UnifiedBus™ (UB) Service Core Software Architecture Reference Design

Revision: 2.0

Release Date: 2025-12-31

Contents

Figures

1 Introduction

1.1 Purpose

This document describes the architecture, software function modules, and Application Programming Interface (API) functions of UnifiedBus™ (UB) Service Core for intelligent and general-purpose computing scenarios.

It is intended for:

- Software development engineers
- Technical support personnel
- Enterprise technical owners
- Other personnel interested in the software architecture reference design of UB Service Core for UB product development, usage, management, and maintenance

1.2 Scope

This document describes the UB Service Core software stack, functional components at each layer, and interconnection guidelines for intelligent and general-purpose computing scenarios, including:

- Software architecture: Describes the layered functions of the software stack.
- Functional components: Describes the positioning, architecture, functions, application scenarios, and usage guidelines for functional components at each layer of the software stack.

1.3 Normative Reference

UnifiedBus™ (UB) Base Specification Revision 2.0

2 Reference Software Architecture of UB Service Core

2.1 Software Architecture of UB Service Core

SuperPoDs constitute the foundational infrastructure of a UB computing system (a computing system powered by UB). By leveraging UB technology, SuperPoDs overcome traditional host constraints to achieve scalable expansion. This architecture accommodates both intelligent and general-purpose computing workloads, yielding substantial performance improvements. It embodies a computing system design tailored to the demands of the AI era.

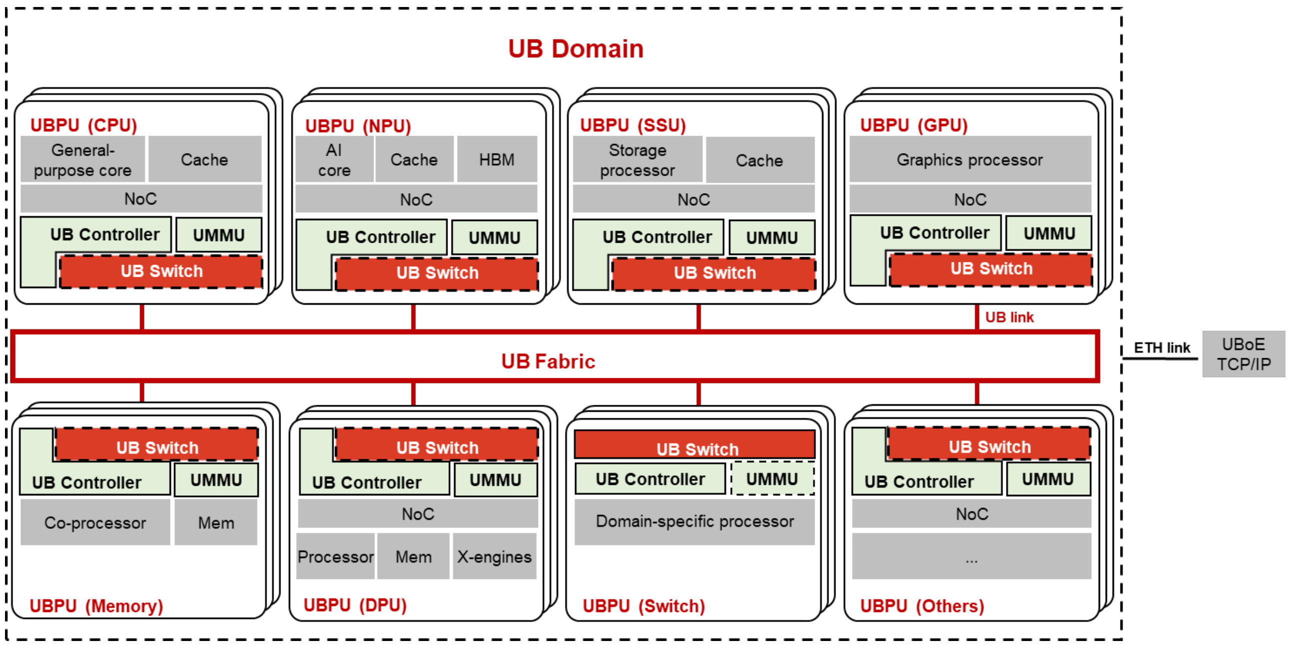

Figure 2-1 Architecture of a UB computing system

UB SuperPoDs redefine host boundaries by introducing a bus-grade, ultra-low latency interconnect and a unified protocol. This allows for flexible resource orchestration, large-scale networking, and high availability (HA) through all-resource pooling and peer-to-peer coordination.

The following figure shows the overall reference architecture of a UB computing system.

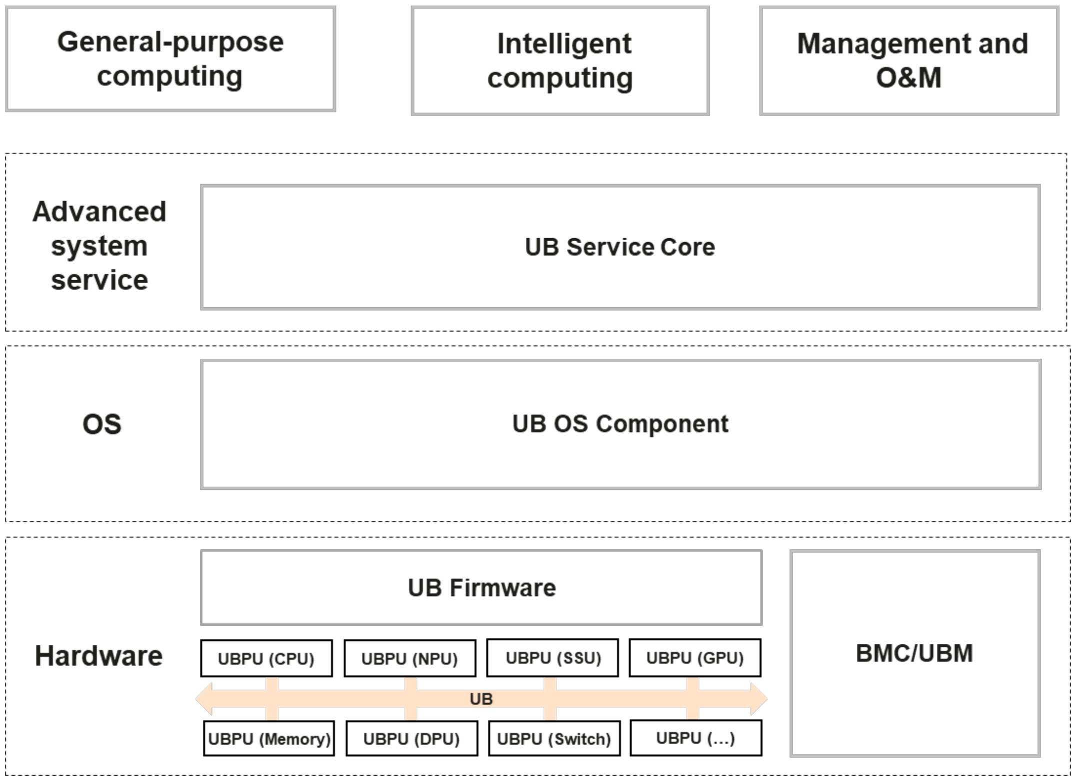

Figure 2-2 UB computing system

The reference architecture of a UB computing system is divided into four layers, which work together to maximize the SuperPoD architecture advantages.

- **UB hardware system: Enables SuperPoD definable computing.**
  - UB Firmware/baseboard management controller (BMC): Serves as the foundation of a UB computing system. It enables bus-grade interconnect, unified protocol, peer-to-peer coordination, all-resource pooling, large-scale networking, and HA for computing clusters based on UB.
  - UnifiedBus Management (UBM): Configures UB interconnect and orchestrates compute resources flexibly and efficiently. It optimizes allocation of resources according to the required SLA (low latency, high bandwidth, and reliability). This enables flexible computing power of UB SuperPoDs.
- **UB OS Component: Provides abstraction and unified management to enable UB interconnect and UB processing unit (UBPU) operation.**
  - Through Linux kernel extensions, UB OS Component natively supports UB computing systems, enabling unified abstraction and management of UB interconnect and UBPUs.
  - Compatible with the POSIX API, UB OS Component enables easy migration of existing applications.
- **UB Service Core: Maximizes SuperPoD performance.**

  UB Service Core maximizes the benefits of a UB computing system in memory pooling, communication, and distributed processing while reducing the complexity of resource management and scheduling. It enables computational business applications to efficiently tap into UB SuperPoDs' capabilities for significant performance improvements.

- **Application adaptation to UB: Unlocks out-of-the-box performance.**

  In key scenarios like intelligent computing and general-purpose computing, Huawei has adapted mainstream open-source applications to run on the UB architecture—covering big data platforms, AI training and inference workloads, and cloud-native Kubernetes environments. By leveraging the SuperPoD architecture strengths of a UB computing system, this adaptation enables out-of-the-box functionality and delivers more than 25% performance improvement across supported applications.

2.2 Components of UB Service Core

UB tightly interconnects multiple nodes to form a SuperPoD. Traditional Linux operating systems (OSs) are designed for standalone machines, scheduling resources (such as memory and PCIe devices) within a single node. To rapidly empower applications with UB SuperPoD capabilities, UB Service Core encapsulates UB's foundational functionalities and cluster topologies. This approach streamlines and maintains compatibility with the existing ecosystem, allowing applications to utilize SuperPoD resources as if they were local, thereby simplifying application development and enabling UB SuperPoDs.

UB Service Core, open-sourced in the openEuler Linux distribution, offers five cluster-level system services and cross-OS compatibility. By harnessing the SuperPoD's peer-to-peer architecture, it boosts application performance by 30%–50% and fosters rapid development across the UB computing system software ecosystem.

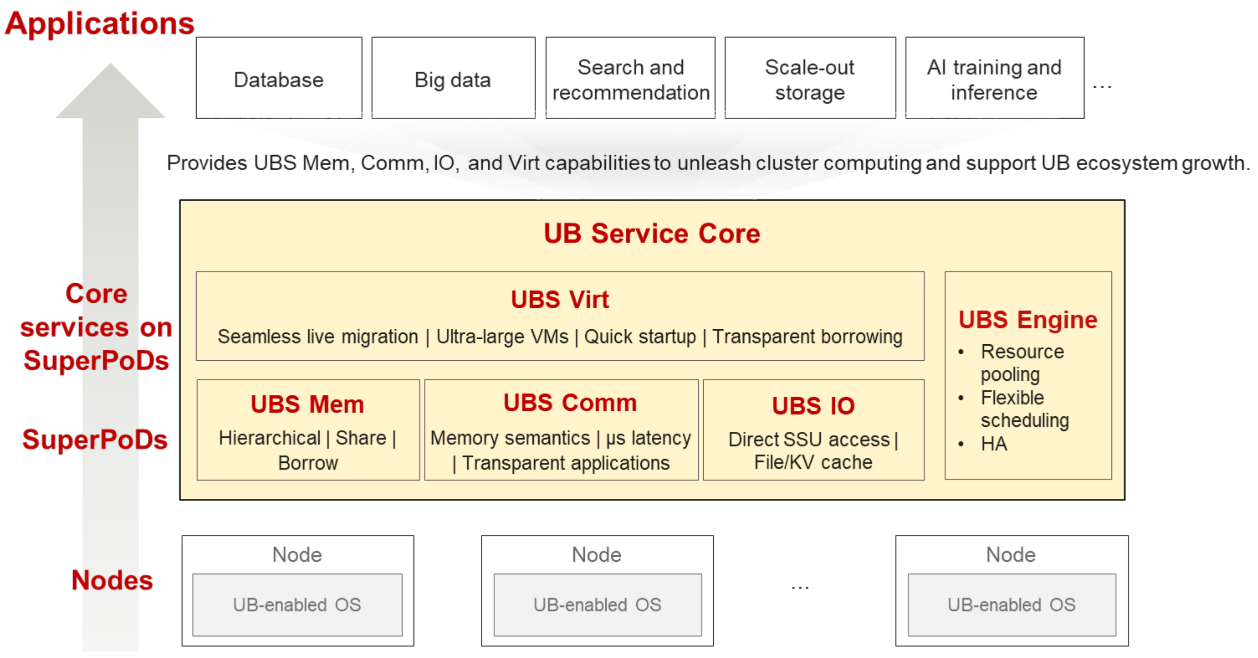
Applications
Database
Big data
Search and recommendation
Scale-out storage
AI training and inference
...
Provides UBS Mem, Comm, IO, and Virt capabilities to unleash cluster computing and support UB ecosystem growth.
Core services on SuperPoDs
UB Service Core
UBS Virt
Seamless live migration | Ultra-large VMs | Quick startup | Transparent borrowing
UBS Engine
Resource pooling
Flexible scheduling
HA
SuperPoDs
UBS Mem
Hierarchical | Share | Borrow
UBS Comm
Memory semantics | μs latency | Transparent applications
UBS IO
Direct SSU access | File/KV cache
Nodes
Node
UB-enabled OS
Node
UB-enabled OS
...
Node
UB-enabled OS

Figure 2-3 Reference architecture of UB Service Core

UB Service Core consists of the following five parts:

- **UB Service Core Engine** (**UBS Engine** for short): Provides resource pooling and dynamic scheduling for memory and DPU resources, supports distributed automatic primary node selection, and implements *N*–1 HA. It serves as the core control-plane reference implementation for the UB computing system.
- **UB Service Core Memory** (**UBS Mem** for short): Supports unified memory semantics programming to implement memory sharing and pooling of UB SuperPoDs.
- **UB Service Core Communication** (**UBS Comm** for short): Provides high-performance, high-reliability, and ecosystem-compatible (user-mode Socket/Verbs over UB) communication protocols based on UB SuperPoDs.
- **UB Service Core IO** (**UBS IO** for short): Provides high-level I/O services for application-affinity global data read/write caching systems (such as SSU passthrough and KV cache) based on UB SuperPoDs.
- **UB Service Core Virt** (**UBS Virt** for short): Supports virtualization and pooling, live migration policy decision, fast recovery and disaster recovery, and accelerated VM-to-VM and container-to-container communication, significantly improving virtualization performance.

3 UBS Engine

3.1 Introduction

As the control engine of a UB SuperPoD, UBS Engine manages pooled resources and schedules them flexibly based on the peer-to-peer architecture. It provides simplified, easy-to-use, and ecosystem-compatible northbound resource management and control APIs to enable UB definable computing.

UBS Engine provides the following core capabilities:

- **Simplified adaptation of cloud management systems and applications**

  Shields complex UB addressing and identification hierarchy such as Entity identifier (EID) and compact network address (CNA), interconnects with UBM and OS (UB OS Component) systems in a unified manner, and provides a single and simplified northbound API for the cluster.
- **Resource pooling, combined with load-aware flexible scheduling**
  - Constructs memory and DPU pooling capabilities and flexibly schedules resources, improving cloud performance and reducing costs.
  - Supports load-aware dynamic memory scheduling between nodes, enhances application performance in high I/O scenarios, and improves resource utilization.
- **Decentralized peer-to-peer architecture, achieving 99.99% HA**
  - Prevents pooled memory from being affected by cross-node faults, ensuring service reliability on SuperPoDs.
  - Uses a decentralized architecture, enabling *N*–1 HA.
- **Open UB system management and control information and APIs for seamless integration with ecosystem applications, ready to use out-of-the-box**

  Supports monitoring and measurement functions compatible with the Prometheus ecosystem and provides open O&M APIs (northbound) for deployment, inspection, and log management.

3.2 Architecture

3.2.1 Software Architecture

The following figure shows the reference architecture of UBS Engine.

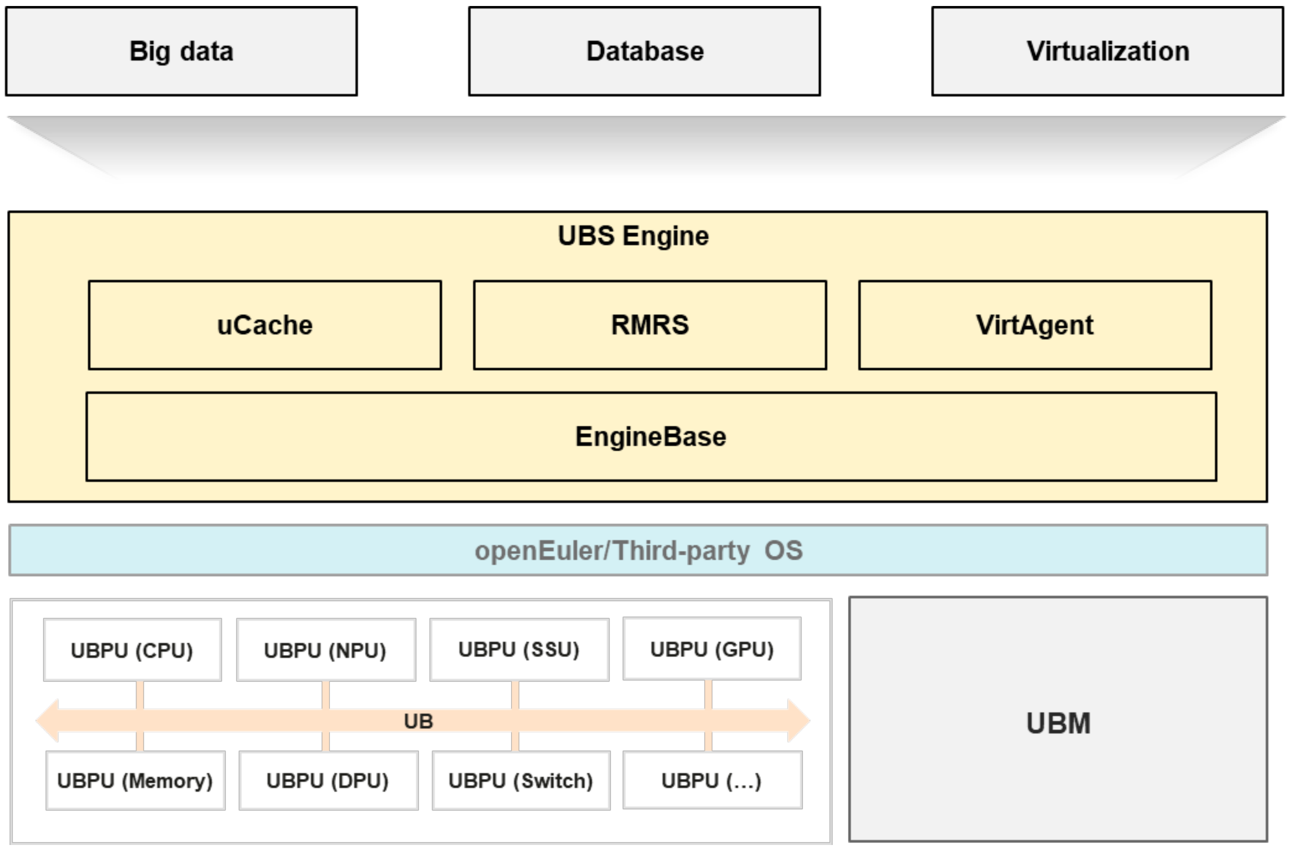

Figure 3-1 Software architecture of UBS Engine

3.2.2 Software Architecture Description

- **EngineBase**: Creates and deletes UB logical resources (such as the bus instance and bonding EID), provides basic memory pooling management capabilities (scheduling, borrowing, returning, and troubleshooting), provides node-level access capabilities for northbound cloud management systems and monitoring systems, and enables decentralized architecture with *N*–1 HA.
- **uCache**: By leveraging memory pooling capabilities of the UB peer-to-peer architecture, creates a UB-powered global I/O cache pool using the strained memory resources of cluster nodes to accelerate I/O-intensive applications.
- **RMRS**: Based on the memory segment distribution of cluster nodes and memory pooling capabilities, reuses the memory segments in the cluster through memory borrowing to start more VMs, improving resource utilization.

- **VirtAgent**: Based on the memory usage of cluster nodes and memory pooling capabilities, borrows memory to balance memory usage, unleash virtualization competitiveness, and improve virtualization resource utilization.

3.3 Application Scenarios

UBS Engine provides memory borrowing and service-based memory optimization capabilities. The typical application scenarios are as follows:

- **Shared memory**: In database and big data scenarios, shared memory is used for primary/standby data synchronization and the UBS Engine API is called via UBS Mem to borrow and release memory.
- **Memory overcommitment in virtualization**: Cloud management systems improve memory utilization through memory overcommitment. UBS Engine automatically borrows memory based on the memory watermark to prevent VMs from exiting due to insufficient memory, improving system reliability.
- **Memory segmentation in virtualization**: UBS Engine identifies hot and cold data, as well as available memory across VMs. It migrates cold data from local VMs to memory segments on remote nodes, thereby freeing local memory. This enables creation of additional VMs and improves overall VM density.
- **Ultra-large VMs**: UBS Engine combines the memory resources of cluster nodes, for example, enabling ultra-large VMs with more than 3 TB of memory.
- **I/O acceleration**: In big data and database scenarios, UBS Engine dynamically reallocates free memory from nodes with surplus capacity to those experiencing memory pressure. This mechanism dynamically expands page cache resources to accelerate I/O operations, thereby improving performance for I/O-intensive applications.
- **Node memory insufficiency**: UBS Engine is triggered during Out-of-Memory (OOM) exception handling to rapidly reallocate memory, preventing application failures caused by node-level OOM events and enhancing overall system reliability.

4 UBS Mem

4.1 Introduction

UBS Mem provides memory sharing, memory data caching, and memory data storage services for applications based on SuperPoDs. The core functions of UBS Mem include:

- **MemFabric**: Provides unified memory addressing and basic capabilities to access data across distributed systems in a SuperPoD.
- **SHMEM**: Provides the memory sharing service in a SuperPoD.
- **MemCache**: Provides the memory data caching service in a SuperPoD, supporting high-bandwidth data access and weak consistency.
- **MemStore**: Provides the memory data storage service in a SuperPoD, supporting low-latency data access as well as strong consistency and reliability.

The following figure shows the position of UBS Mem in the UB system architecture.

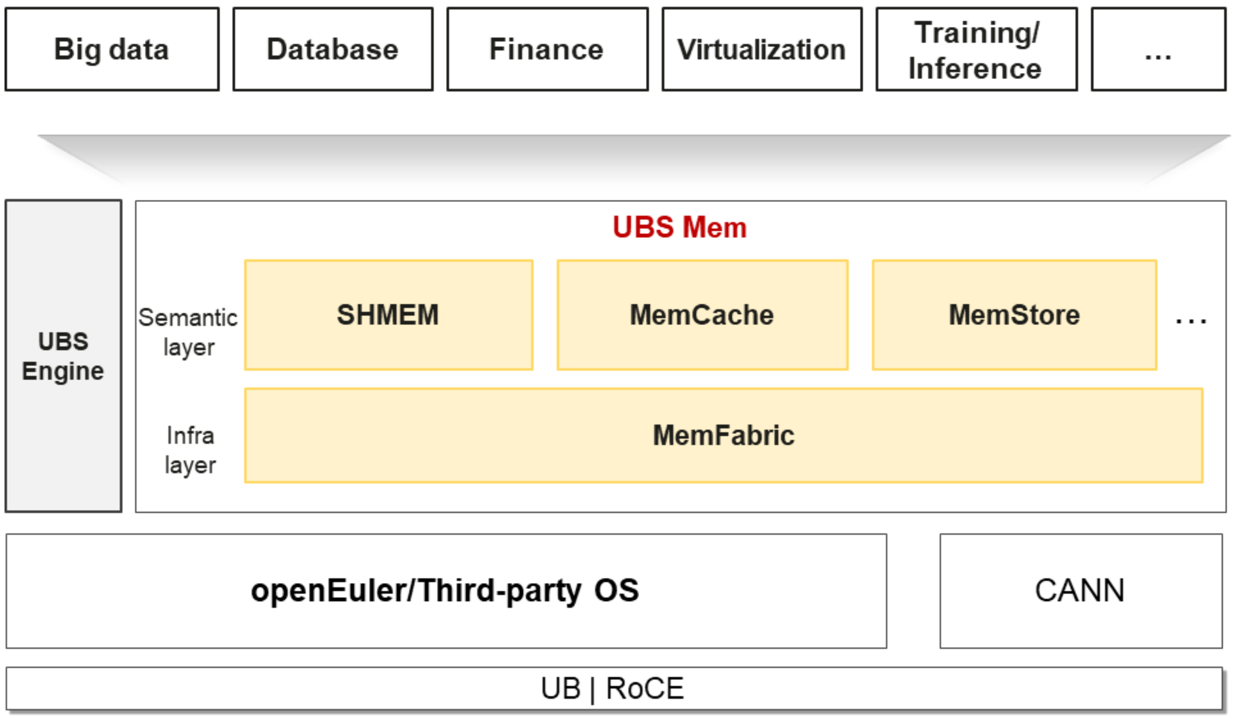

Figure 4-1 Position of UBS Mem

4.2 Architecture

4.2.1 Software Architecture

The UBS Mem software adopts a modular, layered, and decoupled architecture. The following figure shows the reference architecture.

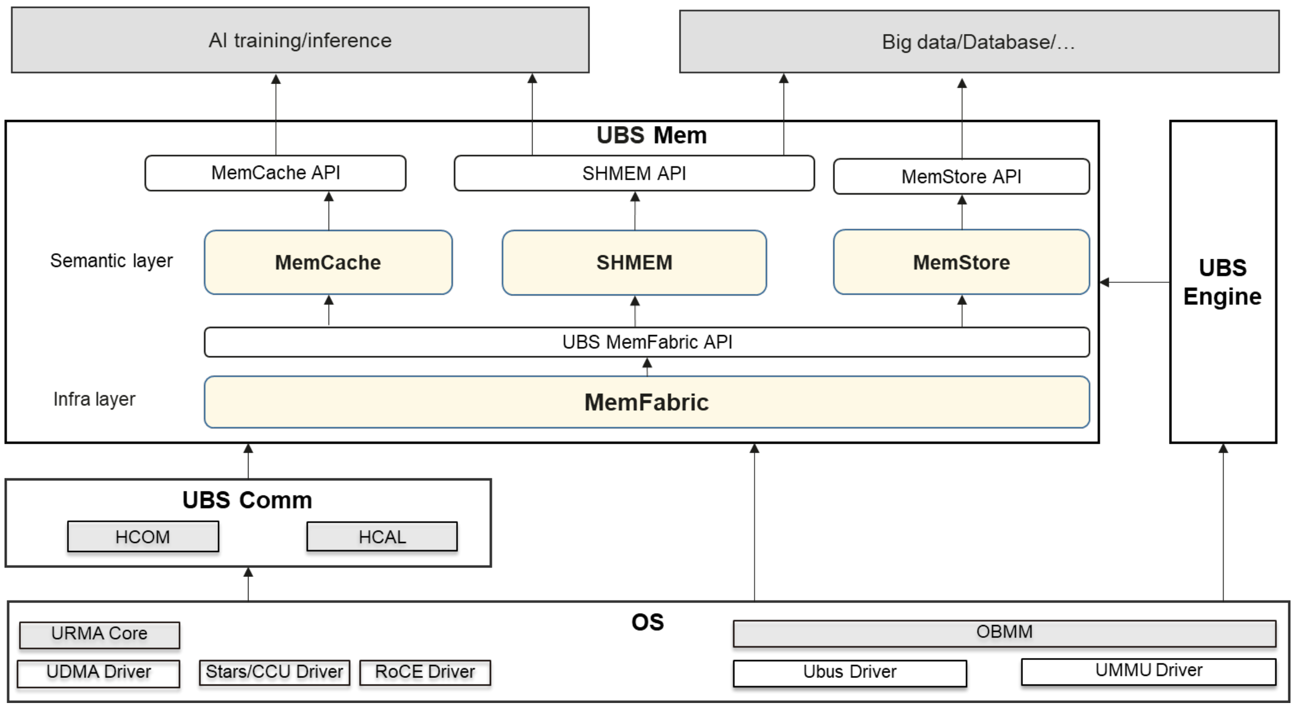

Figure 4-2 Software architecture of UBS Mem

4.2.2 Software Architecture Description

The UBS Mem software architecture is decoupled by service and layer, supporting flexible expansion of functional components. The main functions of each layer are as follows:

- **Semantic layer**: Provides decoupled APIs and services such as MemCache API, SHMEM API, and MemStore API to deliver memory resource sharing, data caching, data storage, and memory semantic communication capabilities for AI inference, big data, database, and finance scenarios.
- **Infra layer**: Shields differences between the semantic layer and hardware, ensures adaptive compatibility across various hardware and OSs, and offers foundational capabilities such as unified memory addressing and data access to support the semantic layer.

4.3 Application Scenarios

The typical application scenarios of MemFabric, SHMEM, MemCache, and MemStore are as follows:

- **MemFabric**: Provides unified memory addressing and data access across distributed systems. It implements unified virtual addressing of memory across hosts based on the UB and RoCE network technology. It implements unified data access based on xDMA technology. Building on these foundations, MemFabric offers specialized semantic libraries to support diverse application scenarios.

- **SHMEM**: Delivers the memory resource sharing service within a SuperPoD. It maintains cross-node cache coherence using UBPU-to-UBPU (such as CPU and NPU) shared memory with supporting software in both intelligent and general-purpose computing scenarios.
- **MemCache**: Uses UB multi-channel concurrency to deliver high bandwidth, supports weakly-consistent memory data caching, and satisfies the high-throughput demands such as KV cache in foundation model inference.
- **MemStore:** Uses UBS Mem C2C communication to implement memory data storage with extremely low latency, ensuring strong consistency across multiple data copies, as well as high reliability. It is well suited for multi-copy memory storage in scenarios such as financial systems.

5 UBS Comm

5.1 Introduction

UBS Comm provides a high-performance, high-reliability, and ecosystem-compatible communication protocol stack for SuperPoDs.

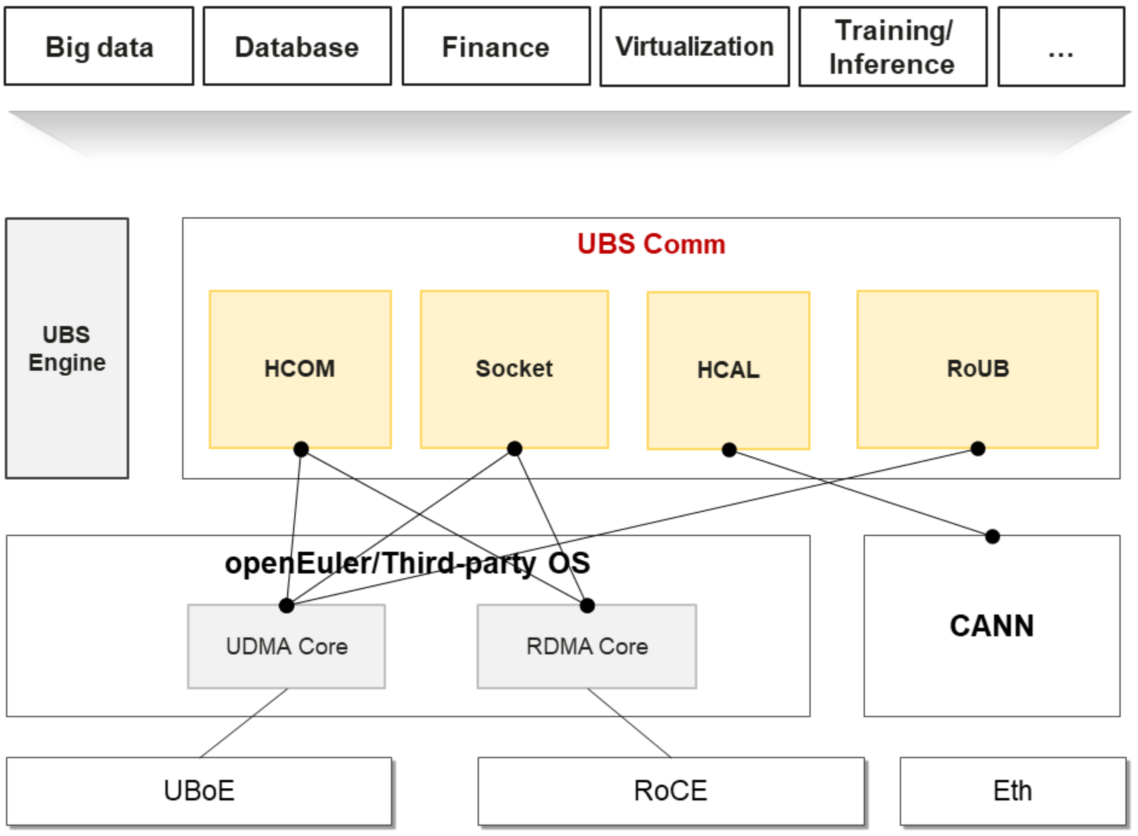

Figure 5-1 Position of UBS Comm

As shown in the preceding figure, UBS Comm provides the following key features:

- **Hyper Communication Library (HCOM)**: Provides a unified UB-native API for connections (northbound) and basic communication libraries that support various protocols (southbound). This simplifies programming and enables reliable, high-performance communication.
- **Socket**: Enables seamless integration without requiring application changes by bypassing the Transmission Control Protocol (TCP)/IP protocol stack at the data plane. With Socket-over-UB technology, it achieves application transparency and enhances performance.
- **HCAL**: Provides cross-node heterogeneous communication capabilities (such as D2H) and supports direct data transmission between CPUs and UBPUs, improving heterogeneous access efficiency.
- **RoUB**: Supports the RDMA Verbs semantic API, allowing seamless migration of existing RDMA applications to the UB network.

5.2 Architecture

5.2.1 Software Architecture

The UBS Comm software adopts a modular, layered, and decoupled architecture. The following figure shows the reference architecture.

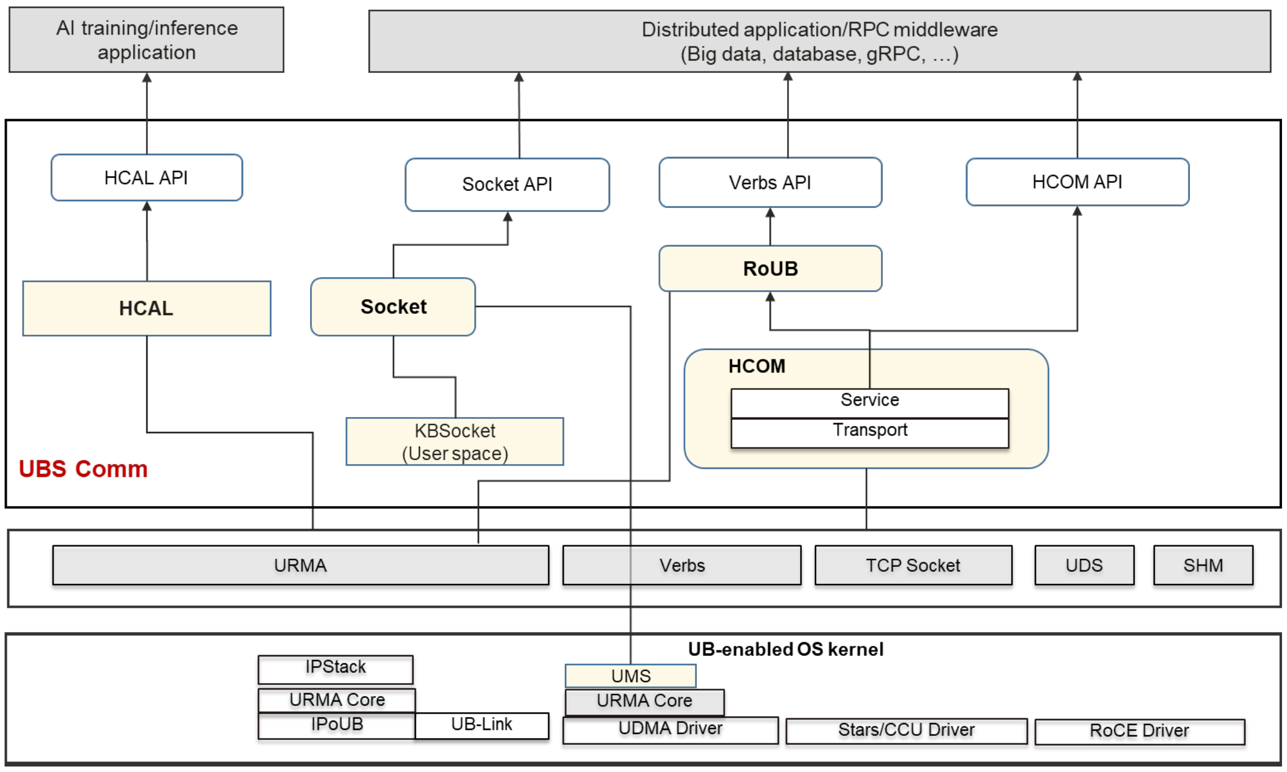

Figure 5-2 Software architecture of UBS Comm

5.2.2 Software Architecture Description

The UBS Comm software architecture is layered and decoupled, supporting flexible expansion of functional components. The main functions of each layer are as follows:

- **Northbound API layer**: Provides mainstream protocol APIs in the industry, including compatible socket APIs and UB-native APIs.
- **Southbound adaptation layer**: Shields the differences between the application layer and hardware, provides dynamic adaptation capabilities, and supports compatibility.

5.3 Application Scenarios

The typical application scenarios of HCOM, Socket, HCAL, and RoUB are as follows:

- **HCOM**: Serves as a multi-protocol, high-performance communication framework applicable to high-bandwidth and low-latency network applications using the client/server (C/S) architecture. It provides a set of advanced APIs that support various protocols, shields the complexity and differences of low-level APIs such as Remote Direct Memory Access

(RDMA), TCP, Unified Remote Memory Access (URMA), and shared memory (SHM), and maximizes the hardware capabilities to ensure high performance.

- **Socket**: Enables seamless access of customer applications to SuperPoDs through socket compatibility, improving end-to-end application performance.
- **HCAL**: Improves the KV cache communication performance in AI inference scenarios through direct communication between UBPUs (NPUs/SSUs/DPUs/GPUs/…) and cross-server CPUs. It can also be accessed as a plug-in to the ecosystem middleware NIXL and Transfer Engine.
- **RoUB**: Converts UB network APIs into Verbs APIs, enabling UB NICs to seamlessly integrate traditional RDMA applications into SuperPoDs.

6 UBS IO

6.1 Introduction

UBS IO implements global read/write caching based on UB, shortening I/O processing paths and improving end-to-end service performance. The following figure shows the position of UBS IO in the UB system architecture.

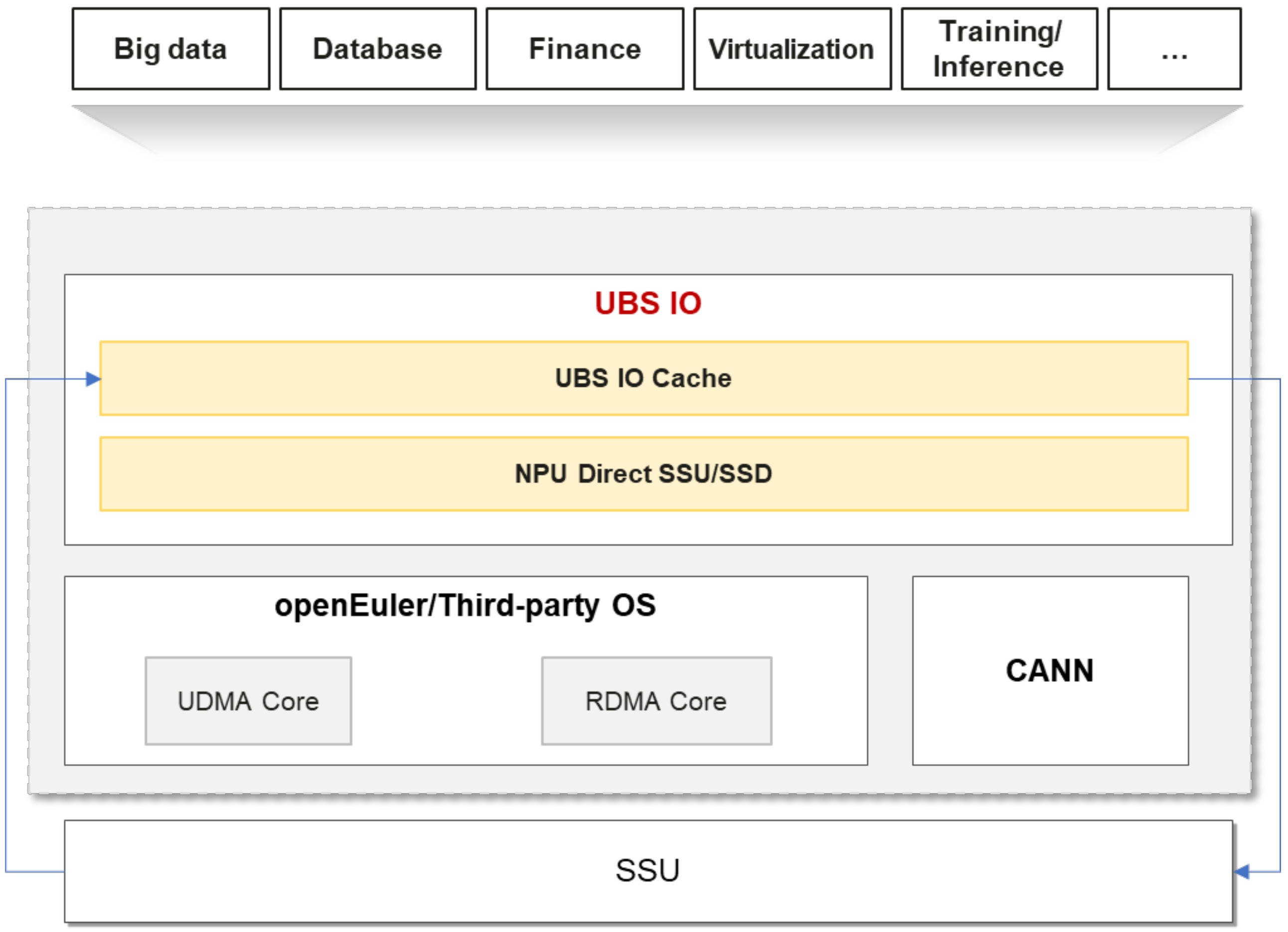

Figure 6-1 Position of UBS IO

UBS IO provides the following functions:

- **UBS IO Cache**: Implements application-affinity global data read/write cache based on SuperPoDs, minimizes the I/O access path, and improves the performance of various applications.
- **NPU Direct SSU/SSD**: Bypasses memory and directly transmits data in the memory to the local SSUs/SSDs, improving the bandwidth utilization of the memory, accelerating data transmission between the high-bandwidth memory (HBM) and SSUs/SSDs, and improving inference performance.

6.2 Architecture

6.2.1 Software Architecture

The following figure shows the software architecture of UBS IO.

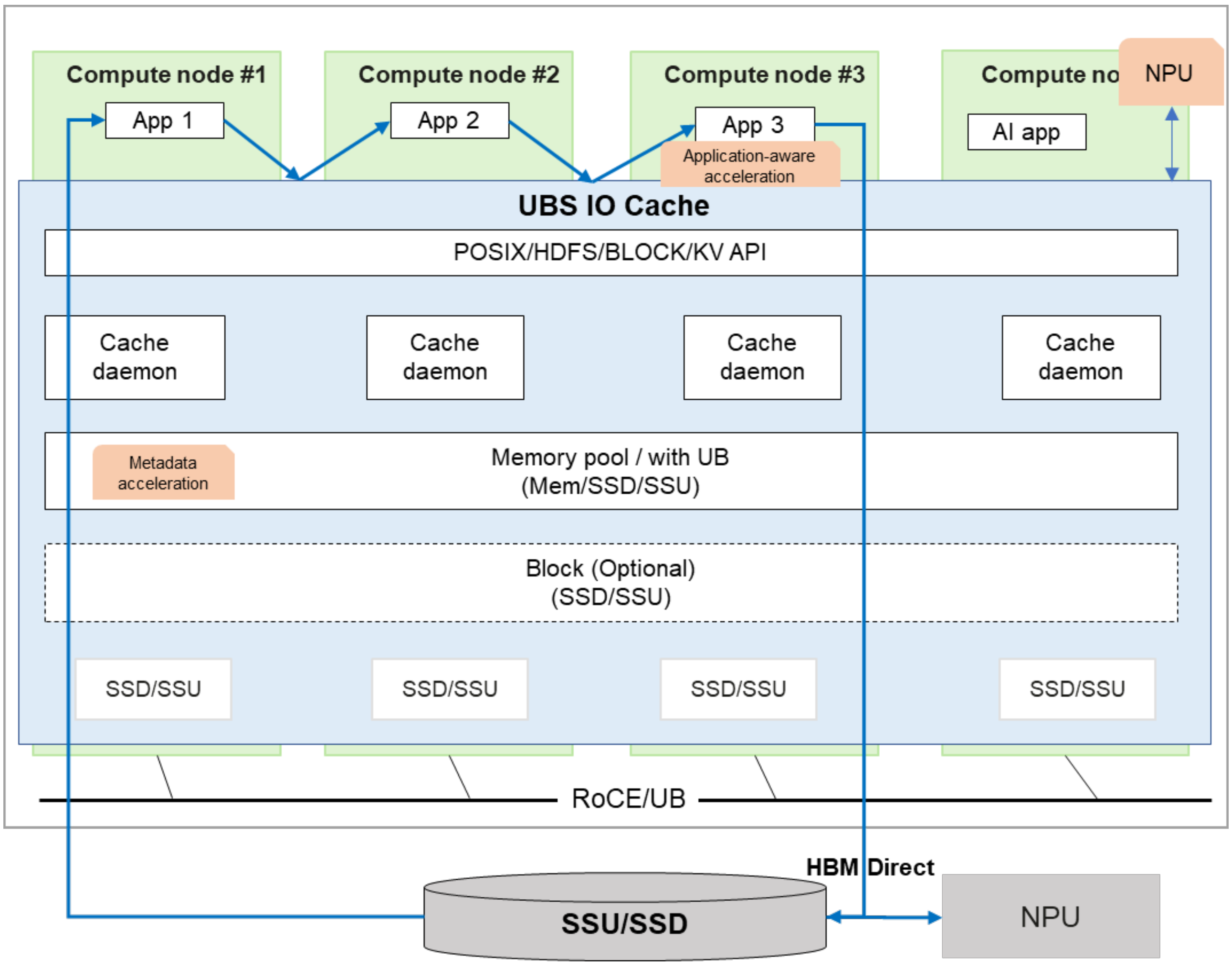

Figure 6-2 Software architecture of UBS IO

6.2.2 Software Architecture Description

As shown in the preceding figure, UBS IO consists of the following function modules:

- **Application-aware acceleration**: Identifies the data access pattern of applications and prefetches or evicts data based on the pattern.
- **Multi-protocol support**: Supports protocols such as POSIX, HDFS, and SCSI to improve the application scope of UBS IO.
- **Cache daemon**: Manages and accesses cache data, including cache space management, data access, and cross-node data retrieval.
- **Metadata acceleration**: Utilizes capabilities of UBPUs such as NICs or DPUs to accelerate I/O metadata access and to improve the data processing efficiency in UBS IO.

- **HBM Direct**: Implements passthrough access from the HBM to the storage media (SSUs/SSDs) or storage systems (such as OceanDisk), accelerating data access efficiency.

6.3 Application Scenarios

The typical application scenarios of UBS IO are as follows:

- **Spark/Hive I/O acceleration**: Optimizes state data storage logic in Spark and Hive to enhance end-to-end application performance, particularly in decoupled storage–compute environments where data read latency becomes a performance bottleneck.
- **Checkpoint acceleration for foundation model training**: Efficient checkpoint saving and loading are critical for training continuity and fault recovery in foundation models. UBS IO accelerates checkpoint operations by caching checkpoint data during training, significantly reducing latency of data saving. Additionally, prefetching checkpoint data into UBS IO shortens recovery time, thereby improving overall training efficiency.
- **KV cache acceleration for foundation model inference**: Access performance of the KV cache is a key factor for inference performance. By offloading data from HBM to local or remote storage systems, UBS IO enhances access bandwidth to the KV cache. This bandwidth optimization directly contributes to higher overall inference performance.
- **HPC data cache acceleration**: High-performance computing (HPC) workloads often generate a large volume of small files that will be frequently accessed (read/written). UBS IO caches these small files and leverages the high-speed interconnect between compute nodes to enable fast access, significantly improving overall computational efficiency.

7 UBS Virt

7.1 Introduction

As a suite for virtualization scenarios, UBS Virt provides VMs, bare metals, and containers to enable the architecture performance of a UB computing system and supports both intelligent and general-purpose computing scenarios.

Its core functions include:

- **UB virtual network enablement:** Extends the data plane and control plane of the infrastructure network based on UB. It leverages peer pooling for flexible resource allocation, enhances system-level performance, and enables UB virtual network.
- **UB super virtualization enablement**: Enables super VMs and containers on SuperPoDs, and provides functions such as cross-node resource pooling, scheduling, and decision-making.
- **Virtualization acceleration**: Provides a UBPU virtualization acceleration suite.

7.2 Architecture

7.2.1 Software Architecture

The following figure shows the logical architecture of UBS Virt.

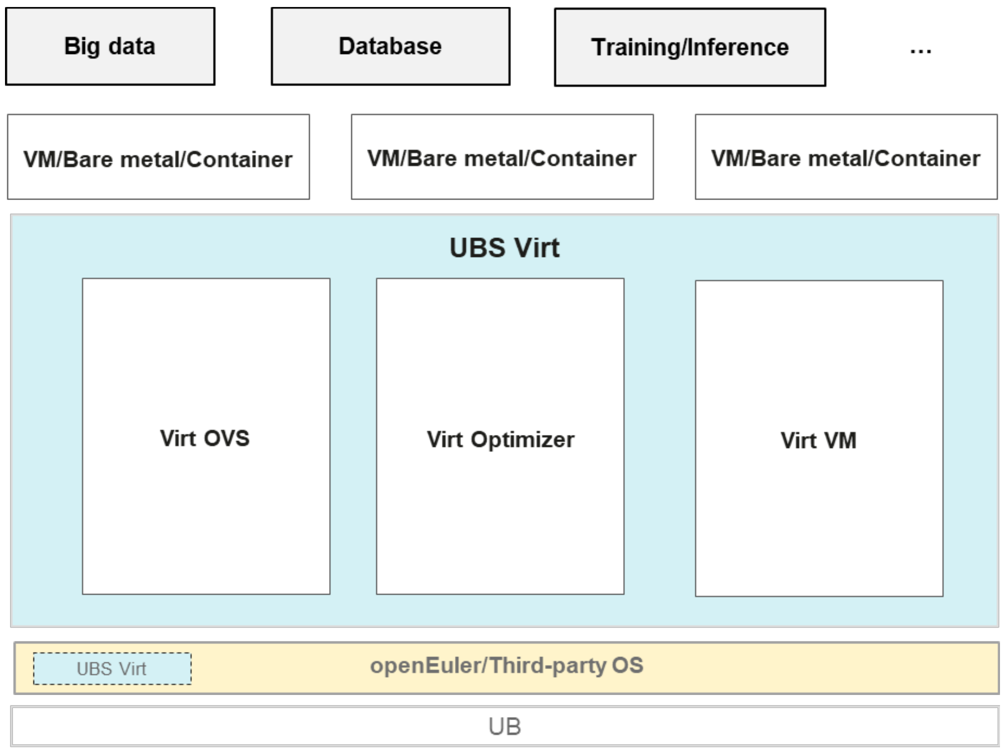

Figure 7-1 Software architecture of UBS Virt

As shown in the preceding figure, UBS Virt provides the following key features:

- **Virt Open Virtual Switch (OVS)**: Enables UB network virtualization capabilities based on the SuperPoD architecture, supporting high-performance virtual networks, multi-tenant isolation, and QoS control.
- **Virt Optimizer**: Provides virtualization acceleration suites and performance tuning tools in intelligent computing scenarios, collaborating with the host, guest, hypervisor, Compute Architecture for Neural Networks (CANN) drivers, and other modules to maximize system performance.
- **Virt VM**: Transcends the boundaries of a single server to offer ultra-large VMs by leveraging UB's bus-grade pooling.

7.2.2 Software Architecture Description

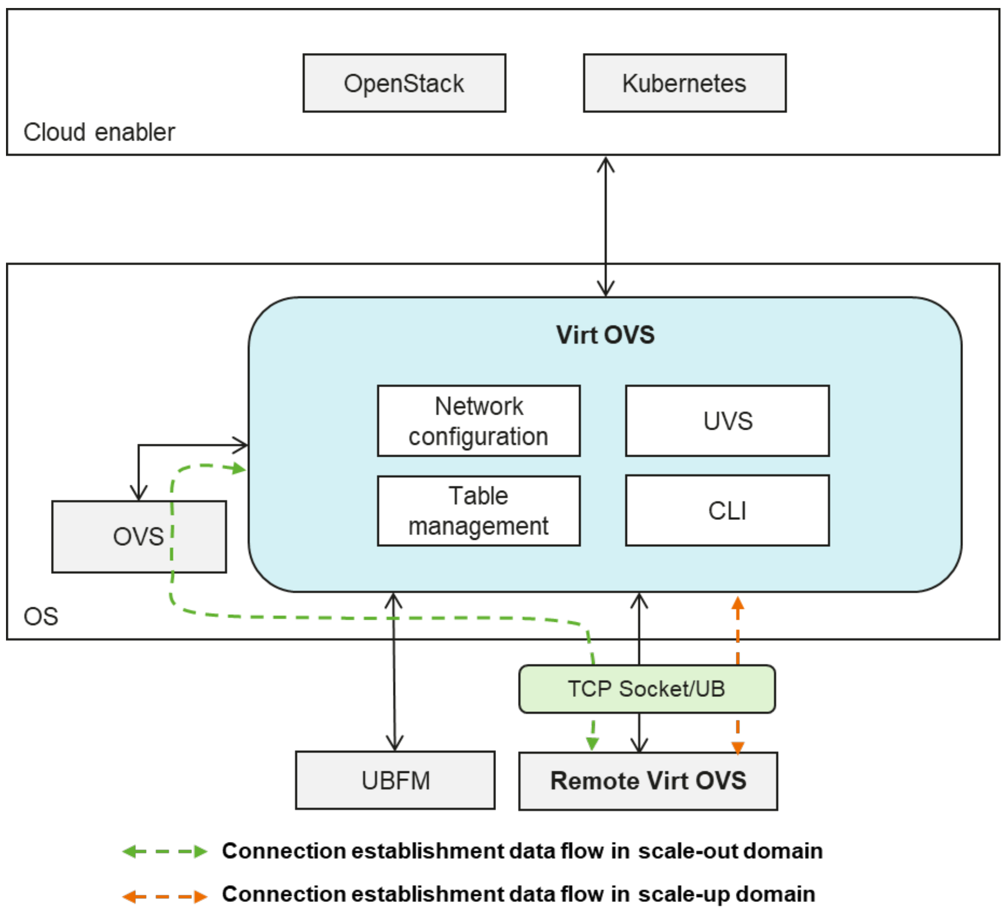

Figure 7-2 Software architecture of Virt OVS

Virt OVS provides the following functions:

- **Network configuration**: Supports UB network configuration and manages attributes across the transaction and transport layers—including EID, UPI, TP, and priority settings—to enable advanced network capabilities such as multi-tenant isolation and QoS control.

- **Unified Virtual Switch (UVS)**: Handles negotiation and establishment of transport layer channels to enable network virtualization.
- **Table management**: Provides persistent storage for Source IP (SIP), Destination IP (DIP), and vTP entries, ensuring data recovery and consistency in the event of component restarts or failures.
- **Command-line Interface (CLI)**: Provides a CLI tool for cloud management and virtualization administrators to configure and manage system resources efficiently.

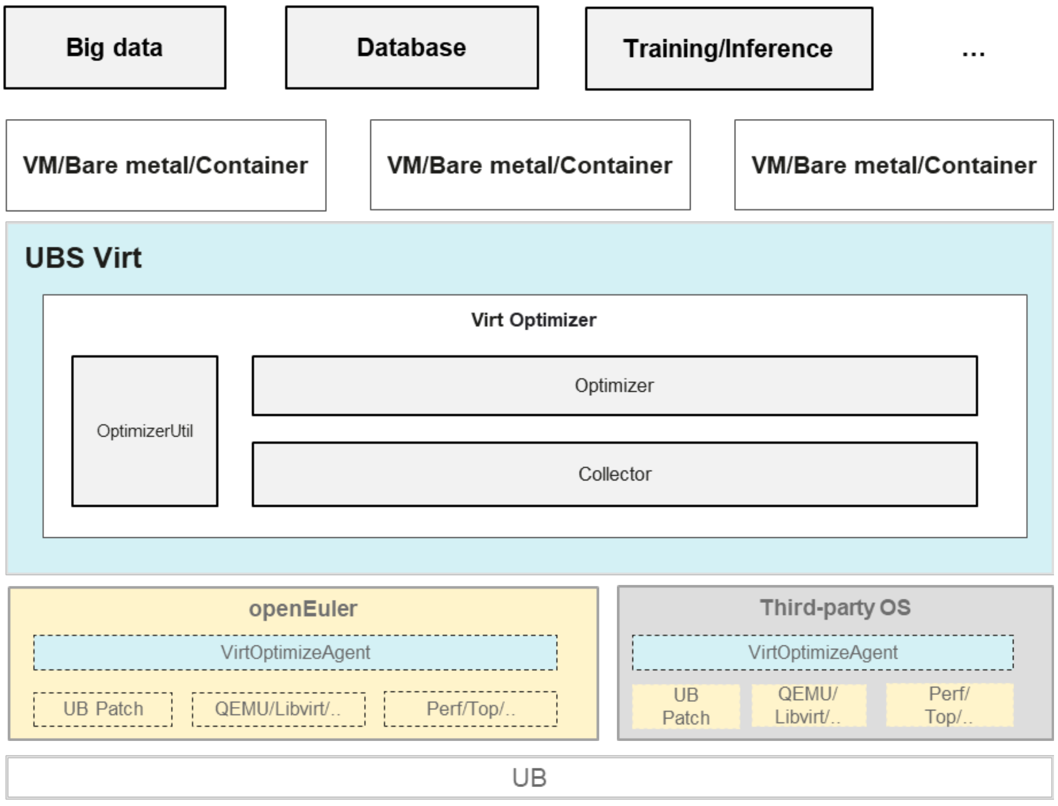

Figure 7-3 Software architecture of Virt Optimizer

Virt Optimizer provides the following functions:

- **Virt Optimizer**: NPU acceleration library, which accelerates NPU virtualization performance, including computing, storage, and network.
- **VirtOptimizeAgent**: Virtualization tuning tool that automatically detects and tunes hosts and guests. The HDK virtualization driver in a guest works with the hypervisor to accelerate virtualization.

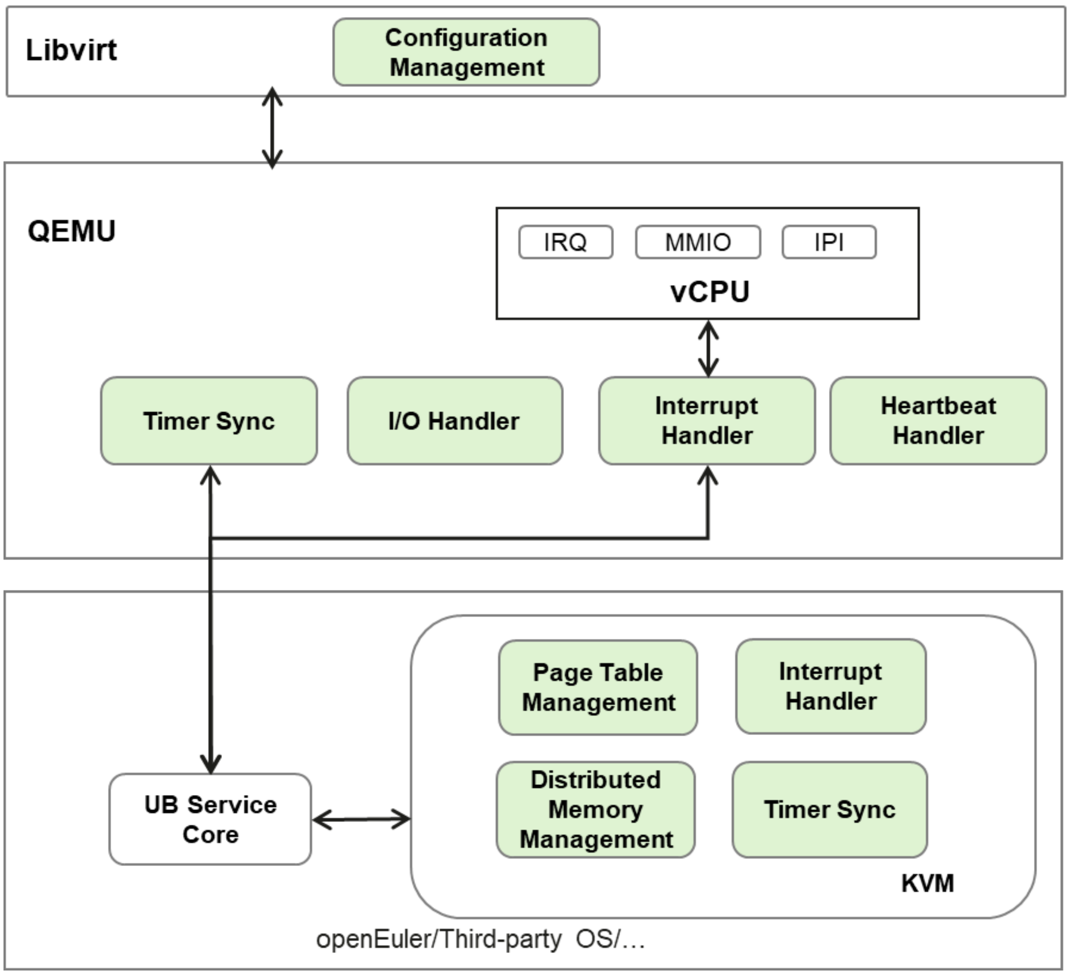

Figure 7-4 Software architecture of Virt VM

Virt VM provides the following functions:

- **Configuration Management**: Parses the SuperPoD topology information delivered by cloud management systems.
- **Timer Sync (QEMU)**: Ensures time synchronization of guests in a SuperPoD.
- **I/O Handler (QEMU)**: Handles I/O interrupts and access requests for QEMU devices in a SuperPoD.
- **Interrupt Handler (QEMU)**: Forwards virtual Generic Interrupt Controller (vGIC) interrupts for QEMU devices in a SuperPoD.
- **Heartbeat Handler (QEMU)**: Detects UB link and QEMU status in a SuperPoD.
- **Page Table Management (KVM-page table management module)**: Manages GPA/HVA mapping in a SuperPoD.
- **Interrupt Handler (KVM-interrupt forwarding module)**: Forwards MSIX interrupts in a SuperPoD.
- **Distributed Memory Management (KVM-distributed memory management module)**: Synchronizes host OS memory status (such as share, modify, and pin) in a SuperPoD.
- **Timer Sync (KVM-clock synchronization module)**: Synchronizes host OS clocks in a SuperPoD.

7.3 Application Scenarios

Typical application scenarios of UBS Virt are as follows:

- **Data plane**: Extends the data plane and control plane of the infrastructure network based on SuperPoDs. It leverages pooling capabilities of the UB peer-to-peer architecture for flexible network resource allocation, enhances system-level performance, and enables UB virtual networks.
- **VM**: Supports ultra-large VMs, memory borrowing, ultra-fast live migration, and automatic detection and optimization of UBPU virtualization performance.
- **Container**: Supports container live migration and quick startup.

Appendix A Acronyms and Abbreviations

Acronym or Abbreviation	Full Name
API	Application Programming Interface
BMC	baseboard management controller
CANN	Compute Architecture for Neural Networks
CNA	compact network address
DIP	Destination IP
EID	Entity identifier
HCAL	Heterogeneous Communication Acceleration Library
HCOM	Hyper Communication Library
RDMA	Remote Direct Memory Access
RoCE	Remote Direct Memory Access over Converged Ethernet
SIP	Source IP
SHM	Shared Memory
TCP	Transmission Control Protocol
TP	transport
UBM	UnifiedBus Management
UBPU	UB processing unit
URPC	unified remote procedure call
URMA	unified remote memory access