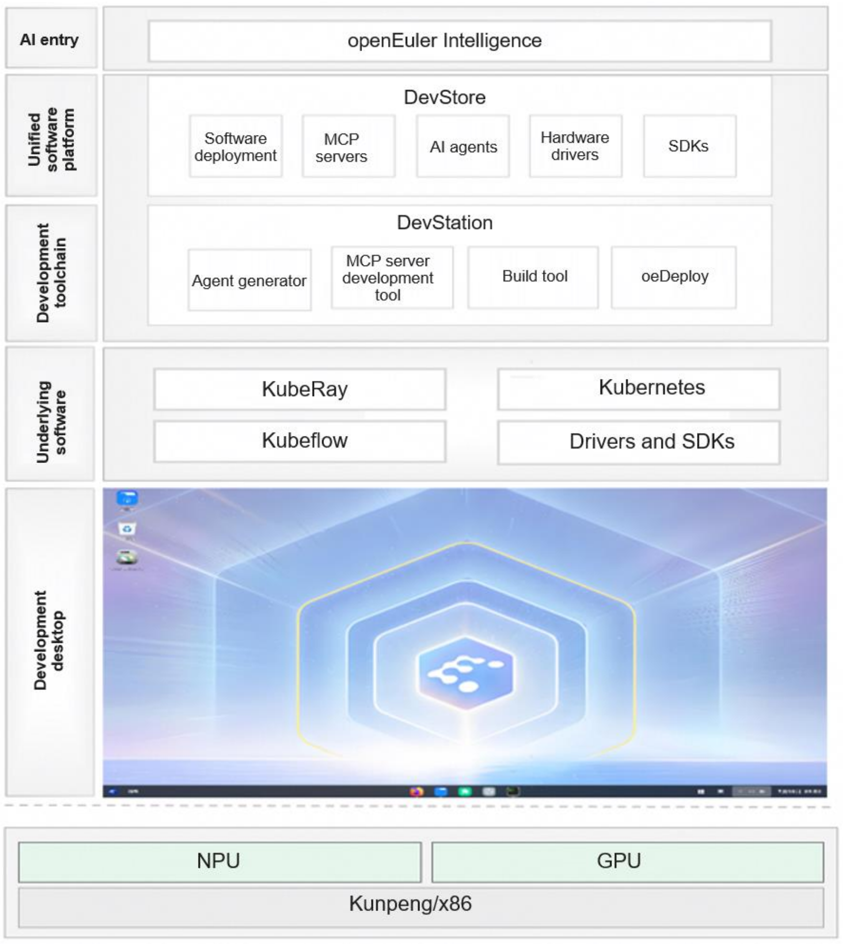
7 Developer Ecosystem

DevStation

DevStation is an intelligent developer workstation built on openEuler, designed for geeks and innovators. It provides an out-of-the-box, efficient, and secure development environment that streamlines the entire workflow from deployment and coding to compilation, building, and releasing. By integrating a one-click runtime environment with a full-stack development toolchain, it enables a seamless transition from system boot to code execution. The new MCP AI engine allows for quick invocation of community toolchains, offering a significant leap in efficiency from infrastructure setup to application development, all without complex installation.

Feature Description

- **Developer-friendly integrated environment**: Pre-installed with a wide range of development tools and IDEs like VSCodium, this distribution supports multiple programming languages to meet the needs of front-end, back-end, and full-stack developers.
- **Intelligent CVE fixing and security O&M**: A CVE fixing system is integrated. It uses LLM analysis, code similarity algorithm (Levenshtein distance), and automated backporting to intelligently identify and resolve code conflicts during patch application. While retaining the core fixing logic, the average time required for manual conflict resolution is reduced from 4 hours to about 10 minutes, greatly improving the response and fixing efficiency for security vulnerabilities.
- **Native community tool ecosystem**: New tools like oeDeploy (a one-click deployment tool), epkg (an extended package manager), DevKit (a development toolchain), and openEuler Intelligence (DevStation intelligent assistant) provide full-lifecycle support from environment configuration to code deployment. oeDevPlugin and oeGitExt are VSCodium plugins designed for the openEuler community. They provide visual management for issues and pull requests (PRs), facilitating code repository cloning, PR submission, and real-time task status synchronization. openEuler Intelligence supports natural language for generating code snippets, creating API references, and explaining Linux commands.



- **GUI-based programming environment**: DevStation integrates graphical programming tools to streamline coding for beginners while offering powerful visual programming capabilities for veterans. It also comes pre-installed with productivity tools like Thunderbird.
- **MCP-based intelligent application ecosystem**: DevStation deeply integrates the MCP framework to build a complete intelligent toolchain ecosystem. It includes pre-installed MCP servers like oeGitExt and rpm-builder, which provide capabilities for community task management and RPM packaging. It intelligently wraps conventional development tools like Git and RPM builders using the MCP protocol, offering a natural language interaction interface. It also provides an MCP conversion and quality assurance toolchain that can automatically generate high-quality and testable MCP servers based on existing tools and test suite (mugen) in the community. This toolchain works with the mcp-testkit test framework to ensure service reliability.
- **Enhanced system deployment and compatibility**: DevStation offers extensive hardware support, especially seamless compatibility with mainstream laptop and PC hardware (such as touchpads, Wi-Fi, and Bluetooth), and a restructured kernel build script (**kernel-extra-modules**) to ensure bare metal deployment experience. It also supports flexible deployment options, including Live CD (instant run without installation), bare metal installation, and VM deployment.
- **New installation tool**: heolleo is a modern client tool designed specifically to simplify the DevStation installation process. Built with a modular design, it easily supports feature expansion across various hardware architectures (like x86 and Arm), file systems, and bootloaders (like GRUB). It offers flexible installation modes, supporting system file acquisition from both local ISO images and network sources (HTTP/FTP).
  - **Local ISO installation**: heolleo provides a local ISO installation mode for users demanding extreme stability, high speed, or deployment in offline or restricted environments. By leveraging existing system image files, it delivers fast, reliable, and completely offline installation experience with automated partition setup.
  - **Network installation**: heolleo's network installation mode aligns with modern system deployment trends. It eliminates the need for manual image downloads by allowing users to obtain the latest system files directly from Internet servers, which is the most convenient access to the newest DevStation version.

Application Scenarios

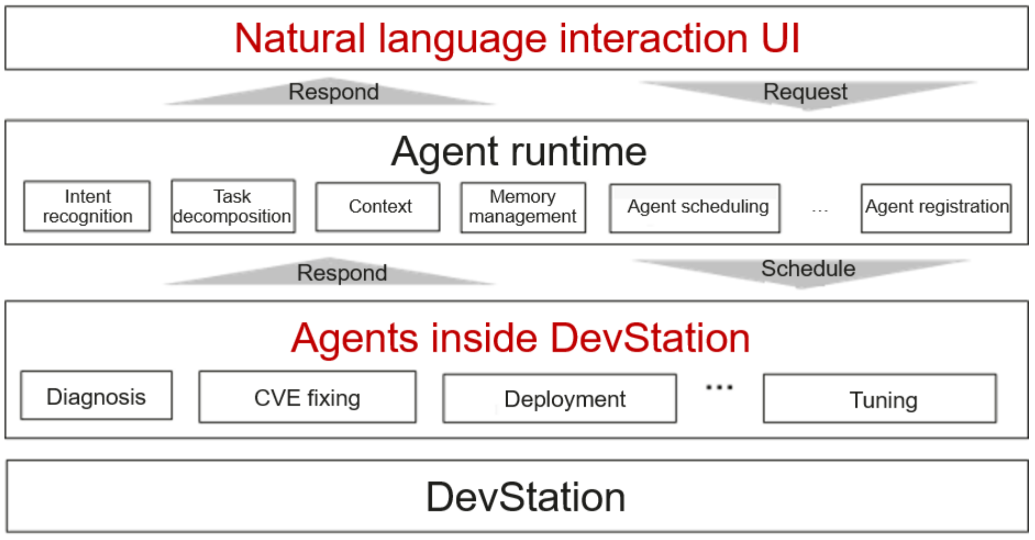
- **OS security and CVE fixing**: DevStation provides intelligent CVE fixing for security teams and kernel maintenance personnel, automatically handles patch conflicts, and reduces the single-patch processing time to less than 10 minutes. It quickly responds to and fixes vulnerabilities to ensure system security.
- **Multi-language development**: DevStation is ideal for developers working on multi-language projects, such as Python, JavaScript, Java, and C++. With various pre-installed compilers, interpreters, and build tools, it eliminates the need for manual configuration.
- **Quick installation and deployment**: DevStation integrates oeDeploy to deploy distributed software such as Kubeflow and Kubernetes within minutes. oeDeploy provides a unified plugin framework and atomic deployment capabilities, allowing developers to quickly release custom installation and deployment plugins.
- **Hardware compatibility and bare metal testing**: For testers and developers focusing on southbound compatibility, DevStation ensures robust hardware support for mainstream laptops and servers, and allows bare metal deployment for driver compatibility testing.
- **Improved developer efficiency**: The MCP RPM-builder toolchain improves MCP usability by supporting automated packaging and releasing of RPM packages to the community, ensuring 100% availability for instant MCP installation. It helps build a complete MCP intelligent application ecosystem repository that covers scenarios like deployment, testing,

and performance tuning, allowing developers to query assigned community issues, create PRs to submit code changes, and automate build and verification via CI/CD.

openEuler Intelligence

Feature Description

openEuler Intelligence is a built-in desktop AI assistant of DevStation. It provides a dialogue window to actively identify user intent, enabling interaction with the OS through natural language, greatly facilitating development and O&M.



- **Interaction UI**: openEuler Intelligence provides a natural language interaction UI, through which user requests are transmitted to the agent runtime for processing.
- **Agent runtime**: As the brain of the AI assistant, the agent runtime understands user intent, decomposes it into sub-tasks, and delegates the sub-tasks to the agents and MCP servers integrated in DevStation. The sub-tasks are scheduled cyclically until all of them are complete.
- **Agents inside DevStation**: Leveraging the extensive MCP server and agent repositories of openEuler, DevStation is evolving into an agentic OS composed of agents. In addition to built-in agents, openEuler Intelligence supports the installation of relevant MCP servers and agents via DevStore.

Application Scenarios

openEuler Intelligence centers on natural language interaction and is deeply integrated into the E2E developer workflow. Leveraging precise intent recognition, it transforms complex system operations and technical tasks into simple dialogue prompts. This comprehensive approach lowers the barriers to development, O&M, and system usage.

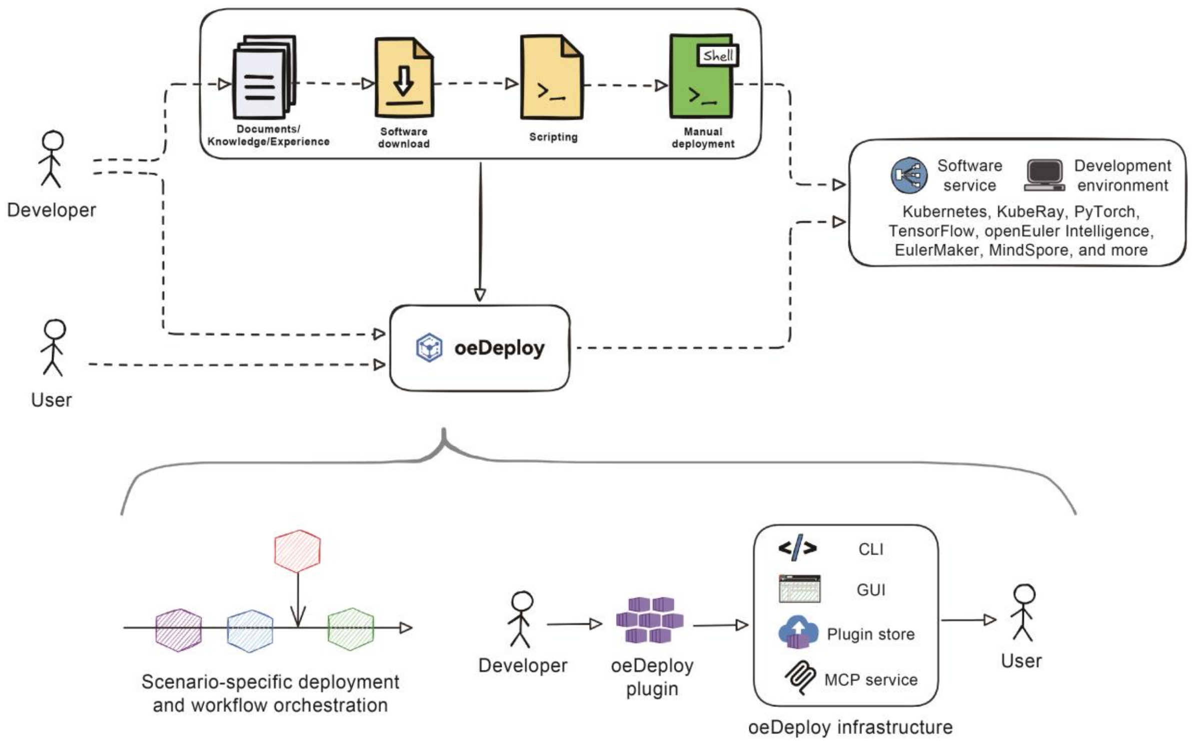
Developers can perform environment configuration by entering task requirements into the dialogue window, eliminating the need to memorize complex commands. openEuler Intelligence identifies user needs and automatically completes tasks such as environment configuration, service deployment, code development, testing, and building. It delivers OS capabilities as direct conversational services to assist novice developers with rapid onboarding, help senior developers enhance productivity, and enable O&M personnel to simplify troubleshooting. Such natural and intelligent interaction allows developers to focus on core service R&D without being burdened by underlying technical details.

oeDeploy

oeDeploy is a lightweight software deployment tool that accelerates environment setup across single-node and distributed systems with unmatched efficiency.

Feature Description

- **Multi-scenario support and quick software deployment**: oeDeploy facilitates quick deployment for both single-node applications and clustered software environments. It includes quick deployment capabilities for openEuler community toolchains, popular cloud-native software, AI development components, and RAG tools. It also extends support for the CANN development kit and Ascend NPU driver.
- **Flexible plugin management and excellent deployment experience**: oeDeploy provides an extensible plugin architecture for flexible management of diverse deployment capabilities, empowering developers to quickly release custom deployment plugins. It now optimizes plugin source management and supports one-click generation of local plugin sources. Available on DevStore, oeDeploy features both a streamlined CLI and visual operations, promising efficient software deployment experience with less code.
- **Efficient deployment and intelligent development**: oeDeploy introduces MCP servers, offering an out-of-the-box experience within DevStation. It leverages LLM inference to deploy software with natural language, boosting deployment efficiency by 2x. It can also convert user documents into executable oeDeploy plugins, increasing development efficiency by 5x.



Application Scenarios

ISVs and development teams can adopt oeDeploy as a standardized solution for software product delivery. Its CLI tools and plugin framework minimize development workload while ensuring smooth delivery, reducing user onboarding efforts and enhancing satisfaction.

For developers and maintenance personnel, oeDeploy enables instant setup of complex environments, deploying mainstream AI training and inference frameworks in just minutes. This significantly streamlines software development and eliminates tedious configuration work. Developers can also extend oeDeploy by creating and sharing custom deployment templates, democratizing quick deployment for broader user communities. By leveraging foundation models and MCP capabilities, oeDeploy makes deployment more efficient and development more intelligent.

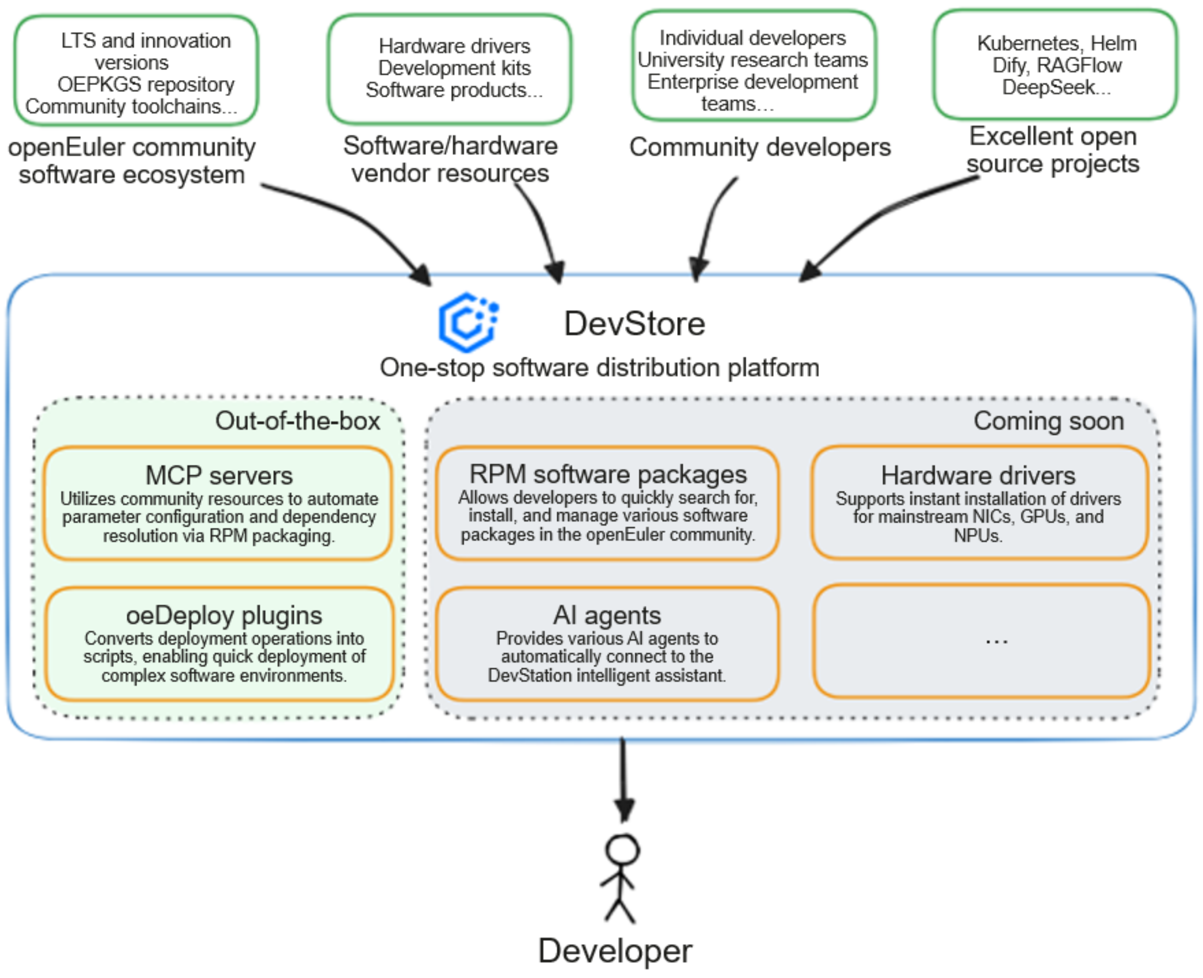
DevStore

DevStore is the application store for the openEuler desktop version, acting as a developer-centric software distribution platform. It supports the search and rapid deployment of MCP servers and oeDeploy plugins. DevStore is provided out-of-the-box on the DevStation platform.

Feature Description

- **Rapid installation of MCP servers**: Leveraging openEuler community's extensive software ecosystem, DevStore packages the software dependencies required for MCP server operations as standard RPM files. Using built-in service management tools, DevStore quickly deploys MCP servers in agent applications. It automatically solves software dependency and MCP configuration issues for users, greatly improving user experience. Currently, DevStore supports the deployment of over 80 MCP servers.
- **Quick plugin deployment**: DevStore utilizes the oeDeploy tool to enable the rapid deployment of mainstream software, substantially reducing the setup time. The supported software categories include AI software (like Kubernetes, KubeRay, PyTorch, TensorFlow, and DeepSeek), toolchains (like EulerMaker and openEuler Intelligence), and RAG tools (like RAGFlow, Dify, and AnythingLLM).

DevStore will continue to incorporate various resources from upstream communities and hardware/software vendors. It will progressively launch management functions for more software types, including RPM packages, agents, and drivers, while supporting search and one-click installation/deployment. This initiative aims to further reduce software acquisition costs.


LTS and innovation versions
OEPKGS repository
Community toolchains...
openEuler community software ecosystem
Hardware drivers
Development kits
Software products...
Software/hardware vendor resources
Individual developers
University research teams
Enterprise development teams...
Community developers
Kubernetes, Helm
Dify, RAGFlow
DeepSeek...
Excellent open source projects
DevStore
One-stop software distribution platform
Out-of-the-box
MCP servers
Utilizes community resources to automate parameter configuration and dependency resolution via RPM packaging.
oeDeploy plugins
Converts deployment operations into scripts, enabling quick deployment of complex software environments.
Coming soon
RPM software packages
Allows developers to quickly search for, install, and manage various software packages in the openEuler community.
Hardware drivers
Supports instant installation of drivers for mainstream NICs, GPUs, and NPUs.
AI agents
Provides various AI agents to automatically connect to the DevStation intelligent assistant.
...
Developer

Application Scenarios

As the application store for the openEuler desktop, DevStore enables developers, including beginners, to swiftly obtain mainstream development tools, AI software, and MCP servers. Comprehensive user manuals and operation entries are available on the details page. Notably, all complex deployment operations are consolidated into a unified operation interface, significantly reducing the learning cost and improving user experience.

8 Enhanced Features

LLVM for openEuler Compiler

LLVM for openEuler is a high-performance compiler based on open source LLVM software. It is crafted for compute-intensive scenarios such as server and Internet industries, cutting-edge data center applications, AI computing systems, and video encoding and decoding. In addition, it establishes an LLVM baseline within the openEuler community, providing a stable downstream LLVM distribution that is secure, reliable, and easily innovated upon. It currently supports mainstream languages (C and C++) and chip architectures (x86, AArch64, RISC-V, and LoongArch).

Feature Description

LLVM for openEuler introduces the following compilation features in openEuler 24.03 LTS SP3, optimizing the running efficiency of database and compression/decompression applications to unlock the ultimate software performance.

- **AArch64 endianness conversion acceleration**: In common database applications such as MySQL, data is stored in big-endian format. When loaded into memory for processing, it must first be converted to little-endian format. This involves iterative operations consisting of byte-by-byte loading, shifting, and bitwise OR, which incurs significant performance overhead. This optimization enables the AArch64 REV series instructions and increases the load width to reduce the frequency of loading, shifting, and bitwise OR, thereby enhancing application performance.
- **Machine sinking enhancement**: During compiler optimization, iteration operations on pointer-type loop variables (such as **ptr++** within a loop) are typically sunk to the end of the loop. If a load operation at the beginning of the subsequent iteration depends on the updated value of that pointer, a data dependency occurs, leading to an instruction stall. This optimization mitigates such stalls by scheduling instruction positions to resolve the dependency bottleneck.
- **AArch64 loop idiom optimization**: This optimization is used to identify common loop patterns in diverse applications. By leveraging flexible AArch64 SVE instruction capabilities, these patterns are transformed into optimized loop implementations, significantly enhancing loop execution efficiency.

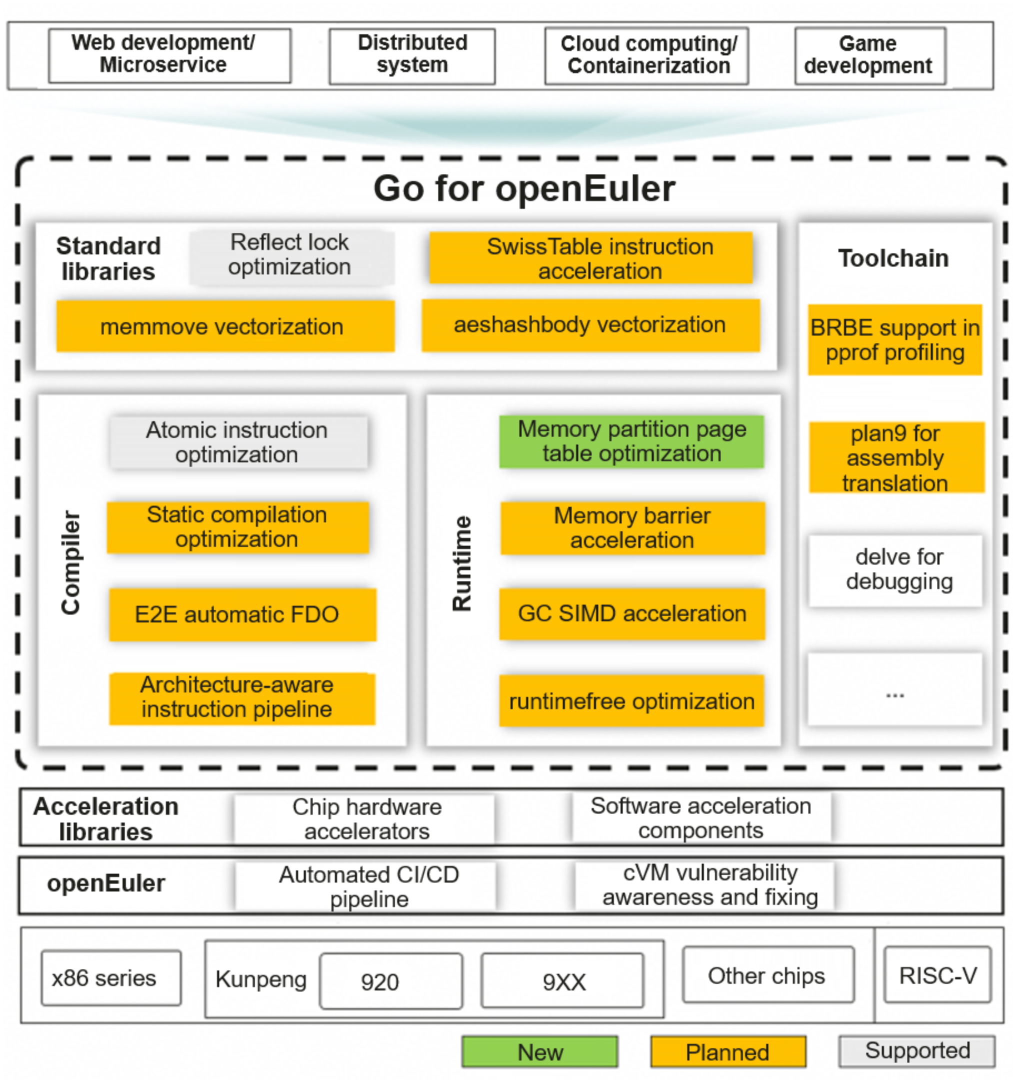
Application Scenarios

The compilation optimizations of LLVM for openEuler deliver a significant performance leap for mainstream database and compression/decompression applications such as MySQL and LZ4.

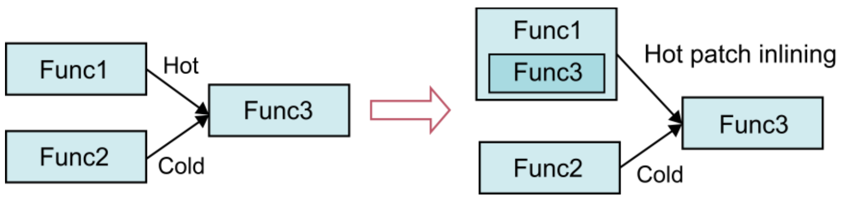
Go for openEuler Compiler

Go for openEuler is a high-performance Go distribution designed for reliability and developer friendliness. It features an optimized compiler that offers wide ecosystem compatibility, openEuler affinity, and superior user experience. Primarily targeted at container cloud scenarios that require agile development and high performance, such as cloud native and microservice applications, Go for openEuler addresses native Go performance limitations to enhance efficiency for mainstream Go workloads. The compiler adapts to hardware platforms such as LoongArch and Kunpeng to unleash their computing power.

Feature Description


Web development/ Microservice
Distributed system
Cloud computing/ Containerization
Game development
Go for openEuler
Standard libraries
Reflect lock optimization
SwissTable instruction acceleration
memmove vectorization
aeshashbody vectorization
Toolchain
BRBE support in pprof profiling
plan9 for assembly translation
delve for debugging
...
Compiler
Atomic instruction optimization
Static compilation optimization
E2E automatic FDO
Architecture-aware instruction pipeline
Runtime
Memory partition page table optimization
Memory barrier acceleration
GC SIMD acceleration
runtimefree optimization
Acceleration libraries
Chip hardware accelerators
Software acceleration components
openEuler
Automated CI/CD pipeline
cVM vulnerability awareness and fixing
x86 series
Kunpeng
920
9XX
Other chips
RISC-V
New
Planned
Supported

- Continuous feature guided optimization (CFGO)
  While ensuring the functional integrity of the program, this technique collects the program's runtime profile to make more accurate optimization decisions, resulting in a refined program with better performance. Based on the principle of program locality, it arranges hot instructions closely to improve cache/translation lookaside buffer (TLB) hits, effectively reducing front-end bottlenecks.



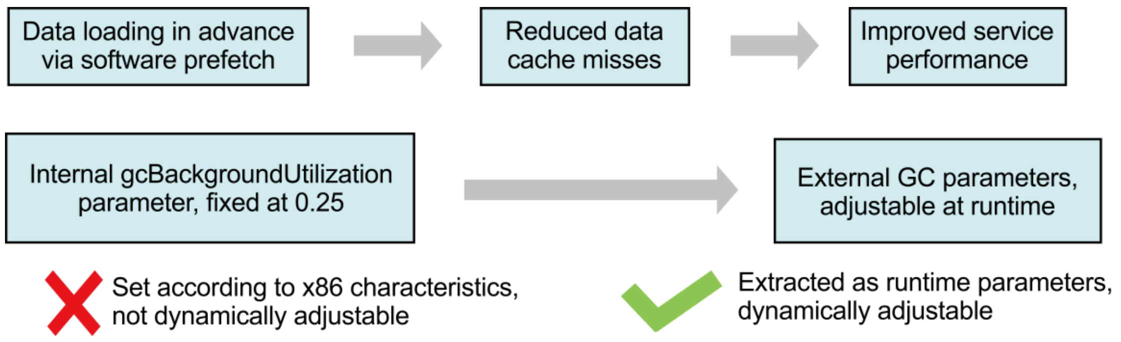
Function inlining

- Arm atomic instruction optimization

In certain service scenarios, the Go runtime incurs significant overhead when invoking compare-and-swap (CAS) operations and load/store instructions. Adopting an Arm-affinity instruction sequence delivers notable performance gains.

- Runtime garbage collection (GC) optimization

  This optimization involves the insertion of software prefetch instructions based on identified program characteristics. Parameters governing the GC coroutine overhead are extracted as runtime parameters supporting dynamic adjustment according to varying service characteristics.

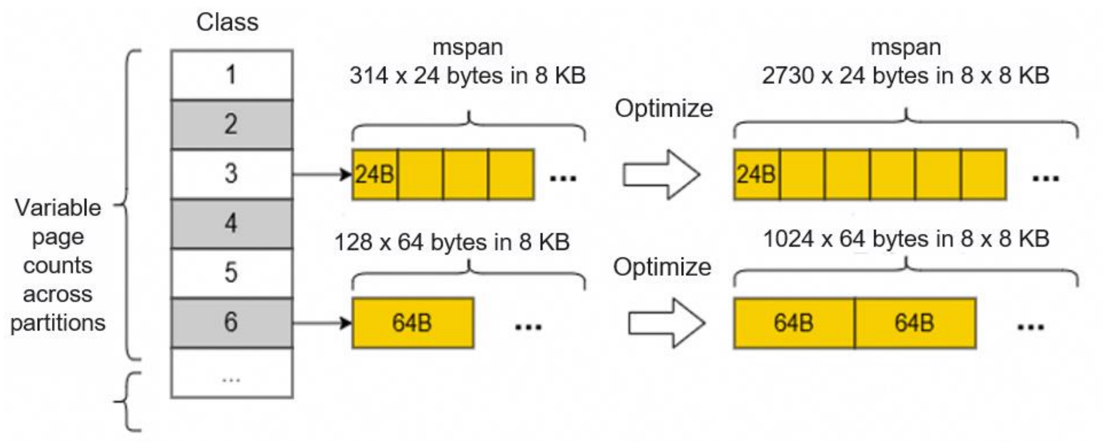


Data loading in advance via software prefetch
Reduced data cache misses
Improved service performance
Internal gcBackgroundUtilization parameter, fixed at 0.25
External GC parameters, adjustable at runtime
Set according to x86 characteristics, not dynamically adjustable
Extracted as runtime parameters, dynamically adjustable

Runtime GC optimization

- Memory page partitioning optimization

  Go utilizes an 8 KB page as its fundamental unit by default. When small objects are managed using single 8 KB pages, frequent creation and retrieval of memory objects may occur. Partitioning by size classes and utilizing optimal-granularity pages for allocation and deallocation significantly improve memory management efficiency. In Go malloc benchmarking, this optimization yields an average performance increase of 10%, with the MallocLargeStruct test case showing a significant increase of 50.29%.



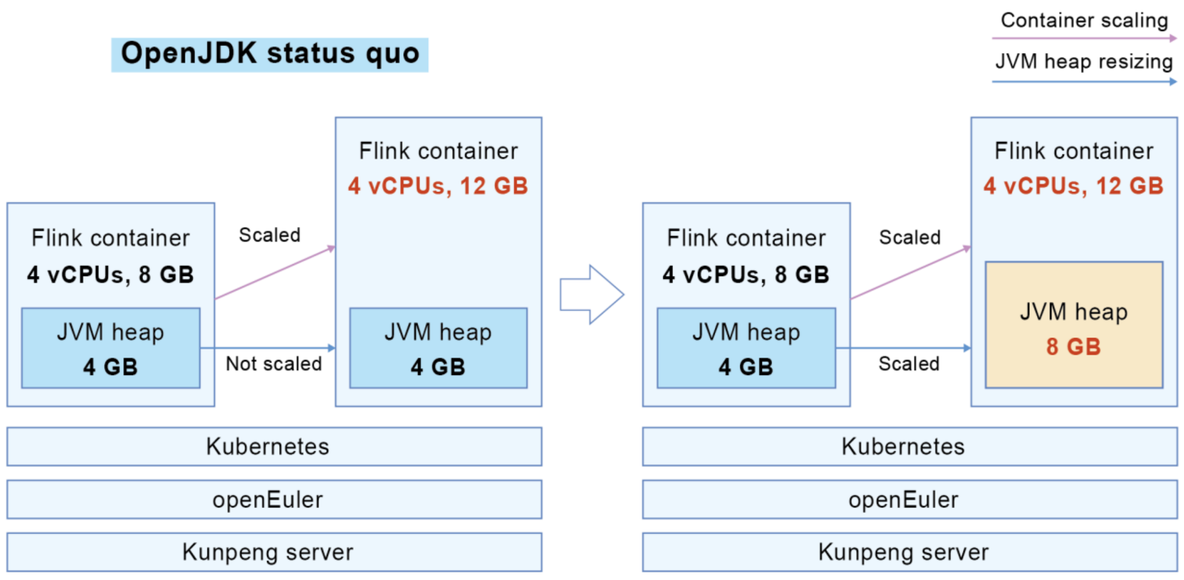
Application Scenarios

As a piece of foundational Go software, Go for openEuler is used in Linux environments such as openEuler, covering cloud native, distributed storage, and cloud gaming workloads.

Heap Resizing by the BiSheng JDK

Feature Description

In modern containerized deployments, most container environments support resource scale-up. However, OpenJDK 8 limits maximum heap size configuration to the startup stage only. This restriction prevents Java applications from utilizing additional memory provided by container scaling without a process restart. To address this issue, BiSheng JDK introduces online heap memory resizing for the Garbage-First Garbage Collector (G1GC) and Parallel Scavenge Garbage Collector (PSGC) across BiSheng JDK 8/21/integrated version. This feature allows users to dynamically update the Java heap memory limit during runtime, eliminating the need for a JVM restart.



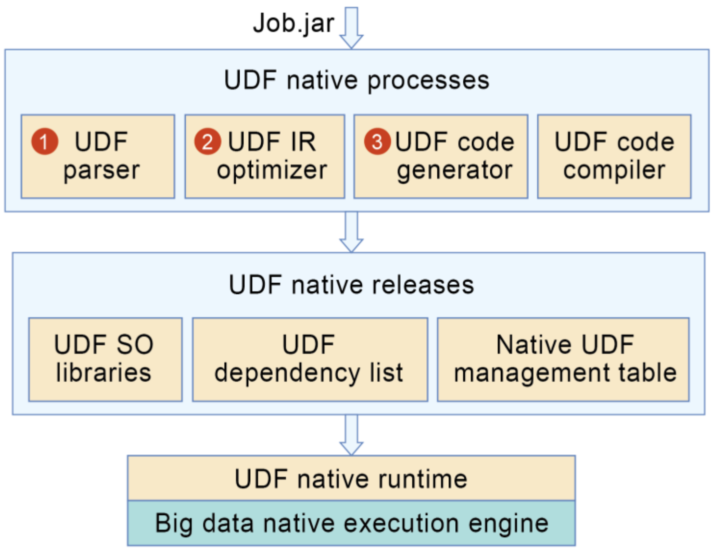
Application Scenarios

This capability is crucial for Internet and other container-based deployments that require Java application heap memory to resize online in response to container resource scale-up. It has been implemented in JD big data scenarios.

UDF Automatic Native Framework

Feature Description

The UDF automatic native framework addresses the inefficient JVM execution often seen in open source big data systems. It automatically converts Java user-defined functions (UDFs) into C/C++ native UDFs, significantly boosting big data processing performance through efficient memory management and hardware affinity. Essentially, the framework implements a seamless, automatic Java UDF native acceleration mechanism. It comprises the UDF parser, UDF IR optimizer, UDF code generator, and UDF code compiler modules.



The UDF parser automatically converts the bytecode of a service JAR package into Intermediate Representation (IR) code and extracts UDF code by identifying specific features. The UDF IR optimizer optimizes the IR through automatic memory object management and hardware-affinity acceleration. The UDF code generator translates the optimized IR into native code, and the UDF code compiler compiles the native code into native binaries online. Finally, these native binaries are deployed to big data execution nodes, where the native execution engine dynamically loads and executes the native binaries to enhance processing performance.

Application Scenarios

The UDF automatic native framework is designed to seamlessly integrate with the Flink big data native execution engine. By leveraging a Flink DataStream native base library, it achieves automatic native acceleration for Flink DataStream UDFs, all without requiring any user intervention.

De-optimization Observability in BiSheng JDK 17

The JDK Flight Recorder (JFR) Streaming API of JDK 17 is a key feature that enables JFR to move from post-event static analysis to real-time monitoring.

In conventional JFR usage, a user must start recording, stop recording, dump contents into a .jfr file, and finally use Java Mission Control (JMC) tools for offline analysis. This is a post-event analysis mode, which is effective for troubleshooting problems that have occurred.

The Streaming API introduces a new mode, which allows a Java application to continuously subscribe to and consume JFR event streams from the JVM in real time without interrupting JFR recording or generating a complete .jfr file.

Feature Description

When using the Streaming API, a user can obtain JFR events such as de-optimization events before the current time from positions marked as ******.

// 1. Create a recording stream.

RecordingStream rs = new RecordingStream();

```
// 2. Enable events of interest and configure related settings.
rs.enable("jdk.GCPhasePause").withPeriod(Duration.ofSeconds(1));
rs.enable("jdk.Deoptimization").withPeriod(Duration.ofSeconds(1));
// 3. Subscribe to an event and set corresponding event handler (callback function).
rs.onEvent("jdk.GCPhasePause", event -> {
// Read the following fields from the event.
Duration = event.getDuration("duration");
String name = event.getString("name"); // For example, "GC Pause"
    *****************
    shell:    jcmd JFR.start delay=-1 filename=xxx.jfr
    *****************
});
// 4. Start the stream (non-blocking call).
rs.startAsync();
```

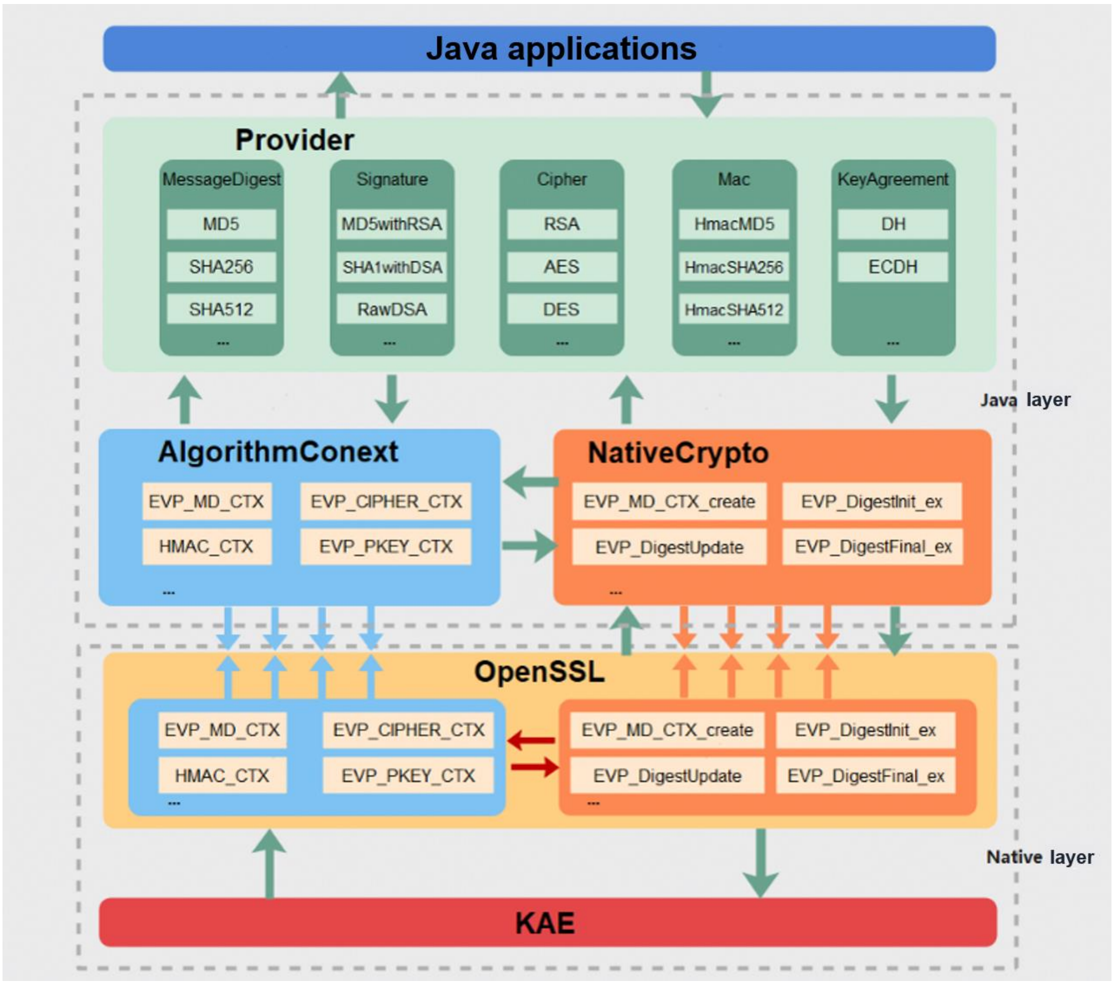
Application Scenarios

The combination of online monitoring and post-event analysis ensures that the JFR file captures all events within a specific period preceding the current moment.

BiSheng JDK 21 KAE Enhancements

KAE encryption and decryption is a module of the Kunpeng Accelerator Engine (KAE). It invokes the underlying KAE hardware by modifying the OpenSSL library.

Therefore, application code only needs to call the OpenSSL library, and KAE offers acceleration capabilities as the underlying engine of OpenSSL. KAEProvider is located between the application and OpenSSL and provides a convenient way to use the OpenSSL library and enable KAE acceleration.



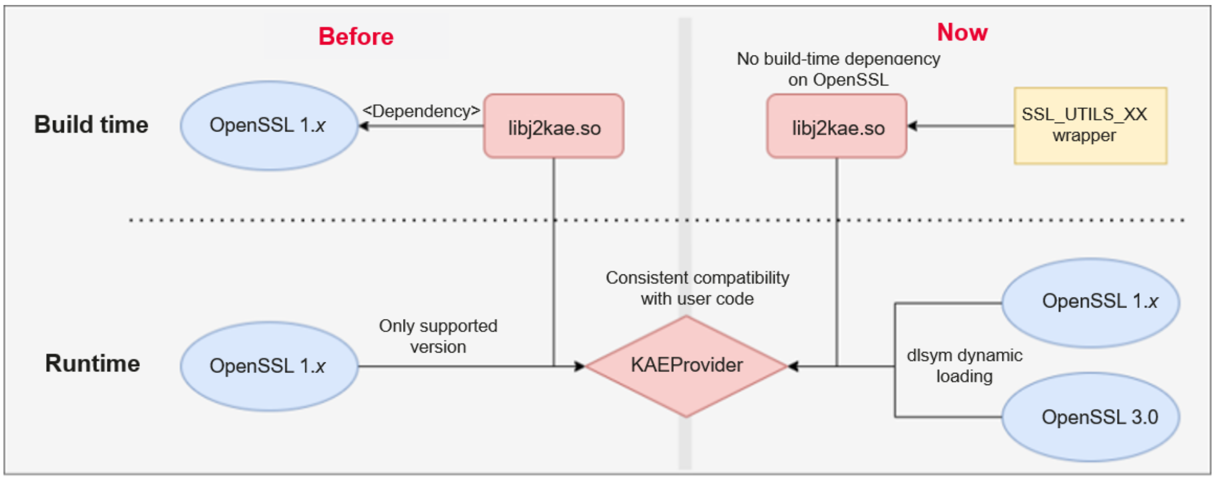
The KAE compression module supports compression of zlib and gzip data formats, increasing the compression bandwidth. In Java applications, compression and decompression features are extensively utilized. Especially in data storage and HTTP services, the CPU usage for compression can exceed 70%, making the compression speed of open source zlib a significant bottleneck. If KAEZlib is enabled for Java applications, service performance will be greatly improved.

Feature Description

KAEProvider supports OpenSSL 1.*x*. In 2021, OpenSSL 3.0 is released, with significant enhancements made to its architecture, scalability, and security, making it a better choice for secure communication applications. However, OpenSSL 3.0 has deprecated, added, or modified several APIs. KAEProvider cannot directly support OpenSSL 3.0 and code adaptation is required.

After this feature is added, the KAEProvider build result remains unchanged, but the build process does not depend on a local OpenSSL library in the build environment (previously, building the target **libj2kae.so** required a local OpenSSL 1.x environment). Instead, the specific OpenSSL version to be used is dynamically determined and loaded at runtime.

The usage of KAEProvider remains unchanged. Users need to manually enable the feature by modifying the **OPENSSL_ENGINES** environment variable and the **kae.useOpenSSLVersion** configuration. The original encryption and decryption logic remains unchanged, and the original functions and usage patterns of the JDK are not affected.



For KAEZip support, the **libzip.so** dynamic library is split into **libzip.so** and **libz.so** in the JDK, which contain the JNI and zlib code, respectively. To enable KAEZlib, use the **-DGZIP_USE_KAE** parameter. During compression and decompression, the Java side does not process the file header and tail. The generation and parsing of the compressed data structure depend on the zlib and gzip compression algorithms of the underlying zlib library.

Application Scenarios

KAEProvider and KAEZip can effectively improve algorithm performance in encryption, decryption, compression, and decompression scenarios.

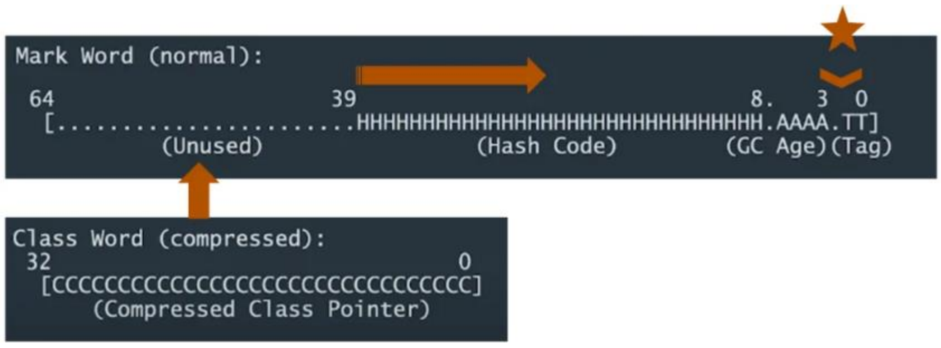
BiSheng JDK 21 Metadata Compression

The storage of Java objects incurs additional overhead known as the object header, which is used to store metadata associated with an object. With the increase of live objects and the existence of a large number of small objects, the issue of space occupied by object headers becomes increasingly severe. This feature compresses the object header size from 128/96 bits to 64 bits on the AArch64 platform.

Feature Description

In 64-bit hotspot, a Java object has a 128-bit object header: a 64-bit mark word and a 64-bit class pointer. The average object size is typically 5 to 6 words, two of which are consistently occupied by the object header. This feature capitalizes on the fact that the high-order bits of the mark word are unused in normal cases. By relocating the class pointer into the high-order bits and shortening hash code while modifying tag bits, the object header is compressed to 64 bits.

Object header before modification:



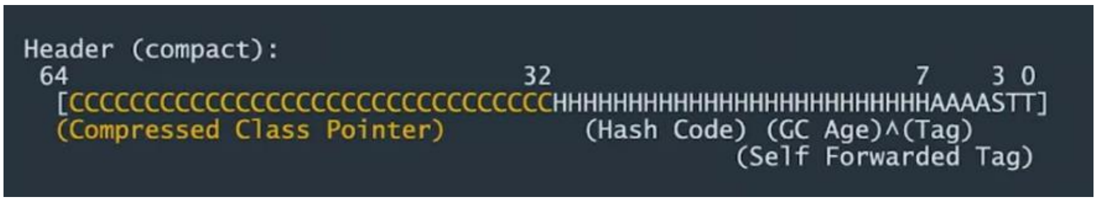
```
Mark Word (normal):
 64                     39                              8.   3  0
  [.......................HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH.AAAA.TT]
          (Unused)                  (Hash Code)        (GC Age)(Tag)
```

```
Class Word (compressed):
 32                               0
  [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
      (Compressed Class Pointer)
```

This feature modifies the object header as follows:

- Relocates the class pointer to the high-order bits of the mark word. Only compressed class pointers are supported, which satisfies the requirements of most applications.
- Compresses hash code to 25 bits, which may cause performance deterioration for ultra-large hash tables.
- Adds a one-bit self-forwarded tag to represent GC copy failure. Previously, this state was indicated by replacing the mark word with a pointer directed back to the object itself.

  Object header after modification:

  ```
  Header (compact):
   64                              32                           7    3 0
    [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHAAAASTT]
    (Compressed Class Pointer)        (Hash Code) (GC Age)^(Tag)
                                                (Self Forwarded Tag)
  ```

  This feature can significantly mitigate memory pressure, achieving:
- Lower heap usage
- Higher object allocation rate
- Fewer GC activities
- Closer object arrangement and better cache locality

Application Scenarios

This feature is suited for a wide range of scenarios, and the actual reduction in memory usage varies depending on applications' data characteristics. It delivers superior E2E performance especially in scenarios involving high frequencies of small object allocation and provides significant performance gains in big data services.

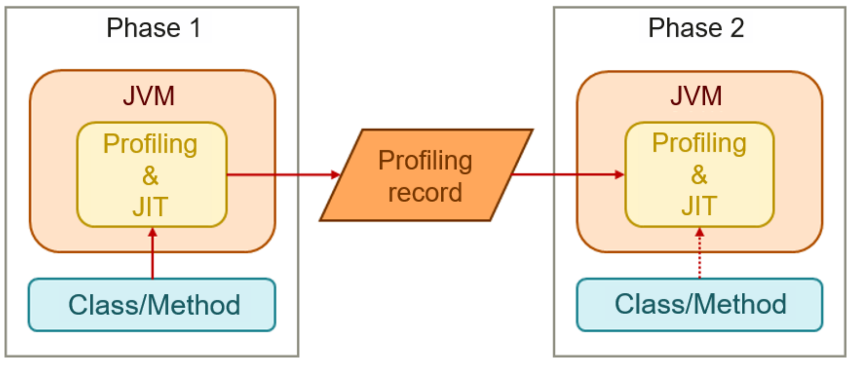
BiSheng JDK 21 JIT Warmup

This feature enables Java processes to reach peak performance rapidly after startup. The transition from process initiation to peak performance is traditionally a lengthy procedure, involving interpreter execution, profiling, C1 compilation, further profiling, and C2 compilation. This workflow integrates profiling and online just-in-time (JIT) compilation, and constitutes a critical part of application warmup. This feature effectively accelerates VM startup and warmup by streamlining this workflow, helping applications reach peak performance faster.

Feature Description

The JProfilecache solution of the BiSheng JDK divides application release into two phases:

- **Recording**: At the conclusion of program execution, the profiling information of hotspot methods—primarily method invocation counts and back-edge counts—along with the associated classes (including parent classes) are exported to a specified file.
- **Pre-compilation**: In the next startup, the JVM reads the classes containing the hotspot methods and performs pre-loading. Simultaneously, these hotspot methods are added to the compilation queue for ahead-of-time (AOT) compilation. This approach bypasses the profiling stage and compiles the hotspot methods into high-performance native code before a user request arrives, thereby reducing the warmup overhead.



Application Scenarios

This feature is particularly suited for cloud-native scenarios with stringent startup speed requirements, as well as Spark big data environments involving the repeated initiation of driver and executor processes. In these contexts, this feature effectively accelerates VM startup and application warmup, enabling applications to reach their peak performance rapidly.

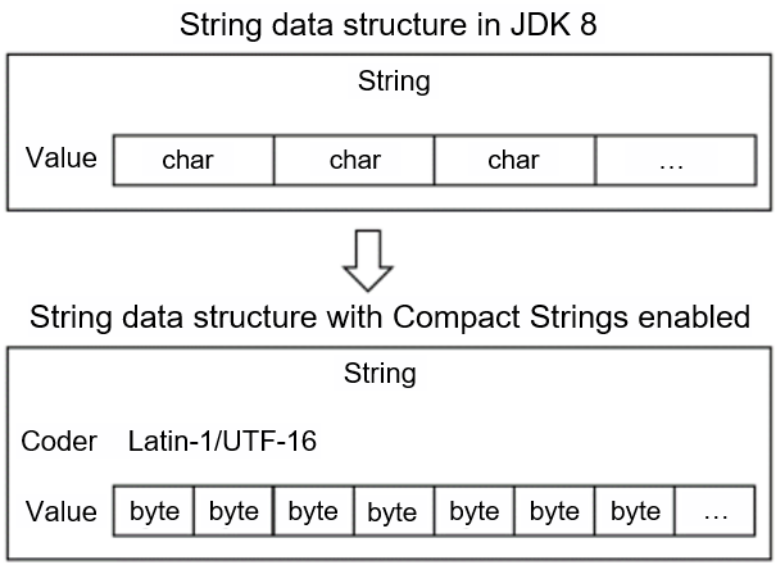
BiSheng JDK 8

Compact Strings is a significant memory optimization feature introduced in Java 9, designed to improve string storage and reduce memory footprint. In Java 8 and earlier versions, strings were internally stored using a char array, where each character occupied 2 bytes (UTF-16 encoding), regardless of the actual content. This led to inefficient memory utilization when processing large volumes of strings containing only ASCII characters. Currently, Compact Strings has been backported from JDK 17 to BiSheng JDK 8, further enhancing the overall performance of BiSheng JDK 8.

Feature Description

Compact Strings replaces the internal representation of the string object from a UTF-16 char array to a byte array coupled with a coder field. This updated string class encodes characters as either Latin-1 (1 byte per character) or UTF-16 (2 bytes per character) based on the content of the string. The coder field indicates the specific encoding used for the string, as detailed below:

- Strings that contain only Latin-1 characters are stored in byte arrays. Each character occupies only one byte.
- Strings that contain non-Latin-1 characters (such as Chinese and Japanese) are still stored in char arrays. Each character occupies two bytes.
- Each string object contains a coder field, which indicates the encoding of this string. The coder field value can be:
  - **0** (Latin-1): Contains only Latin-1 characters.
  - **1** (UTF-16): Contains non-Latin-1 characters.



Application Scenarios

Compact Strings enables significant memory savings in string storage. For strings containing only Latin-1 characters, memory consumption is reduced by approximately 50%. Therefore, Compact Strings is highly effective for string-intensive applications, which further reduces GC frequency and enhances overall system performance.

GCC for openEuler CFGO

The continuous growth in code volume has made front-end bound execution a common issue in processors, which impacts program performance. Feedback-directed optimization techniques in compilers can effectively solve this issue.

Continuous feature guided optimization (CFGO) in GCC for openEuler refers to continuous feedback-directed optimization for multimodal files (source code and binaries) and the full lifecycle (compilation, linking, post-linking, runtime, OS, and libraries). The following techniques are included:

- **Code layout optimization**: Techniques such as basic block reordering, function rearrangement, and hot/cold separation are used to optimize the binary layout of the target program, improving I-cache and I-TLB hit rates.
- **Advanced compiler optimization**: Techniques such as inlining, loop unrolling, vectorization, and indirect calls enable the compiler to make more accurate optimization decisions.

Feature Description

CFGO comprises CFGO-PGO, CFGO-CSPGO, and CFGO-BOLT. Enabling these sub-features in sequence helps mitigate front-end bound execution and improve program runtime performance. To further enhance the optimization, you are advised to add the **-flto=auto** compilation option during CFGO-PGO and CFGO-CSPGO processes.

- CFGO-PGO

  Unlike conventional profile-guided optimization (PGO), CFGO-PGO uses AI for Compiler (AI4C) to enhance certain optimizations, including inlining, constant propagation, and devirtualization, to further improve performance.

- CFGO-CSPGO

  The profile in conventional PGO is context-insensitive, which may result in suboptimal optimization. By adding an additional CFGO-CSPGO instrumentation phase after PGO, runtime information from the inlined program is collected. This provides more accurate execution data for compiler optimizations such as code layout and register optimizations, leading to enhanced performance.
- CFGO-BOLT

  CFGO-BOLT adds optimizations such as software instrumentation for the AArch64 architecture and inlining optimization on top of the baseline version, driving further performance gains.

Application Scenarios

CFGO has good versatility and is suitable for C/C++ applications, such as databases and distributed storage, where the overall system performance bottleneck lies in CPU front-end bound execution. It can typically achieve a performance improvement of 5% to 10%.

ANNC

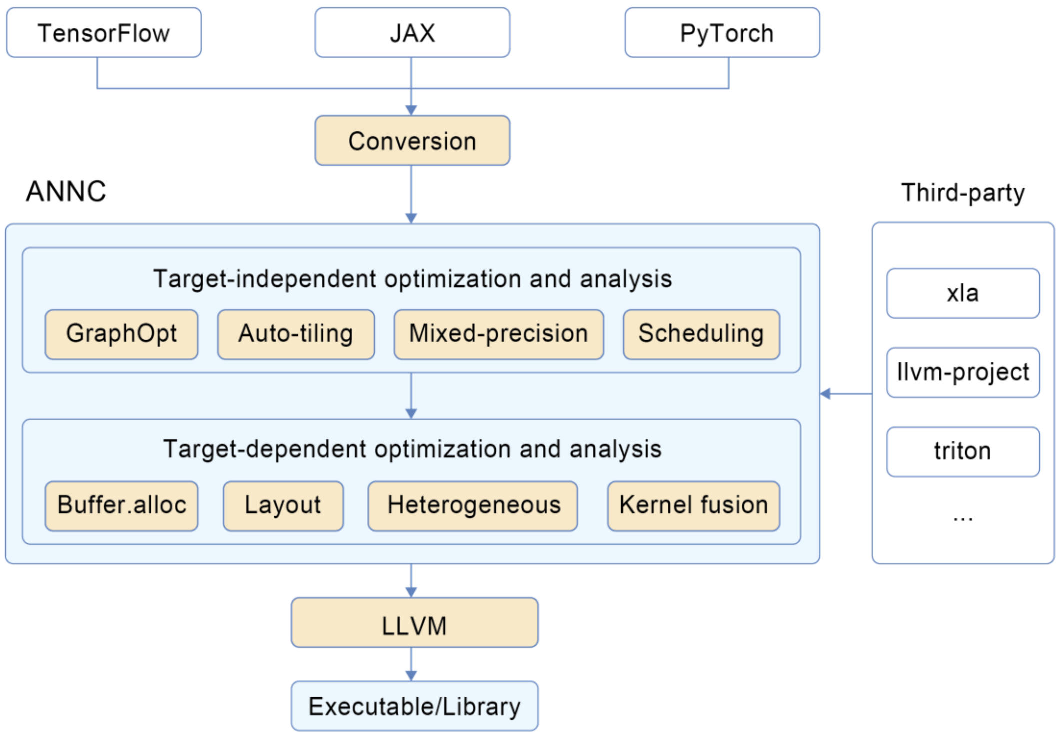
Accelerated Neural Network Compiler (ANNC) is an AI compiler designed for accelerating neural network computations. It focuses on computation graph optimization, high-performance fused operator generation, and efficient code generation and optimization to boost the inference performance of models like recommendation systems and LLMs. It can integrate with mainstream open source inference frameworks and various hardware backends to enhance software extensibility.

Feature Description

Computation graph optimization refines the neural network's computational flow. From an algorithmic perspective, it reduces redundant operations, performs mixed-precision rewriting, and automatically schedules subgraphs to lower the computational load and improve cache utilization. From a hardware architecture perspective, it optimizes the tensor data layout, operator fusion and conversion, and subgraph partitioning and scheduling to further lower the load and fully utilize hardware resources.

Generating and integrating a high-performance fused operator library comprises three parts: front-end computation graph pattern recognition and conversion, high-performance operator library query and integration, and automated operator library generation. At the assembly instruction level, it reduces memory access and accelerate parallel computations through optimization techniques like data prefetch, model parallelism, and new instruction sets.

ANNC aims to accelerate AI inference and reduce power consumption through graph compilation optimization and high-performance operator generation and integration, thereby improving inference efficiency per unit cost. In addition, its software compatibility and ease-of-use design reduce operational costs and environmental impact.



Application Scenarios

ANNC now focuses on graph compilation optimization and operator library selection and invocation. It yields significant benefits for coarse- and fine-ranking models in mainstream search and recommendation systems. Graph compilation optimization achieves even better results for complex embedding layer networks that handle intricate feature processing.

Moreover, ANNC has also laid the groundwork for performance optimization in future scenarios, such as LLMs. By integrating with LLM inference frameworks and combining existing graph-operator fusion and memory layout optimization techniques with core performance optimization methods like GEMM, it aims to reduce inference latency and increase inference throughput.

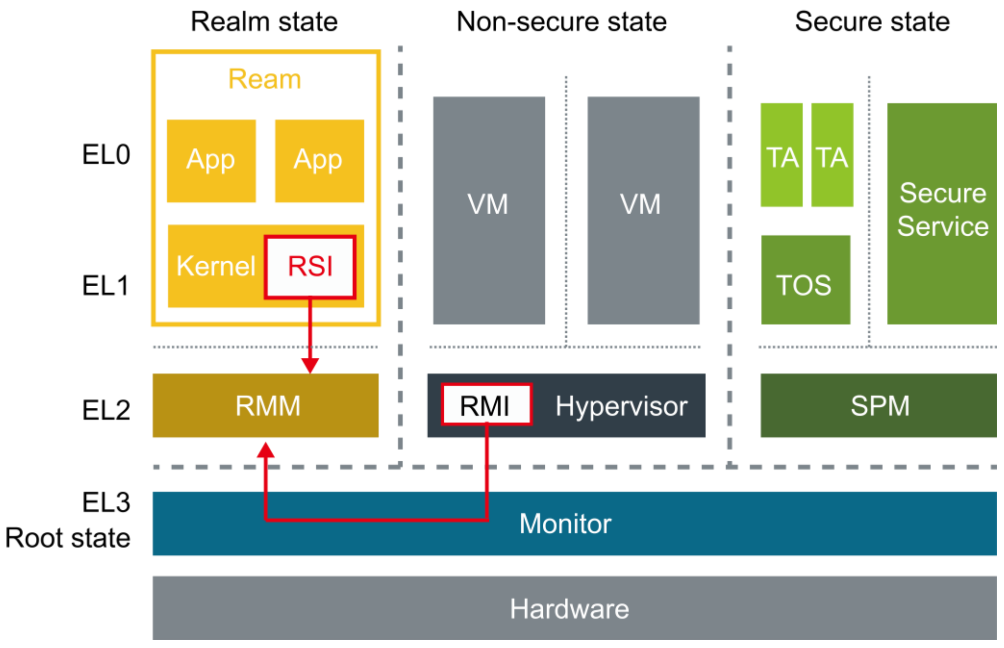
CCA

Arm Confidential Compute Architecture (CCA) is a specification newly introduced in Armv9. It is designed to define a standard confidential computing solution for next-gen computing devices. CCA establishes realms as TEEs to protect the confidentiality and integrity of data and code in use, ensuring effective protection even against privileged infrastructure software or cloud service providers.

Based on the Arm CCA specification, openEuler has implemented support for CCA across its relevant OS components, including KVM, QEMU, libvirt, and the guest kernel. The openEuler community edition provides native support for realm cVMs, fulfilling the security requirement to protect data in use. Crucially, this implementation also offers the ease-of-use and compatibility necessary for seamless integration with traditional application ecosystems and existing VM management software.

Feature Description

Arm CCA utilizes the following core components working in synergy to construct a realm, an isolated and protected execution space. The realm is completely segregated from the normal world in terms of code execution and data access.



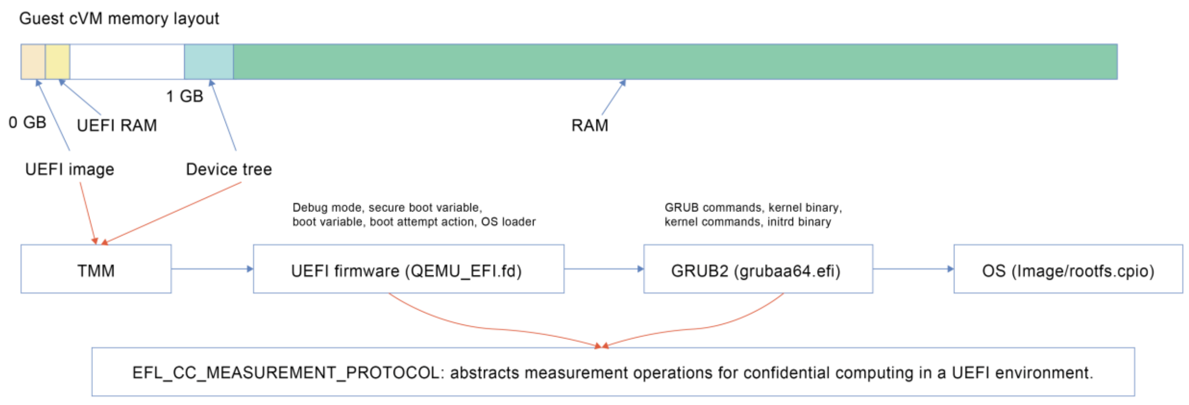
- **Realm**: A realm is the core abstraction of CCA. It is a new execution environment that runs parallel to the normal world and secure world (TrustZone). It is hardware-isolated and designed to host sensitive code and data. It is independent of the host OS and Hypervisor, which can manage a realm but cannot access the content within.
- **Dynamic management**: Hypervisor can dynamically create realms and allocate memory and CPU resources to them as required. However, once a realm is initialized, Hypervisor hands over control to the realm management monitor (RMM), a protected secure virtualization module, and can no longer access the data within the realm.
- **Memory management**: CCA extends the system memory management unit (MMU) to identify and isolate realm memory. Any access attempt from outside the realm (including Hypervisor) is blocked by the hardware, ensuring data confidentiality.
- **Remote attestation**: Each CCA-enabled processor has a unique, hardware-based identity. When a realm starts, it can generate an attestation token that is cryptographically signed by the hardware. Users can obtain this report, verify its signature, and check the component measurements to ensure that their workloads are running in a genuine, unaltered Arm CCA environment.

Application Scenarios

cVMs are mainly used in cloud computing environments to provide high-level security isolation for core workloads. They enable the processing of sensitive data (such as financial records, healthcare information, and intellectual property) on untrusted public clouds while ensuring the confidentiality and integrity of the data. These cVMs can block any unauthorized access, even from cloud service providers. This effectively meets the key requirements of data sovereignty, regulatory compliance, and cross-agency privacy data collaboration (such as joint modeling and analysis).

virtCCA

Feature Description


Guest cVM memory layout
0 GB
UEFI RAM
1 GB
RAM
UEFI image
Device tree
Debug mode, secure boot variable, boot variable, boot attempt action, OS loader
GRUB commands, kernel binary, kernel commands, initrd binary
TMM
UEFI firmware (QEMU_EFI.fd)
GRUB2 (grubaa64.efi)
OS (Image/rootfs.cpio)
EFL_CC_MEASUREMENT_PROTOCOL: abstracts measurement operations for confidential computing in a UEFI environment.

The current virtCCA architecture has this constraint: it only supports a boot mode where the kernel and rootfs are mounted separately. However, in mainstream cloud platforms, VMs typically rely on a GRUB bootloader. This requires integrating the Unified Extensible Firmware Interface (UEFI) firmware (like EDK2), kernel, and initial RAM file system (initramfs) into a single image, such as a QCOW2 file. The enhanced virtCCA addresses this by providing the following functions:

- Single image encapsulation
  - **Unified boot stack**: virtCCA integrates the EDK2 firmware, GRUB bootloader, kernel, and initramfs into a QCOW2 image, creating a complete boot stack.
  - **Streamlined boot process**: GRUB uses a configuration file (**grub.cfg**) to locate the kernel path, which requires the kernel and initramfs to reside on the same file system, for example, ext4 or XFS.
- Secure boot chain
  - **Secure boot**: EDK2 verifies the digital signatures of GRUB and the kernel, ensuring that the boot components have not been tampered with.
  - **Hardware resource collaboration**: virtCCA leverages UEFI runtime services to enumerate hardware devices, providing a virtualized resource pool for VM monitors like KVM.
- Cloud native optimization
  - virtCCA supports snapshot cloning and dynamic rootfs expansion (depending on cloud-init in initramfs).

Application Scenarios

The adoption of UEFI as a boot mode in cloud environments is a core technology for modern virtualization. virtCCA's new UEFI support expands its use cases for cVMs:

- Fast instance boot and auto scaling

  UEFI uses parallel hardware initialization (such as simultaneous detection of CPUs, memory, and storage devices) to significantly shorten the boot time.
- Large-capacity drive support

  Be leveraging the GUID Partition Table (GPT), UEFI breaks the traditional master boot record limit of 2 TB. This allows virtCCA to support cloud drives of 100 TB or more (like

Alibaba Cloud's ESSDs), which is essential for big data storage (such as HDFS) and AI training.

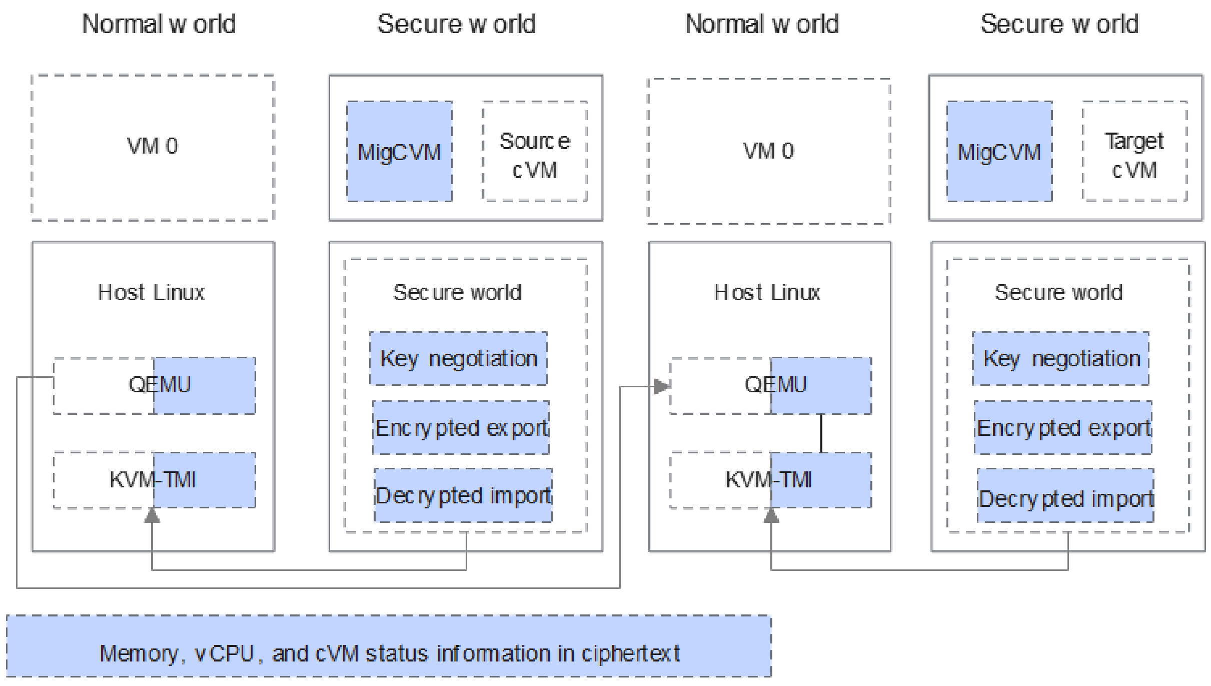
- Automated O&M and batch deployment

  Image standardization: UEFI-compatible images provided by cloud service providers streamline user deployment by enabling GPT partitioning by default.

virtCCA cVM Live Migration

Feature Description

cVM live migration refers to the secure transfer of a running cVM from one confidential computing environment to another verified confidential environment without interrupting services. This process ensures that sensitive data within the cVM remains encrypted or isolated at all stages—before, during, and after the migration.



This feature ensures the confidentiality and integrity of a cVM's state in live migration. This is achieved through the MigCVM component (migration management) and the TMM component (secure world hypervisor). The functional highlights include:

- The source platform authenticates the target platform, and the TMM component performs trustworthiness measurement on MigCVM.
- After the identity authentication is successful, the source platform completes migration key negotiation with the target platform MigCVM.
- An encrypted session is established between the source and target platforms to ensure the security of data migration.
- MigCVM traces the migration status to secure the migration process and handle abnormal status.
- TMM is responsible for exporting and importing the memory status and vCPU status of the virtCCA cVM.

- TMM encrypts the cVM's status data based on the migration key and verifies the data integrity.

Application Scenarios

Mainstream VM management platforms require VMs to support live migration. This enables the transfer of VMs from one physical server to another in server maintenance scenarios, ensuring service continuity throughout the migration process. The virtCCA cVM live migration feature is primarily applicable to:

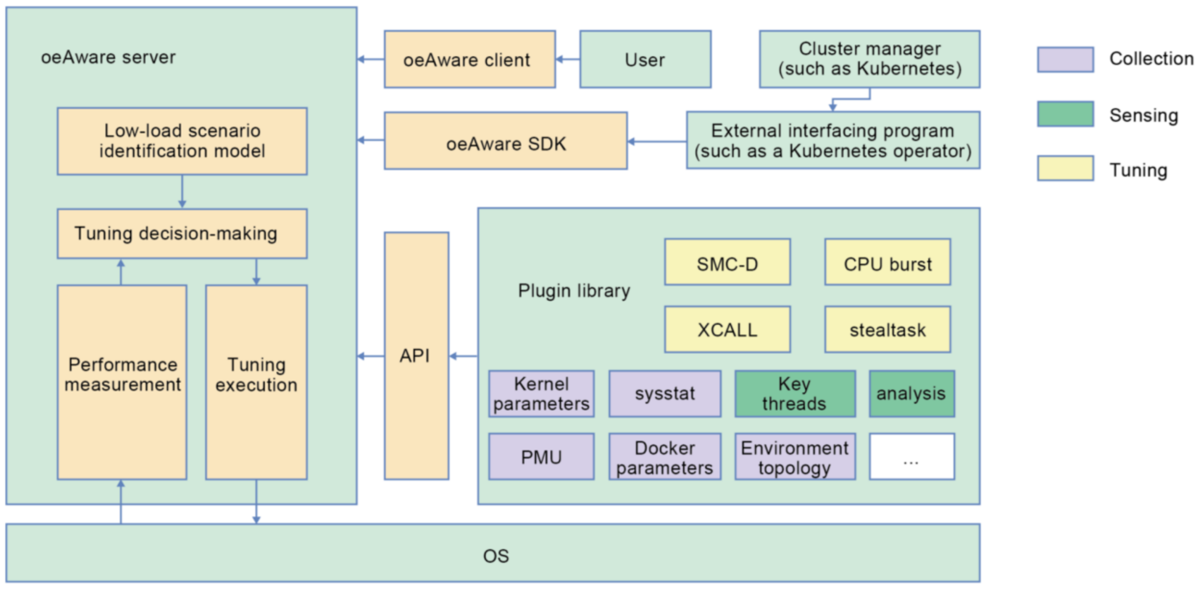
- Hardware and system maintenance and upgrade

  When a physical server undergoes firmware upgrades, hardware replacement, or system updates, cVMs can be migrated to other hosts securely and in real-time, ensuring zero service interruption.
- Load balancing and performance optimization

  When a physical host experiences high CPU or memory load, cVMs with high resource consumption can be migrated to hosts with lower loads. This enables flexible scheduling to ensure optimal overall or partial performance.

oeAware Enhancements

oeAware is a framework that provides low-load collection, sensing, and tuning upon detecting defined system behaviors on openEuler. The framework divides the tuning process into three layers: collection, sensing, and tuning. Each layer is developed as plugins and associated through subscription, overcoming the limitations of traditional tuning techniques that run independently and are statically enabled or disabled.



Feature Description

Each oeAware plugin is a dynamic library built on standard interfaces. A plugin consists of several instances, each acting as an independent functional set for data collection, perception, or tuning. Within these instances are topics, which output the processed data. This data is then shared with other plugins or external applications for further analysis and optimization.

- The SDK enables subscription to plugin topics, with a callback function handling data from oeAware. This allows external applications to create tailored functionalities, such as cross-node information collection or local node analysis.
- The performance monitoring unit (PMU) information collection plugin gathers performance records from the system PMU.
- The Docker information collection plugin retrieves specific parameter details about Docker containers in the environment.
- The system information collection plugin captures kernel parameters, thread details, and resource information (CPUs, memory, I/Os, network) from the current environment.
- The thread sensing plugin monitors key information about threads.
- The evaluation plugin examines NUMA and network information during service operations, suggesting optimal tuning methods.
- The system tuning plugins comprise stealtask for enhanced CPU tuning, smc_tune (SMC-D) which leverages shared memory communication in the kernel space to boost network throughput and reduce latency, and xcall_tune which offers code paths that bypass non-critical processes to minimize system call processing overhead.
- The Docker tuning plugin addresses CPU performance issues during sudden load spikes by utilizing the CPU burst feature.

Constraints

- **smc_tune**: SMC acceleration must be enabled before the server-client connection is established. This plugin is most effective in scenarios with numerous persistent connections.
- **Docker tuning**: This plugin is not compatible with Kubernetes containers.
- **xcall_tune**: The **FAST_SYSCALL** kernel configuration option must be activated.
- **realtime_tune**: This plugin must be used together with the Preempt-RT kernel.
- **net_hard_irq_tune**: This plugin applies only to network communication over TCP.

Application Scenarios

stealtask is ideal for scenarios aiming to boost CPU utilization, such as in Doris. This tuning instance effectively increases CPU utilization and prevents idle CPU cycles.

xcall_tune is designed for applications with substantial system call overhead. It offers code paths that bypass non-critical processes, optimizing system call handling and reducing overhead. However, this approach may compromise some maintenance and security capabilities.

smc_tune (SMC-D) excels in environments demanding high throughput and low latency, including high-performance computing (HPC), big data processing, and cloud platforms. By leveraging direct memory access (DMA), smc_tune significantly reduces CPU load and accelerates interactive workloads.

CPU burst is tailored for high-load container environments like Doris, addressing performance bottlenecks caused by CPU constraints.

realtime_tune is applicable to scenarios that demand low latency, minimal jitter, and strict latency restrictions, such as in the industrial manufacturing sector.

net_hard_irq_tune is best suited for services whose performance is heavily impacted by TCP networking, including applications like Redis and Nginx.

NUMA-aware scheduling applies to workloads, like Spark, where cross-NUMA memory access significantly degrades service performance and fixed core pinning is difficult to maintain.

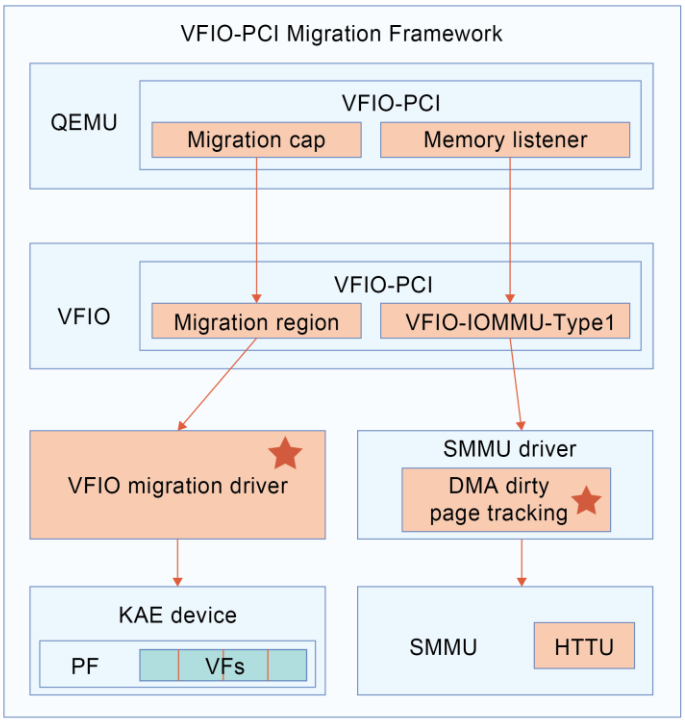
Traffic-based container scheduling is ideal for containerized services, such as containerized MySQL. When the service load is low, it satisfies QoS requirements while using the minimum necessary CPU resources. This reduces CPU migration, idle switching, cache misses, and cross-NUMA access, thereby enhancing resource locality. When the service load is high, it can bypass core pinning constraints to utilize more CPU resources, improving QoS and maximizing overall CPU resource utilization.

vKAE Passthrough Live Migration

Feature Description

The Kunpeng Accelerator Engine (KAE) is a hardware acceleration solution based on the new Kunpeng 920 processor model, featuring HPRE, SEC, and ZIP components for encryption, decryption, compression, and decompression. This enables KAE to significantly reduce processor overhead and boost efficiency. KAE passthrough live migration addresses the scenario where VMs are configured with KAE passthrough devices, offering enhanced operational flexibility and continuous service availability.

SMMU dirty page tracking is a key technology for efficient and reliable live migration of passthrough devices. In the Arm architecture, a purely software-based approach to dirty page tracking incurs significant performance overhead. Hardware Translation Table Update (HTTU) solves this by allowing the hardware to automatically update the SMMU page table status. During a write operation, the write permission bit of the corresponding page table entry is automatically set. During a live migration, the write permission bit of the page table is scanned to collect statistics on dirty pages.



Application Scenarios

This feature empowers live migration for VMs using KAE passthrough devices, making it ideal for fields with high requirements for data security and processing performance, such as finance, cloud computing, and big data processing. Ultimately, it enhances business continuity and operational stability.

Global Trust Authority for Remote Attestation

Feature Description

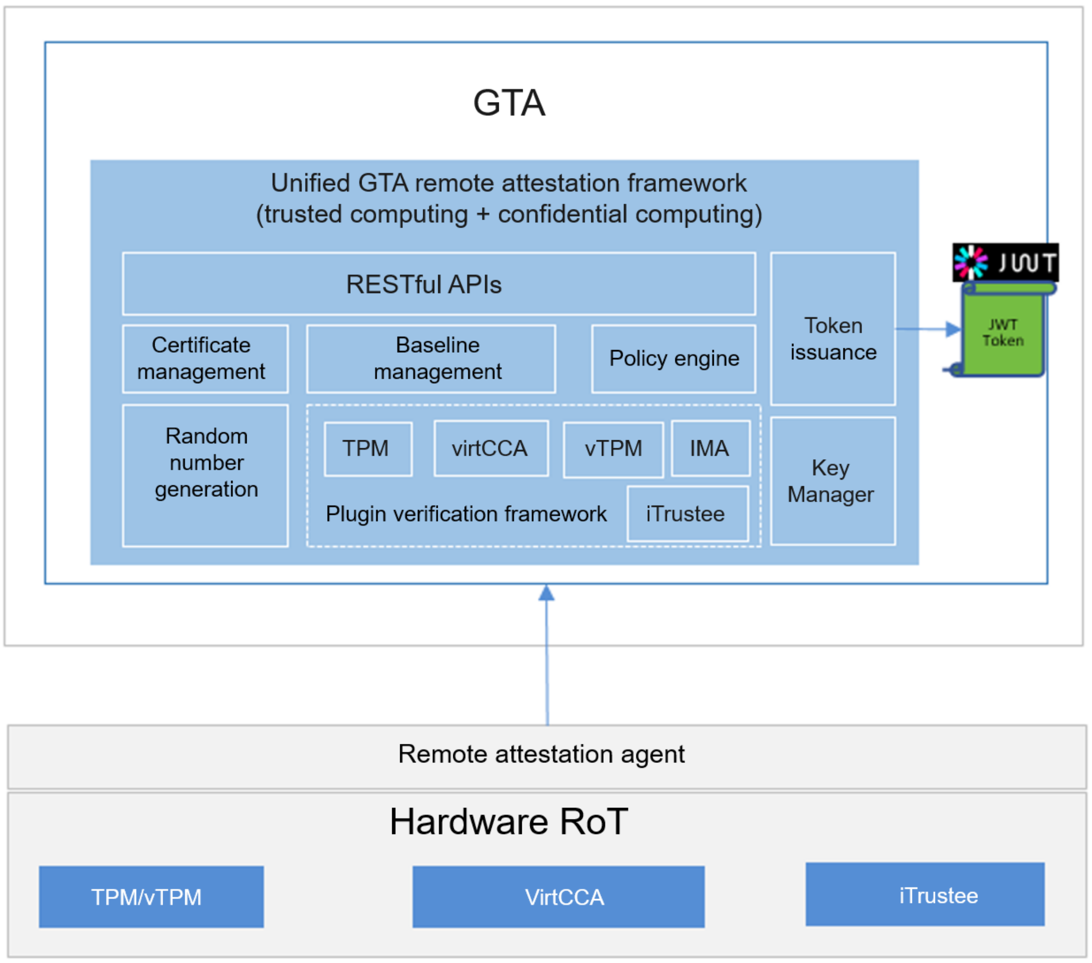
The Global Trust Authority (GTA) remote attestation component adopts a client-server architecture, supporting remote attestation of TPM/vTPM, VirtCCA, and IMA.

- The server provides the remote attestation service framework, which is compatible with trusted computing and confidential computing. It supports the addition, deletion, modification, and query of certificates and policies, quote verification, random number generation, and JWT token generation.
- The client collects local TPM evidence and interacts with the server to verify quotes.

This component provides various capabilities in terms of security and usability.

GTA provides differentiated security competitiveness such as database integrity protection, data link encryption, anti-replay, SQL injection prevention, user isolation, and key rotation.

The passport and background-check models are available. The client supports multiple verification modes, such as scheduled reporting and challenge response. Both the client and server can be deployed using RPM packages and within Docker containers.



Application Scenarios

Remote attestation is a prerequisite for enabling confidential computing and trusted computing. Subsequent computations are allowed only when the operating environment is strictly proven to be secure and trustworthy in the cryptographic sense. Remote attestation should be included as a core component of all end-to-end solutions involving confidential computing. If the operating environment is untrusted, the subsequent tasks must be stopped. This service is widely used in AI model protection, user privacy protection, and key management.

Kuasar Confidential Container

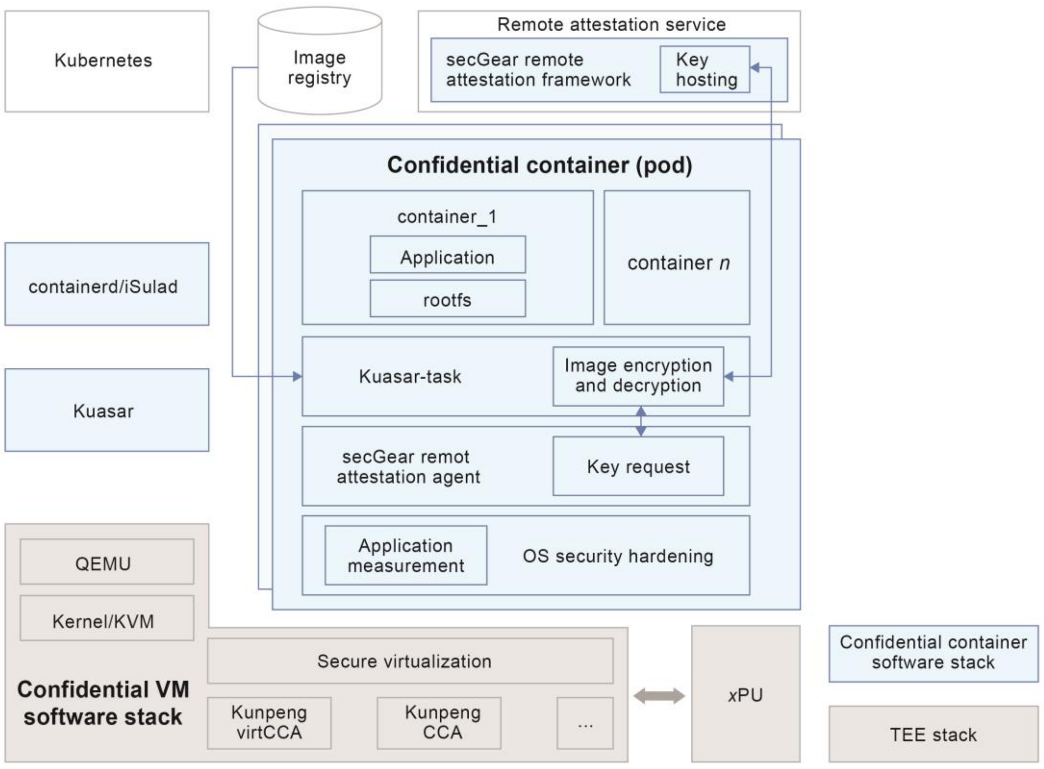
Feature Description

Kuasar has expanded its capabilities to include confidential container support while maintaining its existing secure container functionality. This support can be enabled through iSulad runtime configuration.

The current Kuasar confidential container implementation leverages the iSulad+Kuasar solution to significantly accelerate boot times and drastically reduce memory overhead. On the one hand, the Sandbox API eliminates the need to create a separate pause container during container creation, saving time spent on preparing the pause container image snapshot. On the other hand, the 1:*N* management model allows the Sandboxer process to be persistent. This avoids the cold-start time of the Shim process, greatly accelerating container boot and bringing memory benefits proportional to the number of pods. Finally, Kuasar is implemented in Rust. Compared to Go, Rust provides inherent advantages in memory safety and contributes to overall memory efficiency.

Key functions include:

- Native integration with the iSulad container engine preserves Kubernetes ecosystem compatibility.
- Hardware-level protection via Kunpeng virtCCA technology ensures confidential workloads are deployed in trusted environments.
- The secGear remote attestation framework, which complies with the remote attestation procedures (RATS) (RFC9334), allows containers running in a confidential computing environment to prove their trustworthiness to external trusted services.
- Container images can be pulled and decrypted in confidential containers to protect their confidentiality and integrity.

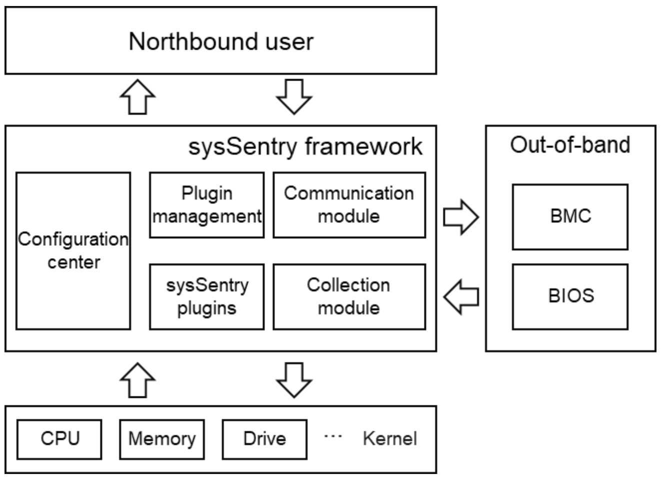


Application Scenarios

Kuasar addresses data security concerns while seamlessly integrating with cloud native ecosystems. It benefits confidential applications from cloud native advantages including high availability, auto scaling, and rapid delivery. The solution finds broad application in confidential computing scenarios spanning AI security, trusted data circulation, and privacy protection.

sysSentry

sysSentry provides a fault inspection framework. By offering a unified northbound fault reporting interface and southbound plugins that support various inspection and diagnostic capabilities, the framework enables the inspection and diagnosis of hardware faults across CPUs, memory, drives, NPUs, and other components.



sysSentry features the following functions:

Unified alarm/event notification service: Receives fault information reported by various plugins and forwards the information in a unified manner. This enables different subscriber services to receive specific fault notifications tailored to their operational needs.

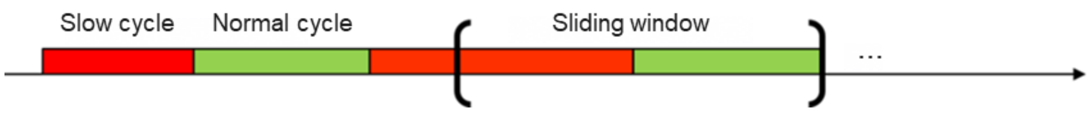
Unified log service: Aggregates and records fault information from all plugins, improving fault location efficiency.

Fault diagnosis/inspection framework: Supports the development and configuration of inspection and diagnostic tasks using a pluggable approach. It allows plugins written in C/C++, Python, or Shell to be managed independently, including start/stop operations and status/result queries.

Lightweight data collection service: Retrieves hardware status information via kernel, BIOS, and BMC interfaces for analysis and use by various plugins. It is designed for high adaptability, supporting a wide range of underlying architectures, software versions, and data collection requirements.

Slow I/O or Drive Detection

Feature Description



Slow I/O detection utilizes a sliding window to analyze the I/O latency data of various drives. A slow I/O event is triggered for a drive when the number of abnormal cycles within the sliding window exceeds a predefined threshold.

Currently, two slow I/O detection plugins are supported: **avg_block_io**, which calculates abnormal thresholds based on average I/O latency, and **ai_block_io**, which utilizes AI clustering algorithms to determine thresholds. Both plugins support monitoring for up to 10 I/O phases: blk-throttle, wbt, iocost, get_tag, plug, deadline, bfq, kyber, hctx, and driver.

Supported drive types include NVMe SSDs, SATA SSDs, and SATA HDDs.

Application Scenarios

- **Hardware faults**: slow I/O caused by drive damage, aging, or other physical hardware issues.
- **I/O stack exceptions**: slow I/O caused by errors in kernel I/O stack configuration or processing.
- **High service loads**: I/O resource over-utilization caused by heavy business demands.

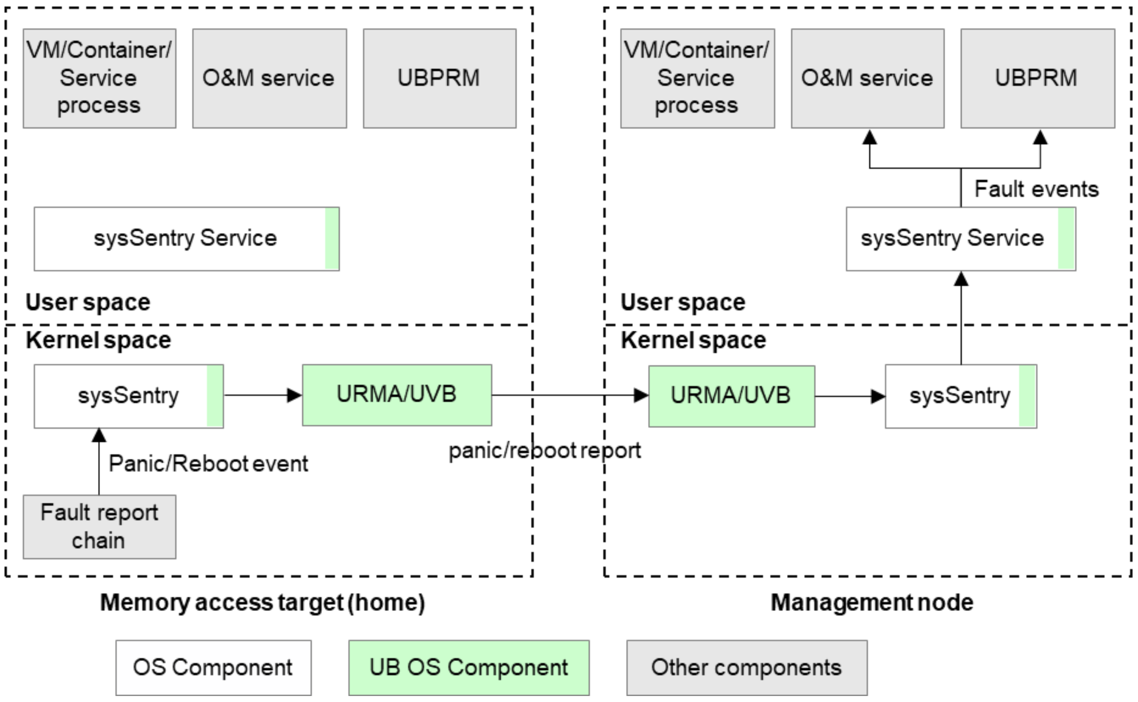
UBPRM

Feature Description

When critical events such as OOM, panic, or reboot occur, sysSentry blocks these events and reports them to UBPRM to prevent data loss or service interruption. It also provides centralized management for UB event reports.

Node Fault Detection and Recovery

When a node on the home side experiences a panic or reboot, the memory it has lent becomes inaccessible, which directly impacts the service processes on the user side. To mitigate this, the data stored in the memory on the home side needs to be migrated.



The report process is as follows:

When a home-side node encounters a kernel failure (such as a panic, reboot, or shutdown), it notifies sysSentry of the impending reset.

1. sysSentry initiates a fault event broadcast via the URMA/UVB channels.
2. The fault event is sent from the URMA/UVB channel of the home-side node to the URMA/UVB channel of the management node.
3. sysSentry on the management node receives the event and notifies UBPRM via sysSentry Service.
4. UBPRM uses **sentryctl** commands to enable sysSentry to hijack panic and reboot events, configure mandatory parameters for cross-node communication, and subscribes to OS panic and reboot events.

Once UBPRM receives the fault report from sysSentry Service, it initiates specific isolation and recovery measures. For example, it can migrate the home-side node's memory to a healthy node and dissolve the borrowing relationship with the faulty node, ensuring that services utilizing pooled memory remain unaffected.

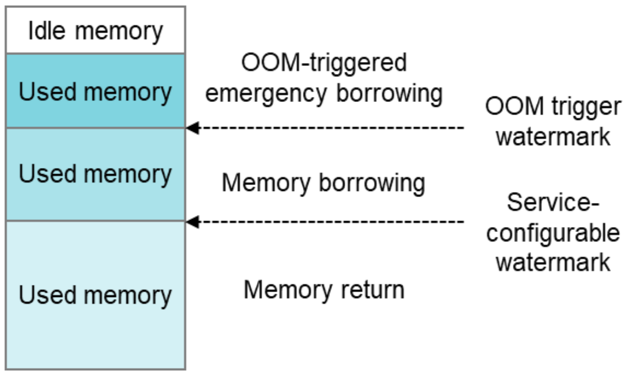
After the fault is rectified, the fault handling results need to be returned. The system performs the recovery according to the following process:

1. UBPRM notifies sysSentry Service of the fault handling results.
2. The fault handling results are sent via sysSentry and the URMA/UVB channels back to the home-side node.

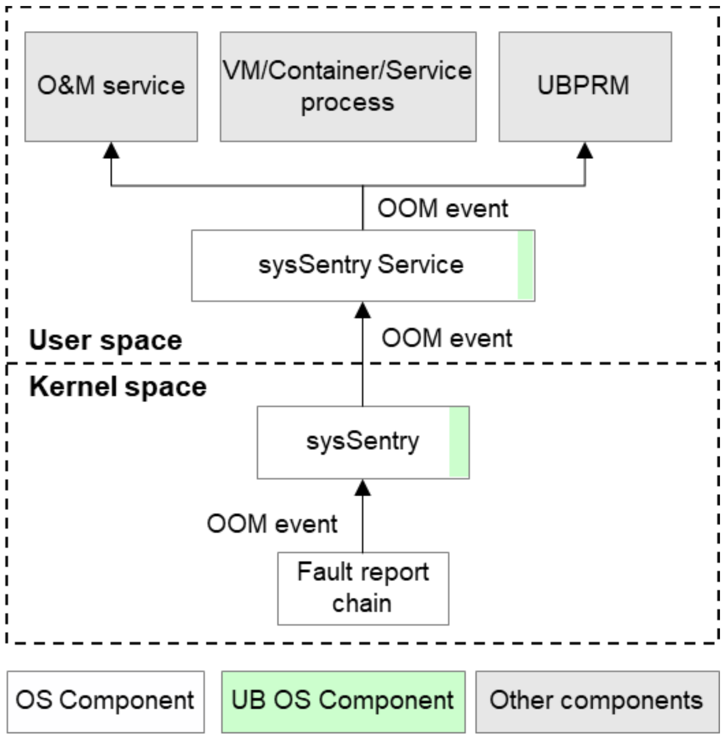
3. sysSentry on the home-side node retrieves the handling results from its local URMA/UVB channel.
4. The home-side node returns to its kernel failure process to complete the reset and reboot.
5. If the home-side node does not receive a notification from UBPRM within a certain period, it will automatically proceed with the reset and reboot.

OOM Detection, Prevention, and Recovery

In memory borrowing scenarios, UBPRM monitors memory usage watermarks at fixed intervals. These watermarks are configurable, enabling the system to dynamically apply memory return or borrowing policy based on actual demand. When a large volume of memory is consumed in a short period, the standard periodic detection mechanism may fail to respond in time. This can lead to the OOM killer terminating critical processes or causing node reboots, resulting in service interruptions. To mitigate this risk, the OOM prevention mechanism is implemented to report impending OOM events to UBPRM and trigger the emergency borrowing policy.



When OOM-triggered emergency borrowing occurs, the report process is as follows.



1. The node experiencing an OOM condition reports an OOM event to sysSentry through the OOM fault report chain.
2. sysSentry reports the OOM event to sysSentry Service.
3. sysSentry Service reports the OOM event to UBPRM or the O&M service.

UBPRM can subscribe to OOM events from sysSentry. When an OOM event occurs on a node, UBPRM receives a notification via sysSentry Service and initiates the following recommended recovery process:

1. UBPRM borrows available memory resources from other nodes and provisions them to the OS of the service node.
2. The OS attempts to fulfill the pending memory request within the OOM context using the newly provisioned resources. If the request is successful, the OS allocates the memory to the application. Otherwise, the OS kills the application. If OOM occurs while kernel space is requesting memory, the panic process is triggered to perform a full hardware reset.

Application Scenarios

In the context of UnifiedBus memory pooling, when a node experiences a critical failure that threatens the availability of borrowed memory, sysSentry hijacks the fault event and notifies the management plane to initiate emergency memory migration, preventing service interruptions.

Raspberry Pi

Raspberry Pi is a series of single-board computers developed by the Raspberry Pi Foundation and Broadcom. They are widely used in industrial automation, robotics, Internet of Things (IoT), education, and enthusiast projects due to their low price, small size, low power consumption, high programmability, and abundant ecosystem. Raspberry Pi 4B and Raspberry Pi 5 are classic products, both using Arm processors. Raspberry Pi 4B is a cost-effective entry-level computer, and Raspberry Pi 5 is an innovative product with significant performance breakthroughs and extended capabilities, making it competitive in high-performance edge computing.

Feature Description

As typical open source hardware products, Raspberry Pi 4B and Raspberry Pi 5 support multiple Linux distributions such as Raspberry Pi OS, Ubuntu, and openEuler. They have extensive peripherals, powerful video encoding and decoding capabilities, LAN on motherboard (LOM), and can be used as independent computer systems.

Application Scenarios

Raspberry Pi 4B and Raspberry Pi 5 are widely used in many fields:

- **Education**: programming language learning such as Python, and electronic experiments using the peripheral interfaces
- **Multimedia and entertainment**: media center or game console
- **IoT and smart home**: sensor nodes or smart home hubs for environment monitoring, automation control, and edge computing
- **Server and network applications**: home servers, lightweight web services, and containerized applications
- **DIY projects**: robot control, 3D printing management, and drone flight control
- **Research and development**: AI experiments and prototype validation in embedded development

- **Industrial automation**: device monitoring, man-machine interface, and machine vision

Enhanced RISC-V Features

ISA-L

Intel® Intelligent Storage Acceleration Library (ISA-L) is an open source function library optimized for storage applications. It features optimized computing code for erasure coding (EC), cyclic redundancy check (CRC), RAID, and compression/decompression.

Feature Description

Originally, ISA-L's acceleration was limited to the x86 architecture, with other platforms relying on standard C implementations that lack optimization. While recent updates have introduced assembly acceleration for Arm, support for RISC-V remains largely restricted to basic C implementations. Consequently, ISA-L cannot yet fully leverage the hardware capabilities of the RISC-V platform, leaving significant room for performance improvements.

To enhance the performance of the ISA-L library on the RISC-V platform, we have made the following optimizations:

CRC Optimizations

Bottleneck: CRC algorithms fundamentally rely on modulo division. Since standard division is computationally expensive, generic implementations typically use precomputed lookup tables. However, lookup tables are complex to handle and offer limited throughput. While most architectures leverage assembly-level polynomial folding to achieve high throughput, ISA-L on RISC-V still relies on basic lookup tables, resulting in a significant performance bottleneck.

Optimizations:

- **Hand-optimized assembly**: We implemented polynomial folding and Barrett reduction using RISC-V assembly, replacing the basic lookup table method with code tailored to the RISC-V instruction set.
- **Register allocation optimization**: By effectively utilizing RISC-V's general-purpose registers, we minimize memory access. Keeping intermediate results in registers significantly reduces access latency.
- **Instruction pipeline scheduling**: Based on RISC-V pipeline characteristics, we applied instruction reordering and loop unrolling. This maximizes execution unit utilization and increases instruction-level parallelism (ILP) and overall throughput.

Application Scenarios

The ISA-L CRC optimizations for RISC-V are designed to accelerate data verification and ensure data integrity. By resolving the performance bottleneck in the RISC-V ecosystem, these optimizations allow devices to verify compressed or decompressed data much more efficiently. Key application scenarios include:

- **Compressed data streams**: CRC values are calculated or verified during data compression and decompression to ensure the correctness of source data. This allows for the seamless integration of data verification in scenarios where high throughput is critical.
- **Network data transmission**: The iSCSI protocol inherits CRC verification at the data packet layer. These optimized algorithms ensure high-speed and reliable data transmission on the network.

- **Silent data corruption prevention**: The T10 Data Integrity Field (DIF) standard relies heavily on CRC. These optimizations accelerate end-to-end data protection, preventing corrupted data from being processed as valid.

Snappy Algorithm

Snappy is a high-speed data compression and decompression library developed by Google. It is designed to strike a balance between extreme processing speeds and reasonable compression ratios. It prioritizes speed over the compression ratio.

Feature Description

- **Fast compression and decompression**: Snappy's core advantage lies in its extremely fast speed. It mainly utilizes a combination of dictionary coding and literal copying to compress data. During compression, Snappy scans the input data for duplicate byte sequences. When a match is found, Snappy replaces the duplicate sequences with a reference consisting of an offset (distance to the previous occurrence) and a length. Data with no identifiable matches is stored as literals.
- **Low CPU overhead**: Due to its focus on speed, Snappy maintains low computational complexity and minimal CPU resource demand. This makes it ideal for scenarios requiring high-speed processing of large datasets in low-latency environments.
- **Speed-first design**: Unlike algorithms such as gzip or bzip2 that prioritize maximum compression ratio, Snappy is designed for use cases where high processing speeds are prioritized over achieving the highest possible compression ratio.
- **Stream-friendly design**: Snappy's data format is inherently suited for streaming compression and decompression.
- **RISC-V SIMD and vector optimization**: To further enhance Snappy's performance, the Vector Extension and Packed SIMD (P-Extension) instructions are leveraged to vectorize core bottlenecks such as sequence searching and bulk data copying. This significantly boosts search efficiency and data throughput.
- **RISC-V memory optimization**: Memory access patterns are refined to maximize the efficiency of the RISC-V cache hierarchy. This involves data alignment and prefetching for literal copies and reference jumps to minimize cache misses.

Application Scenarios

Snappy is widely deployed in environments requiring high-speed data processing and high-throughput storage.

- **Big data storage and processing**: In ecosystems like HDFS, Spark, and HBase, Snappy is often used to compress data blocks and network streams to improve I/O efficiency and reduce network latency.
- **Log file compression**: Snappy is ideal for rapidly compressing massive volumes of log data. It achieves substantial storage savings without creating system bottlenecks from compression.
- **Memory data structure snapshots**: Snappy enables the rapid generation of memory database or cache system snapshots for persistence or disaster recovery.
- **Internal RPC/network communication**: Snappy compresses data transmitted between services to minimize bandwidth consumption while maintaining low latency.
- **High-performance RISC-V servers and clusters**: In HPC clusters and data center servers that adopt the RISC-V architecture, these low-level Snappy optimizations further boost I/O performance and overall system throughput for big data workloads.

- **Edge and embedded RISC-V devices**: On resource-constrained RISC-V edge devices, an optimized Snappy library delivers high-speed compression/decompression with lower power consumption and fewer clock cycles. This maximizes energy efficiency in devices that must process data locally.

LZ4 Algorithm

LZ4 is a high-performance, real-time data compression algorithm. It is essential for performance-critical scenarios such as OS kernels, file systems, and network transfers. This white paper details unaligned memory access optimization for LZ4 on the RISC-V architecture. By leveraging hardware detection and targeted optimization policies, these improvements significantly boost compression performance on RISC-V platforms.

Feature Description

Unaligned Memory Access Optimization

LZ4 relies heavily on unaligned memory access during data compression. Performance for these operations varies significantly across CPU architectures. Following optimization patterns from Arm (such as ARMv6), direct memory access provides a major performance boost over standard **memcpy()**, provided the hardware natively supports it.

This optimization focuses on the RISC-V architecture, specifically for processors supporting the Zicclsm extension. It implements an intelligent memory access selection mechanism:

- **Intelligent hardware detection**: The built-in GCC macro **__riscv_zicclsm** is used to accurately detect if the RISC-V processor supports the Zicclsm extension.
- **Tiered optimization policy**: The algorithm maintains its existing three-level optimization hierarchy (levels 0, 1, and 2). RISC-V platforms with Zicclsm support are now assigned to level 2, the highest performance tier.
- **Security and compatibility**: This optimization is only active when hardware support is explicitly confirmed. For RISC-V platforms without Zicclsm or other architectures, the algorithm falls back to its original behavior.

Application Scenarios

- **Cloud native and edge computing**

  In openEuler-based RISC-V cloud servers and edge devices, LZ4 serves as a core compression component. Its performance directly impacts overall system efficiency. This optimization is particularly beneficial for:

  - **Large-scale data processing**: Enhances performance in high-frequency data compression and decompression tasks, such as big data analytics and log processing.
  - **Real-time data transmission**: Enhances compression performance in latency-sensitive applications like network communication and remote storage, improving user experience.
  - **Resource-constrained devices**: Reduces power consumption and extends battery life in RISC-V IoT and embedded systems.
- **AI and HPC**

  AI training and inference workloads often involve heavy data compression during preprocessing. The optimized LZ4 excels in the following scenarios:

  - **Model parameter compression**: Accelerates the deployment and execution of AI models on RISC-V platforms.
  - **Training data processing**: Speeds up data loading and preprocessing, reducing CPU/GPU idle time.

- **Distributed computing**: Optimizes data transfer efficiency between nodes in RISC-V clusters.

OpenSSL Encryption and Decryption

OpenSSL is the world's most widely used cryptography library. The performance of its core algorithms directly determines the efficiency of critical security operations, including digital signatures, identity authentication, and data encryption and decryption.

Feature Description

With the rapid growth of the RISC-V open standard instruction set architecture (ISA), OpenSSL still primarily relies on generic C implementations. This lack of architecture-specific optimization prevents cryptographic algorithms from reaching their theoretical hardware limits. Consequently, there is significant potential for performance gains and an urgent need for technical improvements.

To enhance OpenSSL performance on RISC-V, we have made the following optimizations:

- **RSA Optimizations**

  Bottleneck: RSA performance depends heavily on the efficiency of modular multiplication and exponentiation. While Montgomery multiplication is the standard optimization for these operations, most platforms use highly tuned assembly implementations. On RISC-V, however, OpenSSL has historically relied on generic C code. This failure to leverage architecture-specific features creates a major performance bottleneck.

  Optimizations:

  - **Architecture-specific assembly**: Montgomery multiplication is optimized in RISC-V assembly to eliminate the overhead associated with generic C code.
  - **Register allocation optimization**: By maximizing the use of RISC-V's general-purpose registers, intermediate calculation data is kept in the registers. This reduces memory access frequency and minimizes data latency.
  - **Instruction pipeline scheduling**: We utilized instruction reordering and loop unrolling tailored to RISC-V pipelines. These adjustments enhance instruction-level parallelism and throughput.

- **AES-128-CBC Optimizations**

  Bottleneck: AES-128-CBC performance is largely dictated by the efficiency of block encryption and matrix transformations. Generic C implementations lack support for RISC-V Vector Crypto Extensions. Furthermore, while CBC mode allows for parallel decryption, the current RISC-V implementation processes blocks serially, creating a significant performance bottleneck.

  Optimizations:

  - **Hardware-accelerated assembly**: We implemented optimized assembly for AES-128-CBC using the RISC-V Zvkned extension. This enables the algorithm to leverage dedicated hardware instructions to accelerate computing.
  - **Parallel decryption processing**: Taking advantage of the fact that CBC decryption can be performed in parallel, we optimized the decryption workflow using a batch processing strategy. By implementing a 6-block parallel decryption loop, we significantly increased data throughput.

- **SM2 Optimizations**

  Bottleneck: SM2 performance is primarily restricted by large-integer modular arithmetic. Generic C implementations often lack the fast modular reduction algorithms specifically designed for the SM2 recommended prime and architecture-specific acceleration for RISC-V, leaving a significant performance gap.

Optimizations:

- **Algorithm optimization**: We optimized the generic C code by implementing a fast modular reduction method tailored to the SM2 prime.
- **Register allocation optimization**: By maximizing the use of RISC-V's general-purpose registers, we minimize frequent memory accesses. Keeping intermediate calculation data within the registers dramatically lowers data access latency.
- **Instruction pipeline scheduling**: We applied instruction reordering and loop unrolling specifically tuned for RISC-V processor pipelines. This increases instruction-level parallelism and boosts throughput.

Application Scenarios

OpenSSL performance optimizations for RISC-V accelerate core security workloads and cryptographic calculations. These enhancements enable RISC-V devices to maintain high security while significantly boosting processing speeds. By addressing diverse application requirements, these updates eliminate performance bottlenecks for commercial cryptography across the RISC-V ecosystem.

- Key RSA application scenarios
  - **Digital signature and certificate verification**: Essential for e-commerce, financial systems, and software distribution. Optimized RSA significantly accelerates digital certificate chain validation during HTTPS handshakes, reducing access latency.
  - **Secure key exchange**: Ideal for VPN gateways, SSH servers, and TLS termination. Performance gains shorten connection setup time and increase throughput in high-concurrency environments.
  - **Identity authentication**: Critical for enterprise single sign-on (SSO) and identity and access management (IAM) platforms. The optimization allows for higher volumes of concurrent authentication requests, boosting overall system responsiveness.
- Key AES-128-CBC application scenarios
  - **Encrypted data transmission**: Critical for network devices, routers, and firewalls. Optimized AES-128-CBC accelerates packet encryption, ensuring faster data transmission across networks.
  - **Storage data protection**: Suited to database, file system, and cloud storage encryption. Performance gains minimize the overhead on business systems, significantly improving data access efficiency.
  - **Batch data processing**: Essential for big data platforms, backup systems, and log encryption. The optimization drastically increases throughput for large-scale cryptographic tasks, reducing total processing time.
- Key SM2 application scenarios
  - **Digital signature and certificate verification**: Vital for e-commerce, financial systems, and software distribution. Optimized SM2 significantly accelerates certificate chain validation in Guomi (GM/T) HTTPS/TLS handshakes, effectively reducing user latency.
  - **Secure key exchange**: Essential for SM2-based key negotiation in VPN gateways, SSH servers, and TLS termination. Performance enhancements shorten connection setup time and increase the capacity to handle high-concurrency traffic.
  - **Identity authentication**: Critical for enterprise SSO and IAM platforms. The optimization allows for higher volumes of concurrent authentication requests, boosting overall system responsiveness.

Go Backport: RVA23 Profile

Feature Description

openEuler 24.03 LTS SP3 introduces the **GORISCV64** environment variable and RVA profile support (up to RVA23) for Go 1.21. This enables Go to leverage advanced RISC-V extensions like Zbb and V, significantly boosting performance on RISC-V.

Application Scenarios

Enabling the RVA23 profile allows Go to utilize advanced instruction extensions on supported RISC-V CPUs, significantly boosting performance. Benchmark tests show a 21.75% reduction in execution time for **math** operations and a 32.65% reduction for **math/bits** operations, with similar gains expected in real-world applications.

OpenSSL Backport: SHA-2 Assembly Optimization on RISC-V

Feature Description

openEuler 24.03 LTS SP3 uses OpenSSL 3.0.12 as its baseline, while the upstream has advanced to the 3.6 series. To enhance OpenSSL performance on RISC-V, we backported key cryptographic assembly optimizations from the upstream mainline.

Specifically, the backported SHA-2 assembly optimization now leverages the **hwprobe** interface to detect CPU instruction extensions at runtime. This enables OpenSSL to dynamically select the most efficient optimized function for specific hardware, significantly boosting SHA-2 performance.

Application Scenarios

The core application scenarios of SHA-2 include:

- Data integrity verification

  Purpose: Ensure that data has not been tampered with.

  Typical application: Calculate hash values for files and messages to verify software download integrity and data transmission accuracy.
- Digital signature foundation

  Purpose: Ensure signature efficiency and security.

  Typical application: Sign the hash of a message rather than the message itself, serving as a preprocessing step for signature algorithms such as SM2 and RSA.
- Secure password storage

  Purpose: Ensure password confidentiality.

  Typical application: Store salted password hashes in system databases instead of plaintext. Authentication is then performed by comparing these hash values.

OpenSSL Backport: MD5 Assembly Optimization on RISC-V

Feature Description

openEuler 24.03 LTS SP3 uses OpenSSL 3.0.12 as its baseline, while the latest upstream version has progressed to the 3.6 series. To further optimize performance on RISC-V, we backported key cryptographic assembly optimizations from the upstream mainline to version 3.0.12.

Following the backport of MD5 assembly optimization for the RISC-V architecture, OpenSSL now utilizes the **hwprobe** interface to detect supported instruction extensions at runtime. This enables the dynamic selection of optimal functions tailored to the specific CPU, resulting in a significant performance boost for MD5 cryptographic operations.

Application Scenarios

The core application scenarios of MD5 include:

- Non-cryptographic security verification

  Purpose: Quickly check for accidental data errors.

  Typical application: Detect non-malicious errors during network transmissions or within file systems.

- Legacy system compatibility

  Purpose: Ensure compatibility with older systems and protocols.

  Typical application: Legacy file verification and data deduplication scenarios where cryptographic security is not a primary requirement.

OpenJDK 21 Backport: RISC-V Optimizations

As the backbone of enterprise and large-scale distributed systems, Java powers critical infrastructure across finance, telecommunications, and the Internet. We have backported key RISC-V optimizations from the OpenJDK master branch into OpenJDK 21. This includes integrating RVA23 profile instruction extensions, assembly-level optimizations for cryptographic and encoding algorithms, and intrinsic implementations for core Java operations. These enhancements significantly improve OpenJDK 21's efficiency on RISC-V across networking, storage, and concurrency, providing a high-performance, production-grade Java runtime for the openEuler RISC-V ecosystem.

Feature Description

- **Backporting RVA23 profile instruction extensions**

  We have backported support for the RVA23 profile instruction extensions, including Zfa, Zacas, Zabha, Zvkn, and Zicond from the OpenJDK master branch to OpenJDK 21. Building on this, we integrated the Linux **hwprobe** system call mechanism. This enables the Java virtual machine (JVM) to dynamically detect the processor's actual instruction set capabilities during startup and just-in-time (JIT) compilation, ensuring that optimized code paths are only enabled when the hardware explicitly supports them. For RISC-V platforms without these extensions or in environments where detection fails, the system automatically falls back to generic implementations, maintaining full functional correctness and cross-platform compatibility.

- **Backporting assembly optimizations for cryptographic algorithms**

  We have introduced high-performance, RISC-V-specific assembly implementations for core cryptographic and encoding algorithms, including SHA-1, SHA-2, ChaCha20, Poly1305, CRC32, Adler32, and Base64. By leveraging advanced instruction extensions such as Zvkn (vector crypto), these implementations maximize the synergy between RISC-V vector and scalar instructions. This results in multi-fold performance improvements under typical workloads. Furthermore, runtime capability detection ensures that these optimized paths are only activated in environments with the necessary hardware support.

- **Backporting intrinsic implementations for core Java operations**

  We have implemented intrinsic optimizations for high-frequency Java operations, including **Object.hashCode()**, **Math.copySign()**, **BigInteger**, and lightweight **fastlock**. By utilizing specific RISC-V instructions to generate more compact and efficient machine code, these optimizations significantly reduce method call overhead and computational latency while ensuring full compatibility with Java semantics.

Application Scenarios

- **High-performance Java Runtime Environment (JRE)**

  On openEuler-based RISC-V cloud servers, Java applications play a critical role in hosting web services, microservice governance, and middleware. The optimizations in OpenJDK 21 significantly enhance operational efficiency in the following scenarios:

  - **High-concurrency web applications**: The **fastlock** optimization and atomic operation extensions (Zacas and Zabha) reduce lock contention overhead. Additionally, the acceleration of **hashCode()** and **BigInteger** improves object processing and identity authentication, leading to a significant reduction in API response latency.
  - **Microservice communication and serialization**: Assembly-level optimizations for encoding and checksum algorithms, including Base64, CRC32, and Adler32, accelerate JSON/XML serialization, gRPC/HTTP communication, and data integrity verification between services.

- **Security and network-intensive systems**

  As RISC-V is increasingly deployed in edge security gateways and Trusted Execution Environments (TEE), encryption and authentication have become primary performance bottlenecks. These optimizations address the following scenarios:

  - **TLS/SSL acceleration**: Vectorized assembly implementations of algorithms such as SHA-1, SHA-2, ChaCha20, and Poly1305 significantly boost handshake speeds and data encryption throughput for secure protocols like HTTPS and QUIC.
  - **File system and storage verification**: Optimized CRC32 and Adler32 accelerate block verification, log writing, and snapshot generation for object storage systems like MinIO.

Technical specifications

- **Architecture**: RISC-V 64-bit (requiring support for Zfa, Zacas, Zabha, Zvkn, and Zicond extensions)
- **OS**: Linux kernel 6.6 or later
- **OpenJDK version**: 21.0.9

Security and compliance

The backported content strictly adheres to the RVA23U64 profile, ensuring robust hardware compatibility and security. All instruction extensions and optimized code paths undergo rigorous hardware feature detection before activation. This prevents undefined behavior or performance regressions on hardware that does not support these specific extensions.

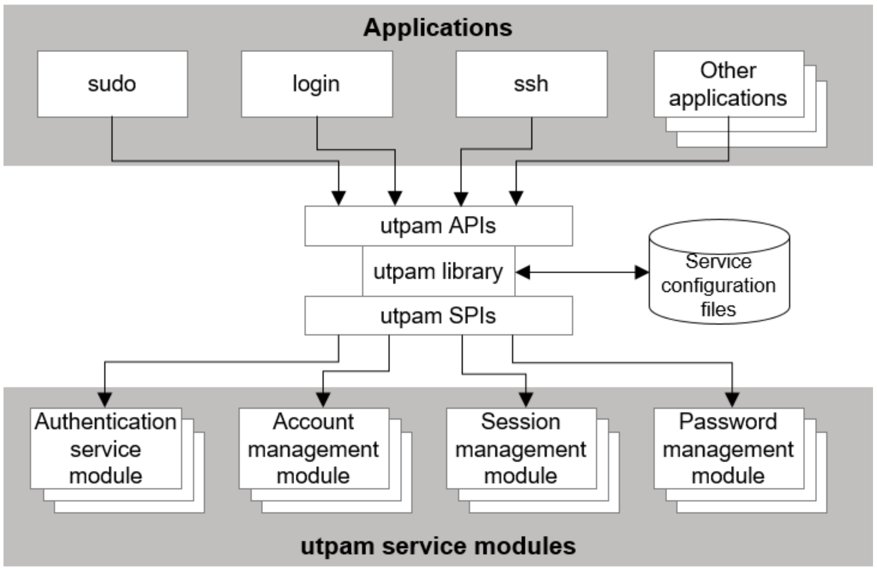
utpam

utpam is a Linux authentication framework. It allows system administrators to define and combine authentication methods for various services across the system as needed. By providing a unified interface, utpam streamlines user authentication for applications.

Feature Description

libutpam is the core library of utpam. It initializes authentication based on configuration files provided by the application. By default, these files are located in **/etc/utpam** and named after their respective services. utpam provides two types of interfaces:

- **APIs (northbound)**: Enable applications to select from four authentication types: user authentication, account management, session management, and password management.
- **SPIs (southbound)**: Allow developers to implement the specific authentication logic corresponding to each API.


Applications
sudo
login
ssh
Other applications
utpam APIs
utpam library
utpam SPIs
Service configuration files
Authentication service module
Account management module
Session management module
Password management module
utpam service modules

Application Scenarios

utpam is a Rust-based extension suite for Linux PAM. It provides pluggable authentication modules and libraries that can be loaded on demand during login, authorization, and session management. Typical scenarios include:

- **Security hardening**: Enforces strict validation or default denials at specific stages to block non-compliant authentication paths.
- **Privilege control**: Implements fast-track approvals or extra checks for root and high-privilege accounts to balance security with operational efficiency.
- **Audit and prompts**: Outputs contextual data (user, TTY, service, source) during the authentication process to improve observability.
- **Heterogeneous integration**: Accesses conventional PAM/C systems via stable C interfaces and headers while leveraging the safety of Rust shared libraries.

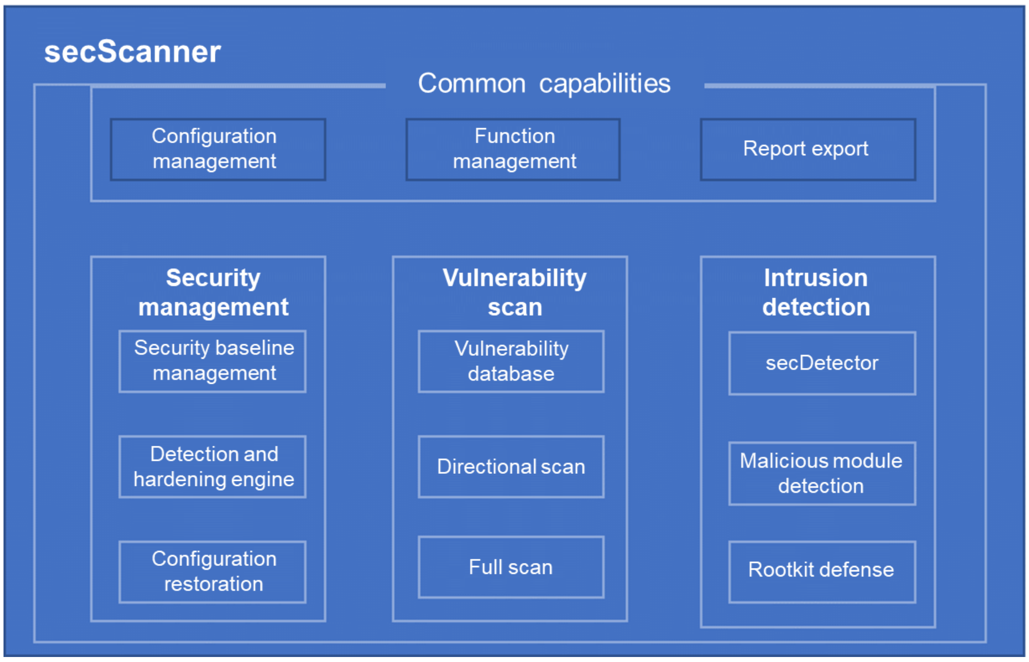
utpam is designed for teams and systems requiring granular control over authentication stages, enhanced auditing, and specialized policies for privileged accounts. It is also ideal for those looking to introduce reliable Rust implementations into existing PAM/C environments.

secScanner

secScanner is a Linux security scanner that provides system management, vulnerability scan, and rootkit detection to keep your OS protected.

Feature Description

Built on a "3 core + 3 common capabilities" architecture, secScanner provides comprehensive, automated, and flexible proactive protection.



- **Security management**: Addresses complex configurations and varying security needs across different scenarios. It supports multiple or custom security baselines with fine-grained parameter control. Key features include one-click security detection, system hardening, and configuration restoration to ensure robust protection.
- **Vulnerability scan**: Provides comprehensive and timely protection against known CVE threats in OS components. It downloads security advisories from openEuler Security Center to maintain an up-to-date local vulnerability database. It supports both full system scans and lightweight, targeted scans of key components. It identifies high-risk areas and provides specific upgrade recommendations to block potential attack paths.
- **Intrusion detection**: Provides generalized protection against unknown threats. To defend against advanced intrusion methods, it utilizes secDetector to identify potential malicious rootkit modules and offers specific cleanup recommendations to ensure system security.

Application Scenarios

secScanner is primarily designed for server security O&M. It enables automated security O&M by detecting weak configurations with a single click and hardening them based on preset baselines. By automating these tasks, it minimizes risks caused by human error, reduces manual workload, and ensures robust system security.

VMAnalyzer

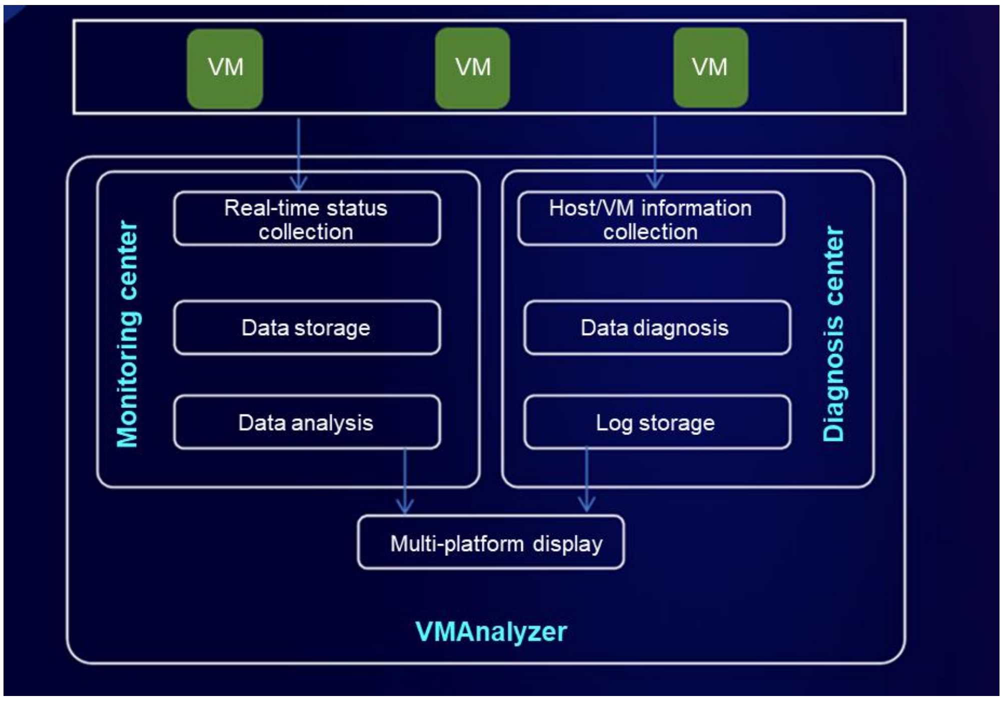
VMAnalyzer is a lightweight virtualization monitoring and analysis tool designed with two core objectives:

- **Virtualization environment monitoring**: real-time tracking of CPUs, memory, and drive I/Os to identify performance bottlenecks.
- **High reliability assurance**: analyzing QEMU processes and physical configurations to predict risks and deliver proactive maintenance policies for failure prevention.

Feature Description

VMAnalyzer consists of two primary modules: the monitoring center and the diagnostic center.

- **Monitoring center**: Provides real-time data collection and granular analysis of VM health. It offers flexible visualization of performance metrics across all cloud hosts via integration with platforms such as the Console, OPS, and DW.
- **Diagnosis center**: Conducts deep-dive diagnostics of the virtualization layer. By executing targeted commands to analyze configurations, system status, and VM resource consumption, it identifies complex issues and exports detailed findings into comprehensive log files.



Application Scenarios

VMAnalyzer is optimized for cloud computing environments. It provides granular insights into VM health and performance, enabling you to easily identify bottlenecks. By streamlining maintenance, VMAnalyzer ensures that VMs operate with high performance and maximum reliability.

9 Copyright Statement

10 Trademarks

All trademarks and logos used and displayed on this document are all owned by the openEuler community, except for trademarks, logos, and trade names that are owned by other parties. Without the written permission of the openEuler community or other parties, any content in this document shall not be deemed as granting the permission or right to use any of the aforementioned trademarks and logos by implication, no objection, or other means. Without prior written consent, no one is allowed to use the name, trademark, or logo of the openEuler community in any form.

11 Appendixes

Appendix 1: Setting Up the Development Environment

Environment Setup	URL
Downloading and installing openEuler	https://www.openeuler.org/en/
Preparing the development environment	https://gitee.com/openeuler/community/blob/master/en/contributors/prepare-environment.md
Building a software package	https://gitee.com/openeuler/community/blob/master/en/contributors/package-install.md

Appendix 2: Security Handling Process and Disclosure

Security Issue Disclosure	URL
Security handling process	https://gitee.com/openeuler/security-committee/blob/master/docs/en/vulnerability-management-process/security-process-en.md
Security disclosure	https://gitee.com/openeuler/security-committee/blob/master/docs/en/vulnerability-management-process/security-disclosure-en.md
Security strategy overview	https://gitee.com/openeuler/security-committee/blob/master/docs/en/vulnerability-management-process/security-strategy-overview-en.md