Version: NexaVM nSSV 1.10.28

Issue: V1.10.28

Copyright Statement

Trademark

Notice

The products, services, or features that you purchased are all subject to the commercial contract and terms of NexaVM Technologies AG., but any part or all of the foregoing

displayed in this document may not be in the scope of your purchase or use. Unless there are additional conventions, NexaVM Technologies AG. will disclaim any statement or warranty, whether implicit or explicit, on the contents of this document.

In an event of product version upgrades or other reasons, the contents of this document will be irregularly updated and released. Unless there are additional conventions, this document, considered solely as a reference guide, will not make any warranty, whether implicit or explicit, on all the statements, information, or suggestions.

Contents

1 Overview

Industry Background

Data center virtualization has become an industry trend in the enterprise IT landscape. As IT infrastructure continues to scale, the complexity of management increases significantly. By enabling dynamic scheduling and allocation of IT infrastructure resources, next-generation data centers leveraging virtualization technologies can effectively address multiple challenges inherent in traditional IT architecture, such as low resource utilization, heightened data security risks, complex system maintenance, and prolonged service deployment cycles. However, with the growing number of high-performance applications, effectively mitigating challenges related to performance, security, stability, and compatibility has become a critical focus for the evolution of virtualization technologies and a fundamental requirement for platform selection.

About NexaVM NSSV

Based on the core principle of high performance, high security, high reliability, and high independence, NexaVM NSSV integrates the powerful NexaVM Cloud engine with 4S characteristics. Leveraging cutting-edge technologies such as server virtualization, network virtualization, and storage virtualization, it also incorporates intelligent advanced O&M capabilities, positioning itself as a next-generation server virtualization solution. With NexaVM NSSV, you can rapidly build virtualized data centers or deploy integrated solutions spanning from IaaS to PaaS by leveraging NexaVM's diverse products.

Applications

High-Performance Workloads/Database Virtualization Transformation:

As the core of modern applications, databases are responsible for storing, managing, and processing large volumes of data. Their performance and reliability are critical to application stability. NexaVM NSSV virtualization technology provides resource isolation and scalability, enabling more efficient management of hardware resources of database servers. Virtualization supports dynamic allocation and adjustment of server resources, such as memory, CPU, and

storage, to meet the needs of diverse applications. Additionally, NexaVM NSSV delivers flexible backup and recovery capabilities, along with high availability and fault tolerance.

Intelligent Production:

NexaVM NSSV delivers a robust digital foundation that supports the deployment of key business systems such as DMP, ERP, and OA. By integrating next-generation AI technologies with the virtualization platform and advanced manufacturing techniques, it enables predictive management, informatized operations, and data-driven product lifecycle management.

Typical use cases include:

- **Production Testing**: NexaVM NSSV facilitates product testing and simulation by replicating real-world production environments and process flows. This allows for performance prediction and process optimization, reducing the need for physical tests and minimizing trial-and-error costs.
- **Supply Chain Management**: NexaVM NSSV enables supply chain information sharing, real-time demand forecasting, and dynamic resource allocation, significantly improving supply chain responsiveness.
- **Equipment Maintenance**: Through NexaVM NSSV, physical devices can be converted into virtual instances, supporting remote monitoring, fault diagnosis, and maintenance. This enhances equipment availability while reducing maintenance costs.
- **Training and Education**: NexaVM NSSV allows the creation of virtual training environments that simulate real production scenarios and operational procedures. This helps employees acquire and master production skills, leading to higher efficiency and quality.

Production Environment Resource Hosting:

Traditional production environments often face challenges such as complex configuration and deployment, environmental conflicts, physical resource constraints, data security risks, and difficult troubleshooting. NexaVM NSSV effectively addresses these issues through the following capabilities:

- **Rapid Deployment and Environment Isolation**: NexaVM NSSV enables quick provisioning of virtual machines, streamlining environment setup and configuration. It allows the creation of multiple isolated virtual environments tailored to different development and testing projects, preventing inter-project interference and improving R&D efficiency.
- **Efficient Utilization of Physical Resources**: By pooling and sharing physical server

resources through virtualization, NexaVM NSSV allows multiple virtual machines to run on a single physical server. This improves resource utilization and reduces hardware costs. It also supports dynamic resource allocation to adapt to changing business needs.

- **Business Data Security Protection**: NexaVM NSSV supports on-demand VM snapshot creation, enabling rollback when necessary to safeguard business data and ensure operational stability and reliability.
- **Flexible Development Environments**: NexaVM NSSV provides adaptable development environments for business teams. Each developer can work within their own VM for development and testing, independently managing and configuring their environment to avoid conflicts and enhance both productivity and output quality.
- **Fault Isolation and Fault Tolerance**: NexaVM NSSV ensures that if one virtual machine fails, others continue operating normally. This reduces downtime caused by software or hardware failures and enhances business continuity and stability.

2 Technical Advantages

High Performance

High-performance workloads, serving as a key lever for users to reduce costs and improve efficiency, place greater performance demands on the underlying virtualization platform. In response to these growing requirements, NexaVM NSSV offers the following advantages:

- Exceptional high-concurrency capability, significantly accelerating service deployment
- Minimized computational resource overhead
- Efficient utilization of diverse network resources
- Flexible integration with high-performance storage solutions

High Security

With the continuous improvement of network infrastructure and the increasing migration of data to online environments, ensuring the secure and stable operation of services has become a core priority. NexaVM NSSV inherently provides comprehensive, multi-layered security capabilities:

- **Network Traffic Control**: Distributed firewalls and security groups filter both north-south and east-west traffic, effectively preventing virus infections.

- **Data Security**: Supports the application of commercial cryptography across various industries and sectors, establishing a robust cryptographic barrier for network security.
- **Service Protection**: Features a kernel-level, agentless virtualization security engine that promptly detects and eliminates threats to safeguard business data.

High Reliability

As the core software system of IT infrastructure, NexaVM NSSV serves as a robust foundation for diverse business capabilities, the ultimate safeguard for data center stability, and a fundamental prerequisite for cost efficiency and operational optimization. As a critical component for operations maintenance, a highly stable NexaVM NSSV offers the following advantages:

- Automatic recovery after power failure
- Zero data loss
- Continuous availability in daily operations

3 Product Architecture

3.1 Software Architecture

NexaVM NSSV software architecture highlights:

- Fully Asynchronous Architecture: Asynchronous messaging, methods, and HTTP calls.
  - Asynchronous messaging: Services communicate via a message bus. When calling a service, the source service sends a message to the destination service, registers a callback function, and returns immediately. Upon task completion, the destination service triggers the callback with results. Asynchronous messages can be processed in parallel.
  - Asynchronous methods: Services communicate through asynchronous messages. Components or plugins inside each service use asynchronous method calls, following the same pattern as asynchronous messaging.
  - Asynchronous HTTP calls: Each plugin has an agent that puts a callback URL in the HTTP header of every request. After task completion, the agent sends the response to the caller's URL.
  - Based on these three asynchronous approaches, NexaVM NSSV builds a layered architecture to ensure all components operate asynchronously.

- With this fully asynchronous architecture, a single management node can process tens of thousands of concurrent API requests per second, while managing tens of thousands of hosts and hundreds of thousands of virtual machines.

- Stateless Service: Each request is independent.
  - Compute node agents, storage agents, network services, console proxy services, and configuration services require no inter-request dependencies. Every request carries complete context and no node needs to maintain or store any information.
  - Resources such as management nodes and compute nodes are authenticated via UUID -based consistent hashing ring. Message senders do not need to know specific service instances, and services simply process messages without maintaining or exchanging resource information.
  - Little information is shared among management nodes. Therefore, a minimum of two management nodes can meet the requirements of high availability and scalability.
  - The stateless service mechanism makes the system more robust. Restarting servers will not lose any state information. This also simplifies the scaling out and scaling in of a data center.
- Lock-Free Architecture: Consistent hashing algorithm.
  - The consistent hashing algorithm ensures all messages of the same resource are always processed by the same service instance. This message aggregation to specific nodes reduces synchronization and parallel processing complexity.
  - Work queues replace lock contention. Serial tasks are stored in memory as work queues, which can process any operation on any resource in parallel to improve system parallelism.
  - The queue-based lock-free architecture enables tasks to run in parallel, thereby improving the system performance.
- In-Process Microservices: Decoupled microservices.
  - A message bus isolates and controls services, such as virtual machine services, identity authentication services, snapshot services, volume services, network services, and storage services. All microservices are enclosed in the management node process. These services communicate with each other through the message bus. All messages are first sent to the message bus, then forwarded to the destination service using consistent hashing ring.
  - In-process microservices adopt a star-like architecture where each service runs independen tly. This architecture decouples the highly centralized control business, achieving high

autonomy and isolation. Service failures will not affect other components, ensuring system reliability and stability.

- Versatile Plugin System: Horizontally scalable plugins.
  - Every plugin provides services independently. Any newly added plugin has no impact on other existing plugins.
  - Plugins support both Strategy Pattern and Observer Pattern designs. Strategy plugins inherit parent-class interfaces to implement specific functions. Observer plugins register Listeners to monitor event changes of the internal business logic in an application. When events occur, the observer plugins will automatically respond and trigger corresponding business flow.
  - This horizontal plugin scalability enables NexaVM NSSV to rapidly upgrade while maintaining robust system architecture.
- Workflow Engine: Sequence-based management and error rollback.
  - Every workflow is clearly defined in XML. If error occur at any step, the workflow rolls back along the original executed path and cleans up the garbage resources.
  - Each workflow can contain sub-workflows for extended business logic implementation.
- Tag System: Enables business logic change and resource attributes.
  - Extend or modify business logic dynamically using system tags and plugin mechanisms.
  - Group and categorize resources with tags, enabling resource searches by specific tags.
- Cascade Framework: Supports chained resource operations.
  - The Cascade framework manages resources through waterfall-like chained operations. For example, uninstalling or deleting a resource triggers corresponding cascading operations on related resources.
  - Resources can be added to the Cascade framework as plugins. Their addition or removal does not affect other resources.
  - The cascading mechanism enables flexible and lightweight resource configuration, allowing rapid adaptation to customer configuration changes.
- Automated Deployment: Agentless auto-deployment with Ansible.
  - Leverages agentless Ansible to automate dependency installation, physical resource configuration, and agent deployment. The whole process is transparent to users and requires no additional intervention. You can upgrade your agents simply by reconnecting the agents.

- Comprehensive Query APIs: Access any attribute of any resource.
  - Supports resource queries with millions of conditions, comprehensive query APIs, and any way of condition combinations.

3.2 Functional Architecture

NexaVM NSSV functional architecture highlights:

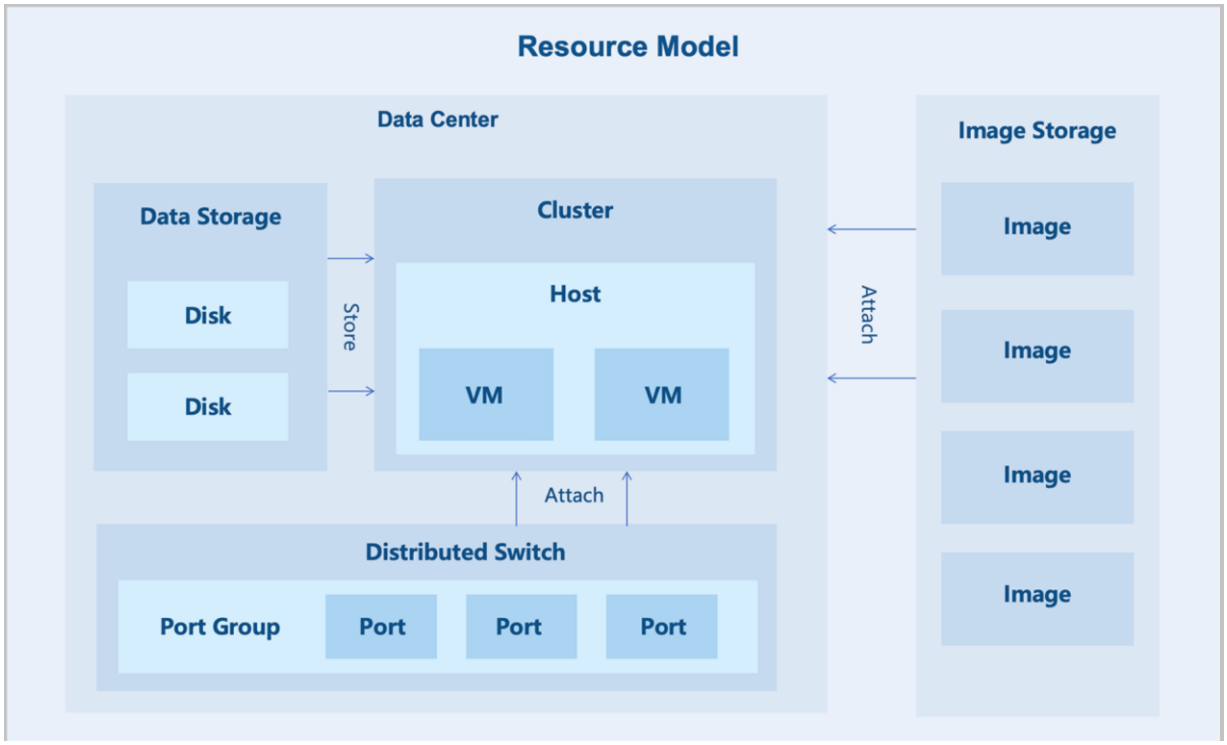
- NexaVM NSSV provides enterprise-grade data center infrastructure services for compute, storage, and network resource management. NexaVM NSSV supports KVM virtualization technology at its foundation. NexaVM NSSV supports DAS, NAS, SAN, and DSS storage types, including local storage, NFS storage, SAN storage, and distributed storage. NexaVM NSSV supports network models like distributed switch and distributed port group.

- With NexaVM NSSV as its core engine, the platform uses a message bus to communicate with MariaDB database and service modules. Key functionalities include virtual machine management, host management, storage scheduling, network services, system administration, monitoring and auditing, and more.
- NexaVM NSSV provides Java and Python SDKs and RESTful APIs for resource scheduling and management.

3.3 Resource Model

Figure 3-2: Resource Model shows the resource structure of NexaVM NSSV.

Figure 3-2: Resource Model



NexaVM NSSV has the following main resources:

- Data Center: A data center is the largest resource namespace within a virtualization platform, including resources such as clusters, hosts, data storage, distributed switches, and distributed port groups.
- Cluster: A logical collection of a group of hosts (compute nodes).
- Host: A host is an x86 or ARM physical server running a KVM virtualization hypervisor, providing resources such as computing, networking, and storage to virtual machines.
- Virtual Machine: A virtual machine is a virtualized host running on a physical host, capable of running an operation system and applications just like a physical host.
- Data Storage: A data storage is a virtualized resource that provides storage space for virtual machines and their application data. A data storage can be categorized into local storage and network shared storage.
- Distributed Switch: A virtual switching device that provides unified virtual network management and monitoring for virtual machines within a cluster.
- Distributed Port Group: A logical grouping of ports on a distributed switch, used for port configuration.
- Image Storage: An image storage is a virtualized resource that provides storage space for image template files used by virtual machines or disks. An image storage can be categorized into standalone image storage and distributed image storage.

- Image: An image is a template file used by virtual machines or disks. Images are categorized into system images and disk images.

NexaVM NSSV resources maintain the following two types of relationships:

- Hierarchical relationships: Similar to interpersonal relationships in human society, including parent-child, sibling, grandparent-grandchild, and peer relationships.

  Definitions:

  - Parent-Child: Resource A is the parent of child of Resource B. For example, clusters and hosts, or hosts and virtual machines. In both cases, the latter resource runs within the former.
  - Sibling: Resource A and Resource B share the same parent. For example, clusters and distributed switches, or clusters and data storage. Both pairs share the data center as their parent.
  - Grandparent-Grandchild: Resource A is the direct grandparent or grandchild of Resource B . For example, a data center is the parent of a cluster, a cluster is the parent of a host, and a host is the parent of a virtual machine. Therefore, the data center is the grandparent of the host, and the cluster is the grandparent of the virtual machine.
  - Peer: Resource A and Resource B do not have any of the above relationships, but they need to collaborate in certain scenarios. For example, data storage and image storage works together to provide services for clusters.

- Cardinality relationships: Similar to quantitative constraints in human society, including 1:n (one-to-many), n:1 (many-to-one), and n:n (many-to-many).

  Definitions:

  - **1:n**: Indicates that Resource A can create, add, or attach to multiple Resource B. For example, one cluster can contain multiple hosts, and one distributed switch can attach to multiple clusters.
  - **n:1**: Indicates that multiple Resource A can be created, added, or attached to Resource B. For example, multiple hosts can be added to the same cluster, and multiple clusters can be attached to the same distributed switch.
  - **n:n**: Indicates that Resource A can create, add, or attach to multiple Resource B, while Resource B can also be created, added, or attached to multiple Resource A. For example, one image storage can be attached to multiple data centers, and one data center can attach multiple image storage.

4 Core Design

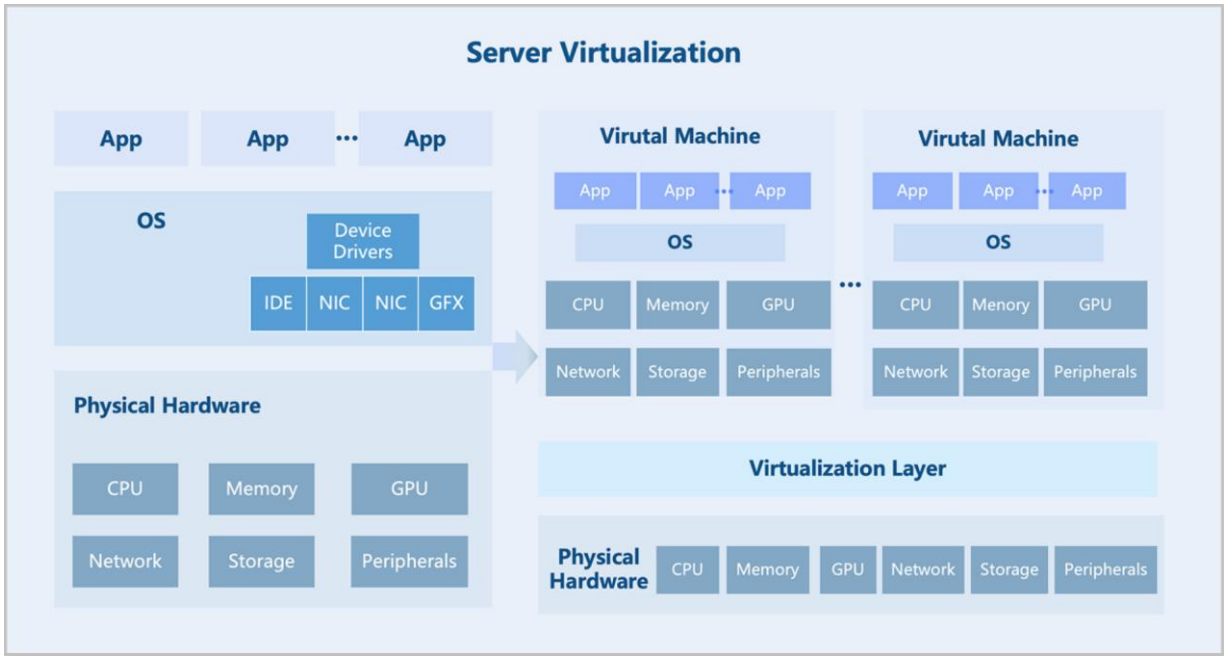
4.1 Resource Virtualization

4.1.1 Compute Virtualization

4.1.1.1 Overview

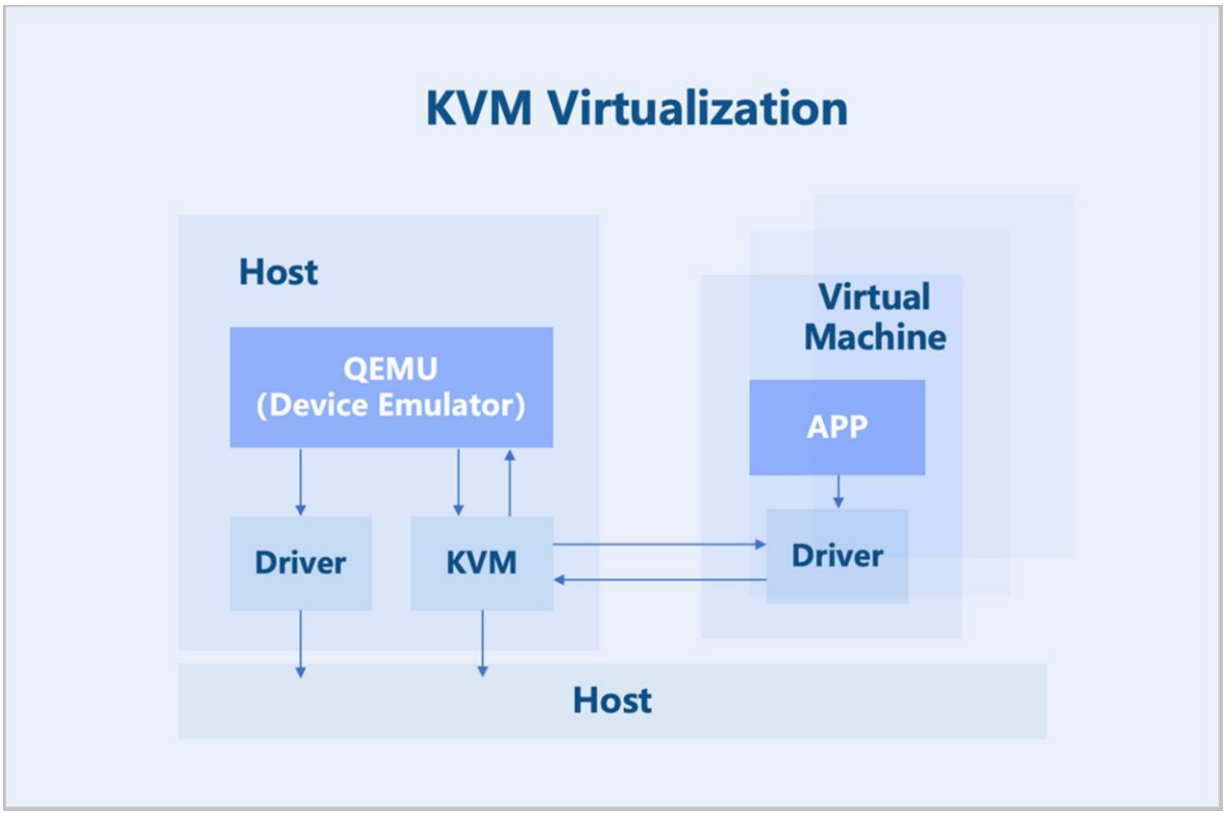
Compute virtualization abstracts physical server resources into logical resources through virtualization technology, enabling a single physical server to function as multiple isolated virtual machines. It pools hardware resources such as CPU, memory, disk, and I/O devices into a virtual resource pool for unified dynamic management. Consequently, it improves resource utilization, reduces system management costs, and increases IT agility to adapt to business changes.

Figure 4-1: Server Virtualization



NexaVM NSSV adopts a KVM-based hardware virtualization technology. As a Linux kernel module, KVM turns the Linux kernel into a hypervisor. KVM operates as a process within the Linux system and is scheduled by the standard Linux scheduler. As a result, KVM can leverage existing Linux kernel functionalities, such as memory management and CPU scheduling. However, KVM itself only provides CPU and memory virtualization. I/O device virtualization requires QEMU for full functionality. QEMU is a user-mode device emulator that provides virtual device models for virtual machines. It is responsible for creating, calling, and managing various virtual devices.

Figure 4-2: KVM Virtualization



4.1.1.2 Key Features

4.1.1.2.1 CPU Virtualization

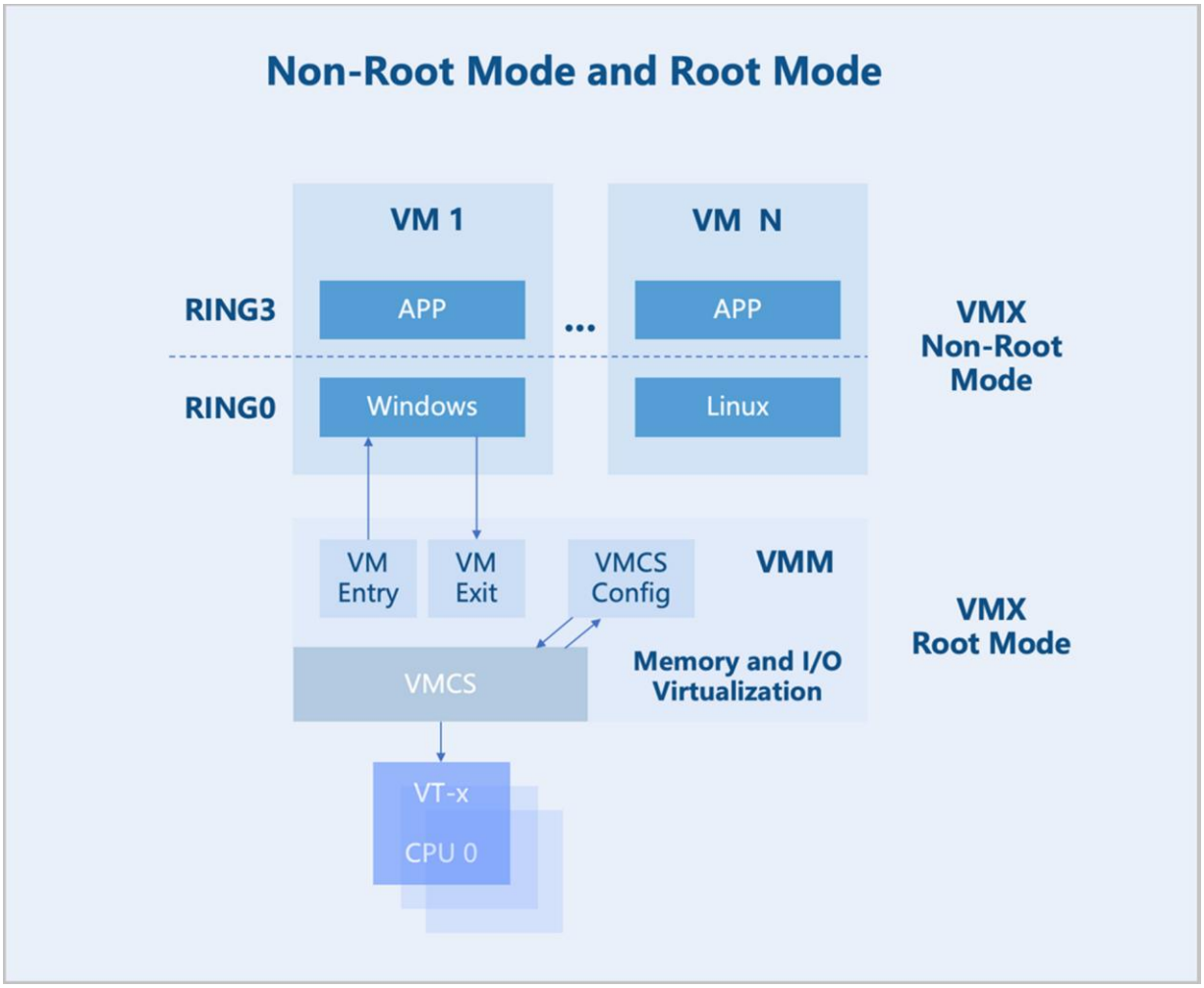
In the x86 architecture, CPUs generally have four privilege levels (RING0 to RING3) for operating systems and applications to access hardware. Linux uses only two of these levels: RING0 (kernel mode) and RING3 (user mode).

VMX Root Mode and VMX Non-Root Mode: For hardware-assisted virtualization, CPUs introduce two operation modes to enable virtual machines to run without modifying the operating system: VMX root mode and VMX non-root mode. The host runs in root mode, with its kernel in RING0 and user-mode programs in RING3. The virtual machine runs in non-root mode, with its kernel in RING0 and user-mode programs in RING3.

VM Exit and VM Entry: A switch from non-root mode to root mode occurs when a virtual machine in non-root mode encounters external interrupts, page faults, or actively executes the VMCALL

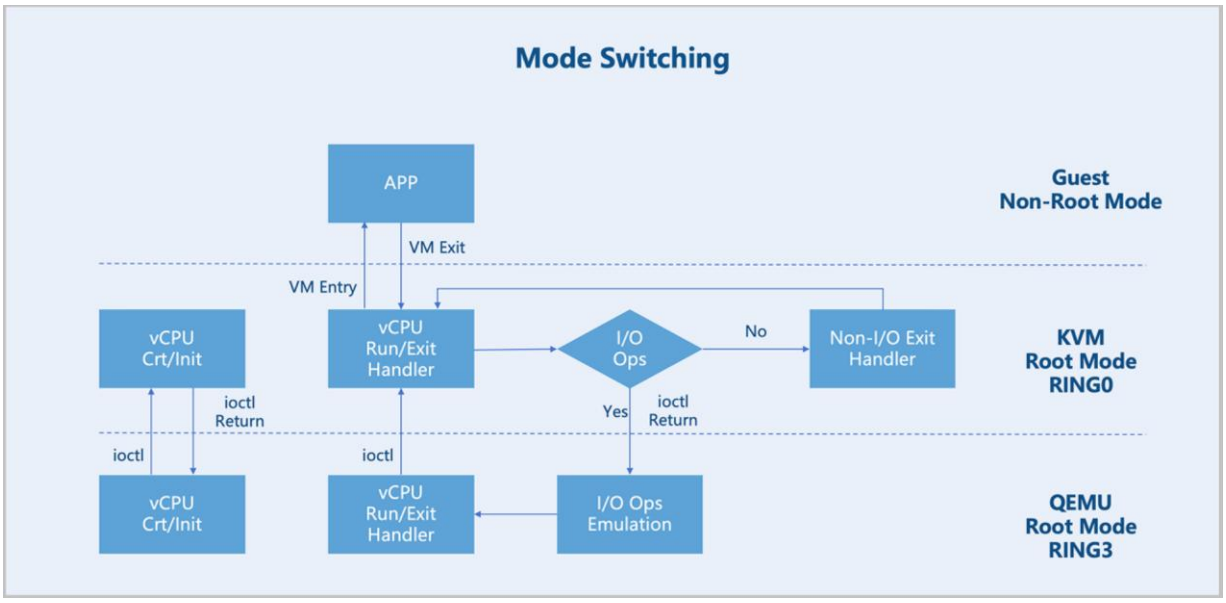
instruction to invoke VMM services. This entire process is called VM Exit. Conversely, when the VMM explicitly executes either VMLAUNCH or VMRESUME instruction to switch to non-root Mode , the hardware automatically loads the virtual machine context and executes VM instructions. This transition is called VM Entry.

Figure 4-3: Non-Root Mode and Root Mode



After a virtual machine triggers a VM Exit from non-root mode to root mode, KVM takes further action based on the exit reason. If the exit is due to an I/O operation, KVM delegates the processing to QEMU. For non-I/O exits, KVM handles them directly. After processing, KVM initiates a VM Entry to switch back to non-root mode for virtual machine execution.

Figure 4-4: Mode Switching



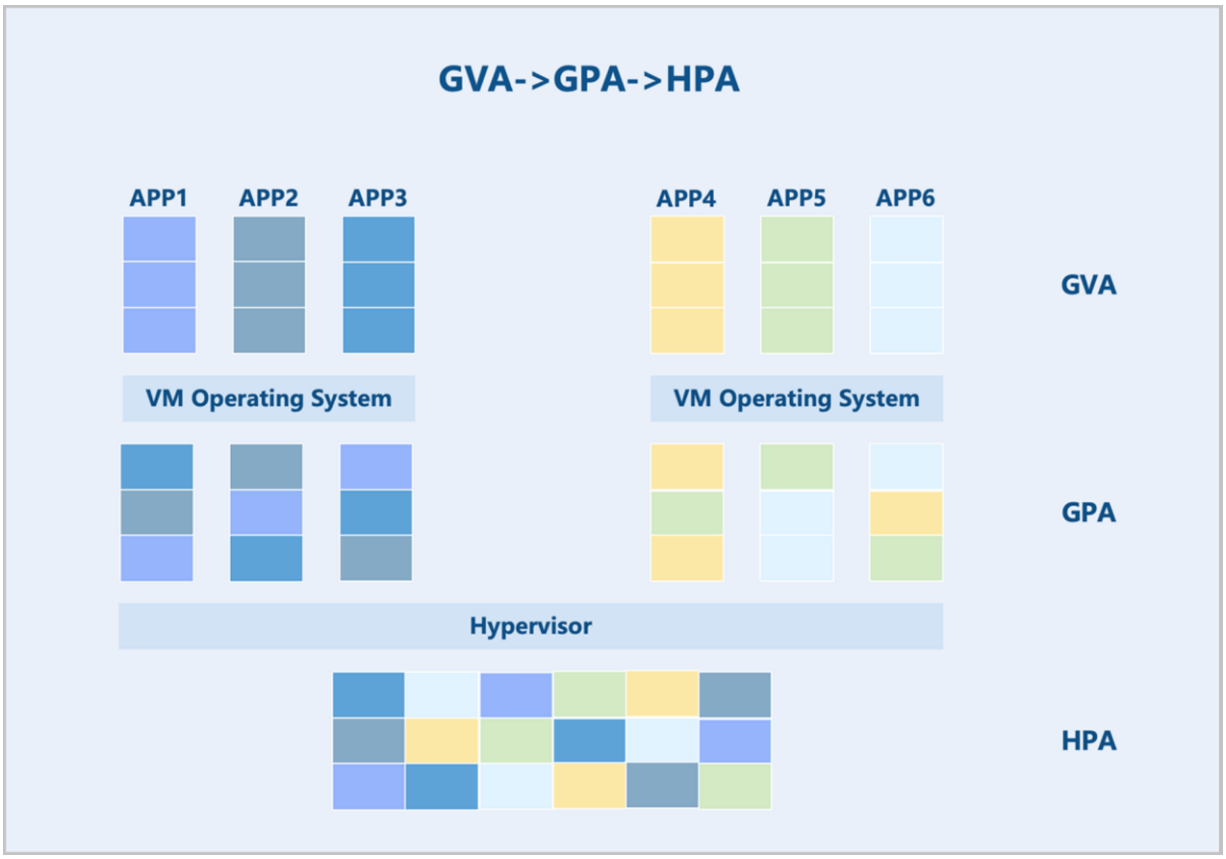
4.1.1.2.2 Memory Virtualization

The Virtual Machine Monitor (VMM) manages and allocates physical memory for each virtual machine. The guest OS sees a virtualized Guest Physical Address (GPA) space. The OS memory management module maps Guest Virtual Addresses (GVAs) to GPAs. The target address in instructions is also a GPA. In a non-virtualized environment, such an address would be the actual physical address. However, in a virtualized scenario, this address cannot be used directly. Instead, the VMM must first translate the GPA into a Host Physical Address (HPA), which is then executed by the physical processor.

The introduction of the GPA space requires memory virtualization to address two main issues:

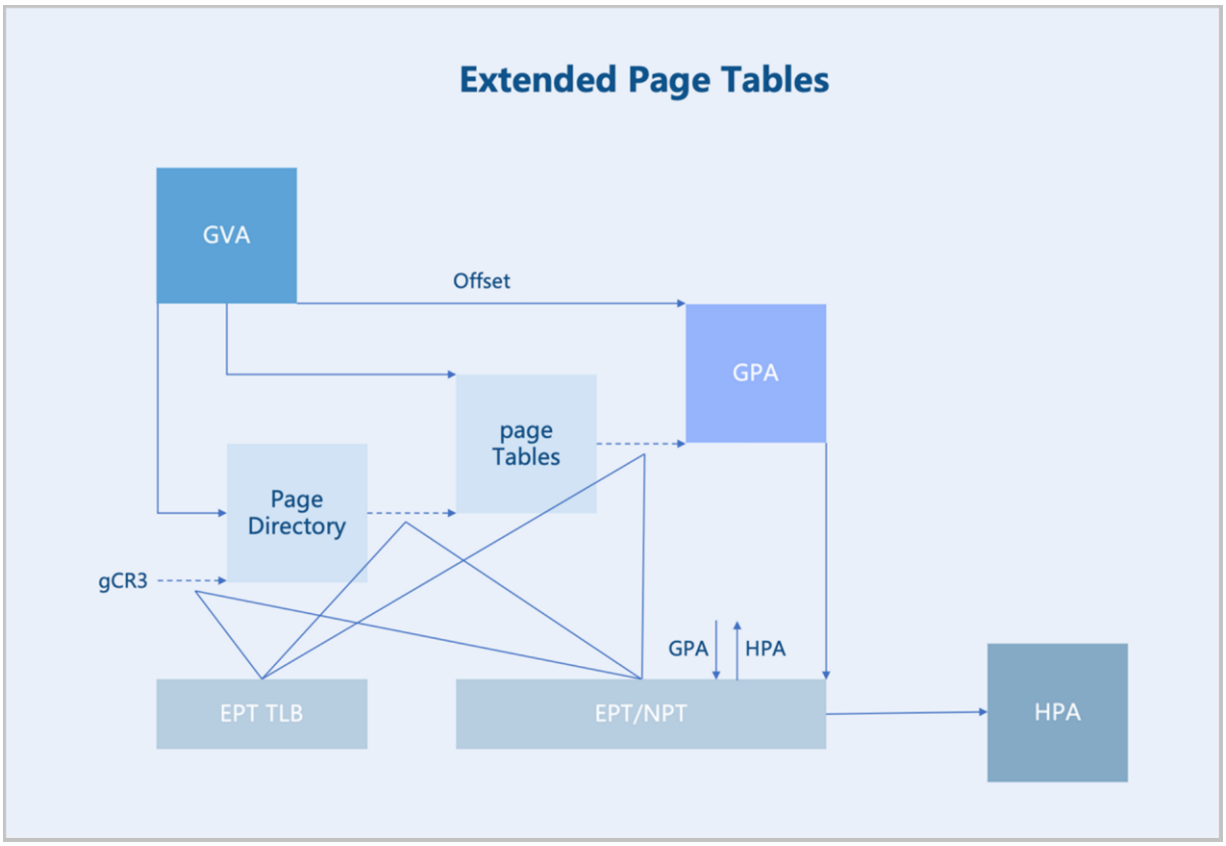
- Maintaining the mapping relationship between GPAs and HPAs.
- Translating GPAs into HPAs whenever a virtual machine accesses a GPA based on the established mapping.

Figure 4-5: GVA->GPA->HPA



Hardware-assisted memory virtualization uses Extended Page Tables (EPT) technology to translate GVAs into HPAs at the hardware level.

Figure 4-6: EPT



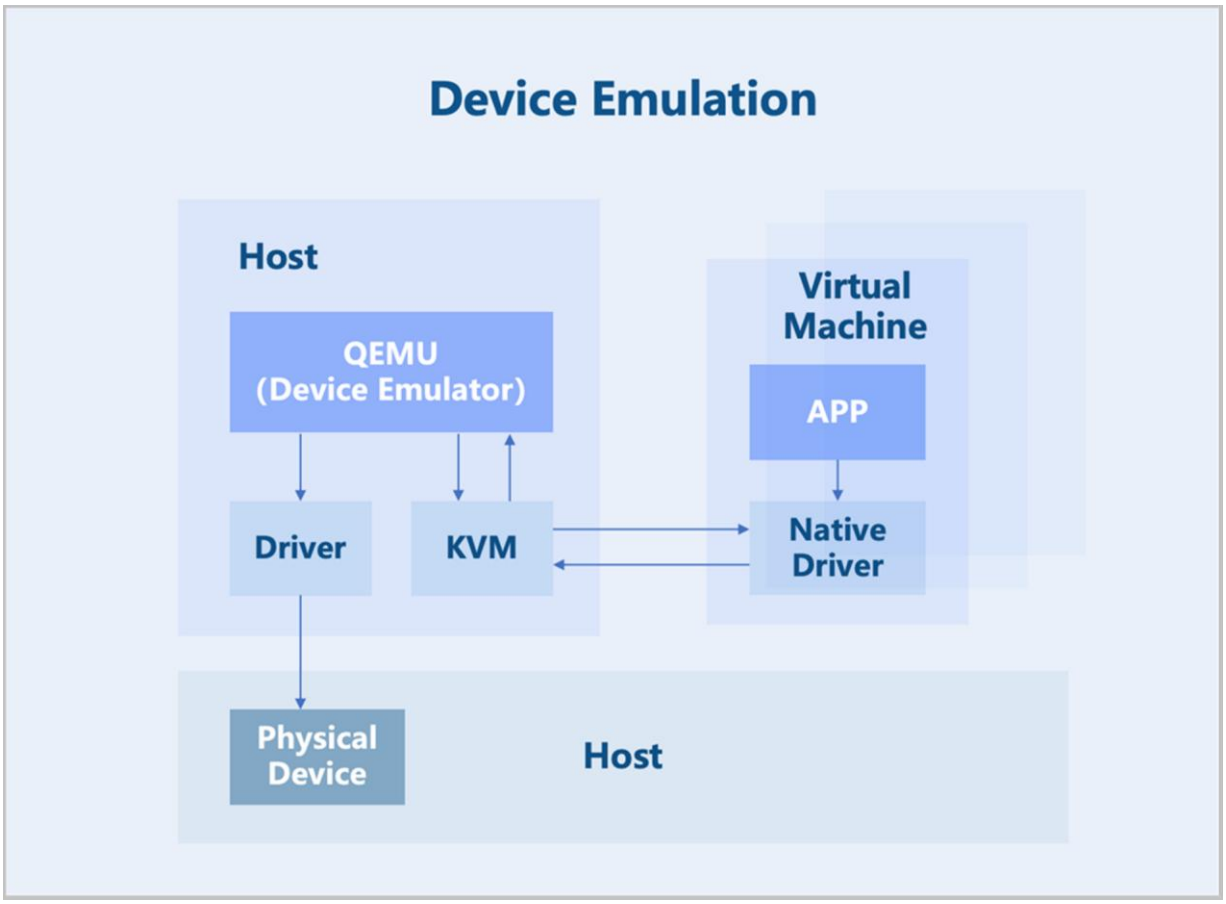
4.1.1.2.3 Device Virtualization

There are three main approaches to device virtualization: device emulation, paravirtualized devices, and device passthrough.

Device Emulation

Device emulation uses the device models provided by QEMU to fully emulate interfaces identical to those of physical devices. Consequently, the VM operating system can use these devices with its native drivers. However, device emulation can only replicate devices with basic functionalities and does not support complex features or models. While fully emulated devices offer good compatibility, they suffer from lower performance since they are purely software-based emulation.

Figure 4-7: Device Emulation



Paravirtualized Devices

Paravirtualized devices implement frontend and backend drivers. Utilizing a frontend driver in the virtual machine, requests are directly sent to a backend driver on the host through a transactio n-based communication mechanism, significantly reducing context switching overhead and

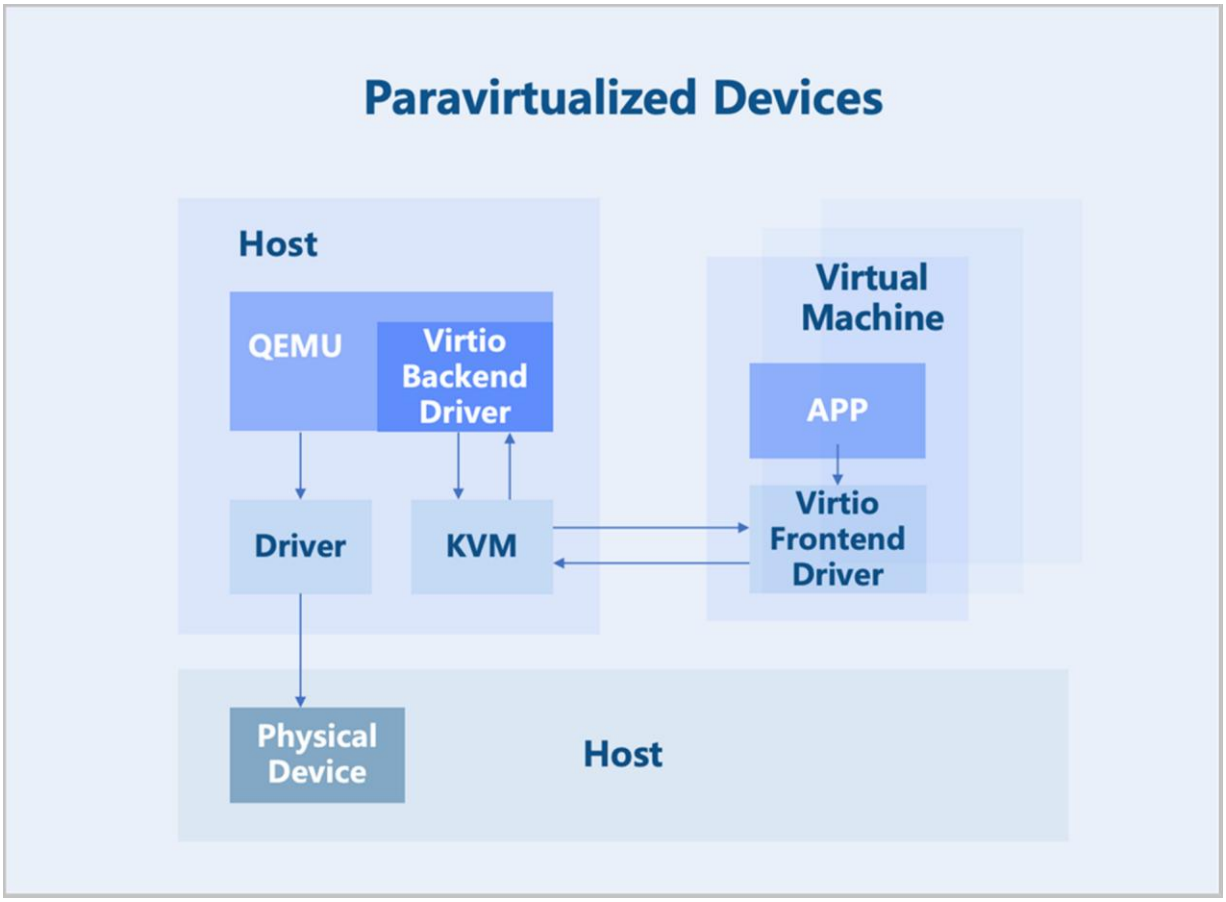
improving performance over full device emulation. However, since the Virtio backend driver is still implemented in QEMU, the I/O processing path involves multiple switches between user space and kernel space. To further enhance performance, the functionality of the Virtio backend driver can be moved into the kernel space. This implementation is known as the Vhost-kernel backend. With this change, data transmission requires only a single switch from user space to kernel space , thereby improving performance.

As technology evolves, moving data processing to user space achieves a greater flexibility.

Consequently, modifications to the original Vhost architecture led to the Vhost-user backend,

which works with relevant user-space libraries from DPDK and SPDK to further boost performanc e.

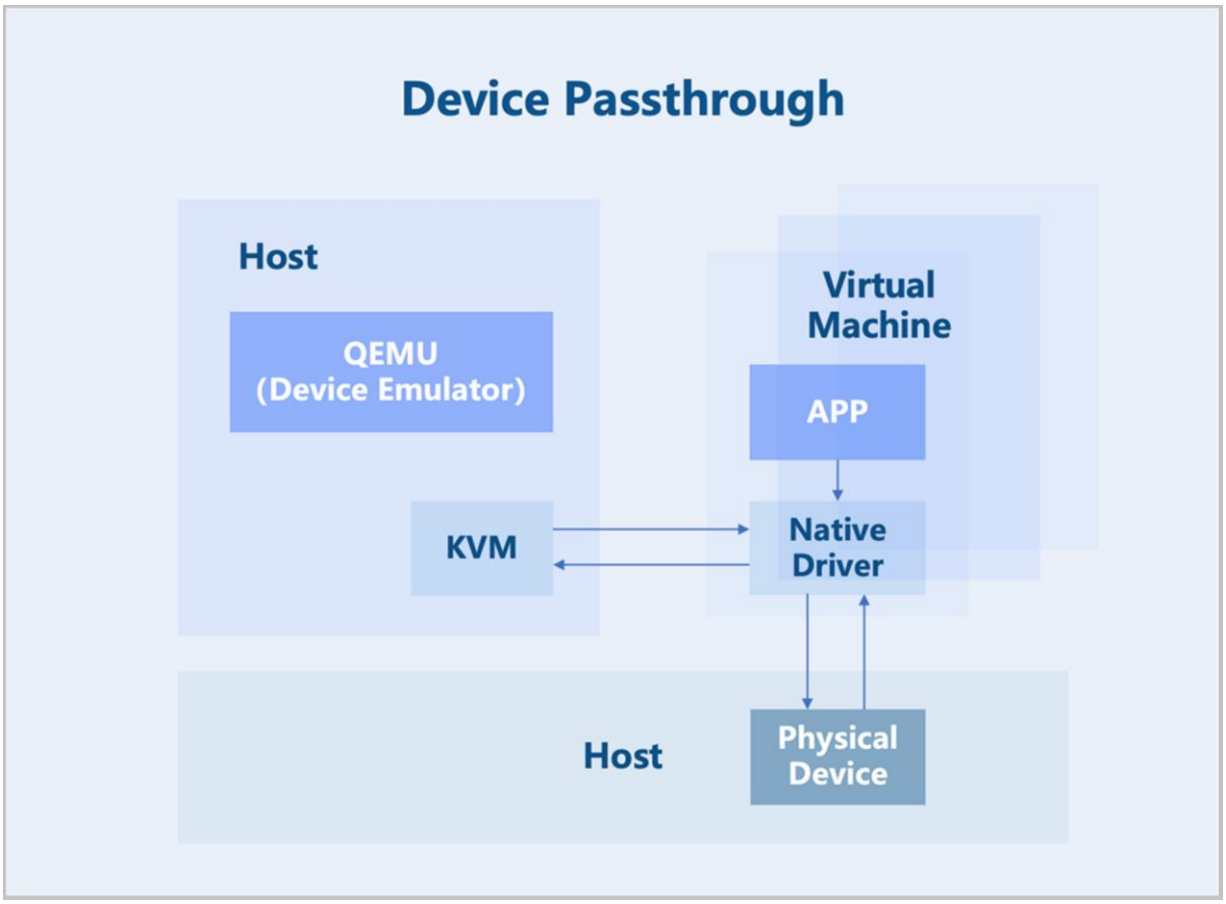
Figure 4-8: Paravirtualized Devices



Device Passthrough

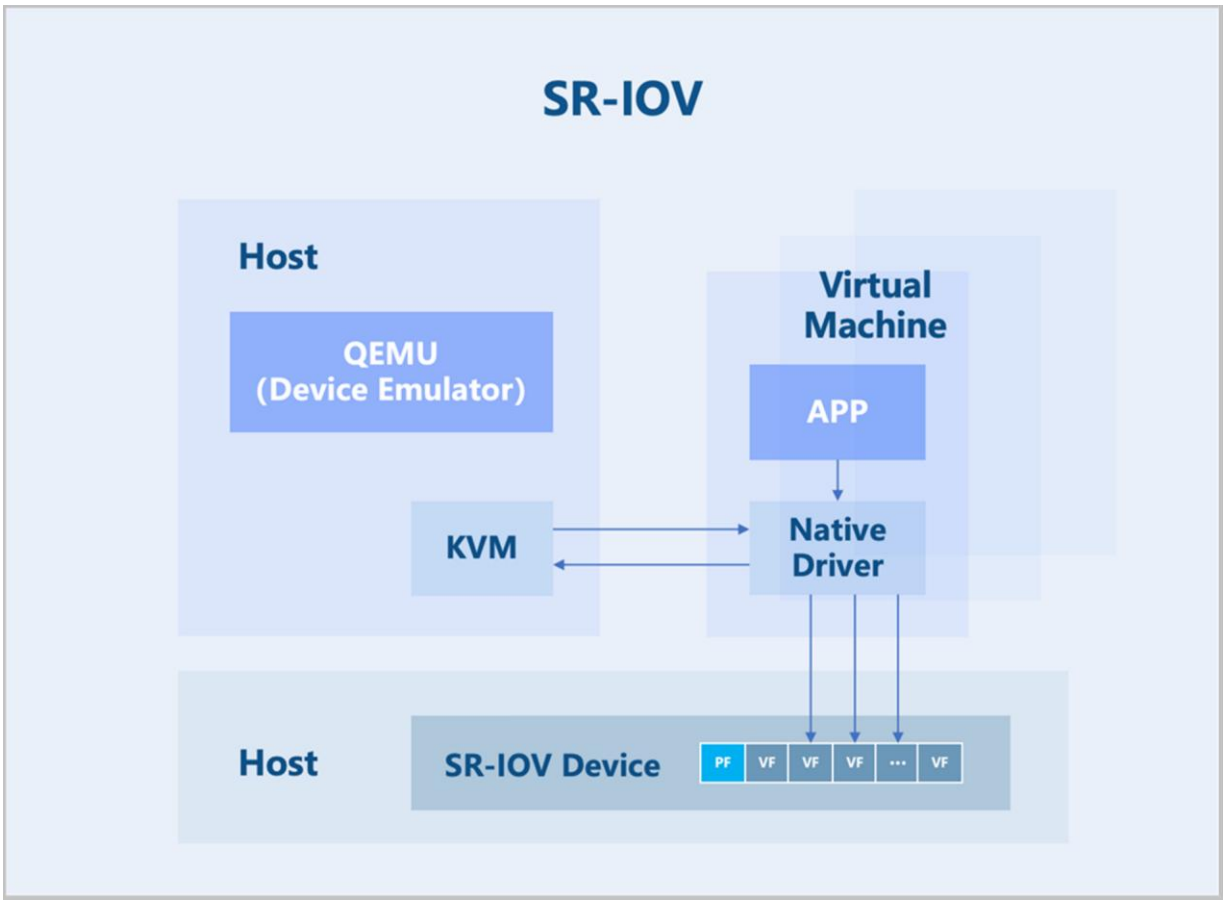
Device passthrough uses the hardware-assisted device virtualization technology to directly map a physical PCI/PCIe device to a virtual machine's address space. The virtual machine can use the native device driver to directly operate the devices, achieving performance nearly identical to that of physical devices. Once a physical device is passed through, it is dedicated to that virtual machine and cannot be shared with other VMs.

Figure 4-9: Device Passthrough



SR-IOV is an extension to the PCIe specification defined by the PCI-SIG. Its purpose is to provide a standardized specification for granting virtual machines independent memory spaces, interrupts , and DMA data streams. SR-IOV enables a single physical PCIe device (Physical Function, PF) to be virtualized into multiple virtual PCIe device devices (Virtual Function, VF). These VFs can then be passed through directly to virtual machines using device passthrough technology. This allows a single physical PCIe device to support multiple virtual machines.

Figure 4-10: SR-IOV



4.1.2 Storage Virtualization

4.1.2.1 Overview

Storage virtualization pools server storage resources to achieve unified integration, management, and scheduling. It provides various storage interfaces to upper-layer services, allowing business virtual machines to flexibly allocate and use storage space from the resource pool based on their needs. NexaVM NSSV supports integration with both centralized storage and distributed storage.

4.1.2.2 Key Features

4.1.2.2.1 Centralized Storage

Centralized storage refers to storing data on a central node composed of one or more servers. All services are centrally deployed on this node, which uniformly manages data for subsidiary nodes.

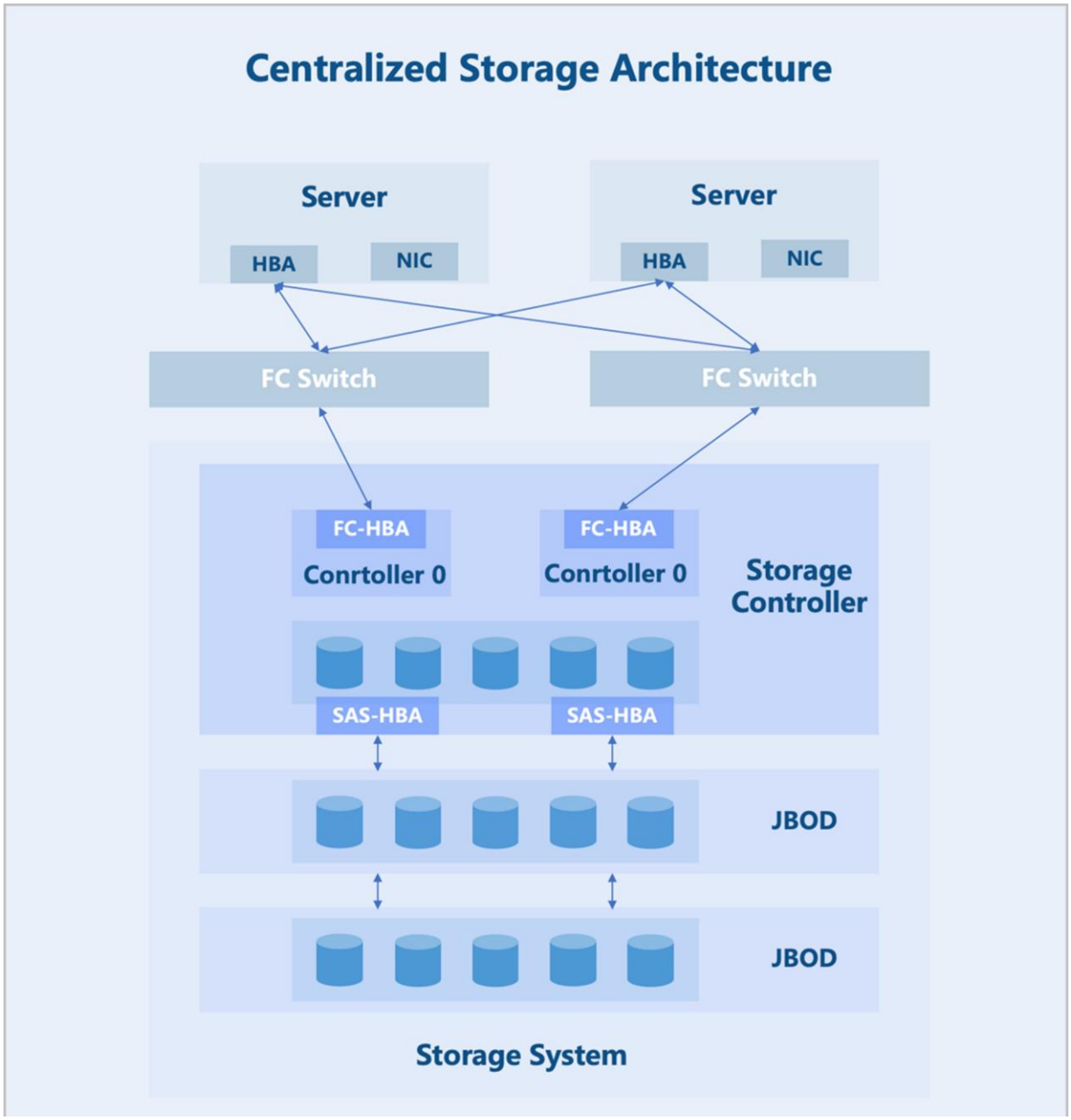
Data access is achieved through a single controller. Centralized storage falls into three categories : DAS, NAS, and SAN. You can select the appropriate type based on your data storage needs.

Functional advantages:

- High Performance and High Reliability: Centralized Storage stores data on a central node composed of one or more servers. It effectively mitigates the risks associated with data fragmentation and enhances both the performance and reliability of the storage system.
- Scalability: Centralized Storage can expand over the network to store more data, allowing the storage system to grow with your business needs.
- Security: Centralized Storage safeguards data security and integrity through security measures such as access control and data encryption.

On architecture: The storage controller is the core component of a centralized storage architecture . Typically, it contains two controllers operating in an active-standby mode to prevent the system-wide failures due to hardware issues. The storage controller features frontend and backend ports . Frontend ports provide storage services to servers, while backend ports expand the storage system capacity. Through backend ports, the storage controller connects to more additional storage devices, forming a large storage pool.

Figure 4-11: Centralized Storage Architecture



SAN

NexaVM NSSV supports integration with various types of centralized storage. This section focuses on SAN storage, which allows LUN devices allocated by users on the SAN to be directly utilized as storage pools and provisioned to business virtual machines. Unlike file system-based data storage, SAN storage offers advantages such as streamlined deployment, flexible scalability, and high performance. According to actual test results, SAN storage can fully leverage the

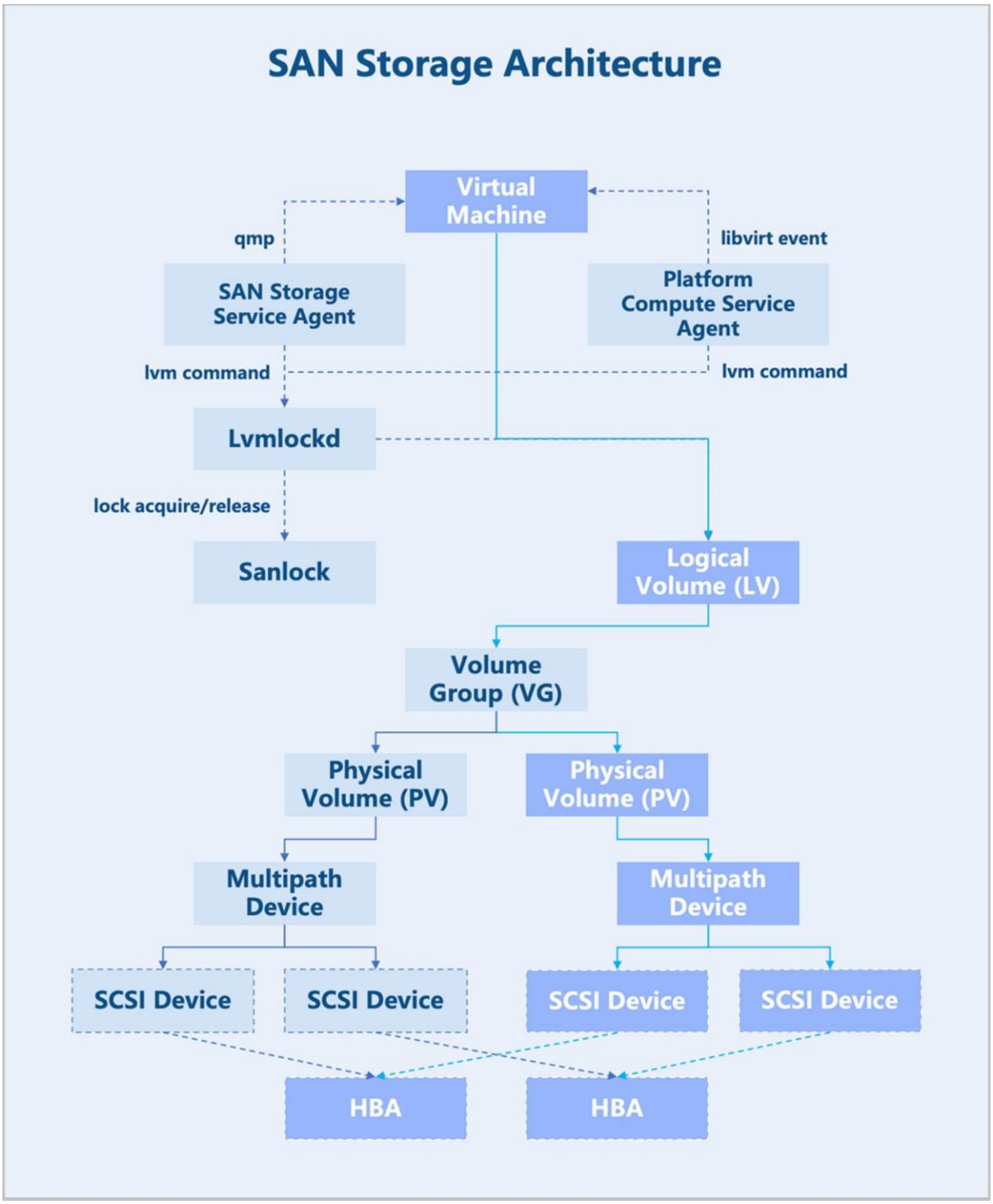
performance of physical disks. It currently supports shared access protocols including iSCSI, FC, and NVMe-oF.

Working principles:

- Block-Level Access: SAN divides data into blocks, each with a unique identifier. Servers read from or write to specific data blocks using these identifiers.
- Parallel Access: Multiple nodes can simultaneously access different blocks within SAN, enabling parallel read and write operations.
- Shared Access: Multiple nodes can share the same SAN, allowing concurrent access to the same data blocks. This is crucial for scenarios requiring data sharing in distributed systems.
- Fault Tolerance: SAN typically provides data redundancy and failure recovery mechanisms to ensure data safety and reliability.
- Scalability: SAN can scale capacity and performance by adding more storage nodes.
- Performance Optimization: SAN usually uses various techniques, such as caching and load balancing, to optimize performance.

On architecture: In centralized storage, a volume is mapped to a host. Depending on the storage controller's connection method, the multipath service on the host automatically aggregates multiple SCSI devices sharing the same WWID into a multipath device. NexaVM NSSV detects multipath devices with the same WWID across all hosts in the cluster and creates a shared volume group (VG) for the user-selected volume. Logical volumes (LV) are provisioned from this VG, corresponding to volumes, snapshots, and other resources in NexaVM NSSV. To maintain storage cluster consistency, NexaVM NSSV implements a shared storage lock mechanism (Sanlock) for maintaining storage heartbeats, node management, and metadata management.

Figure 4-12: SAN Storage Architecture



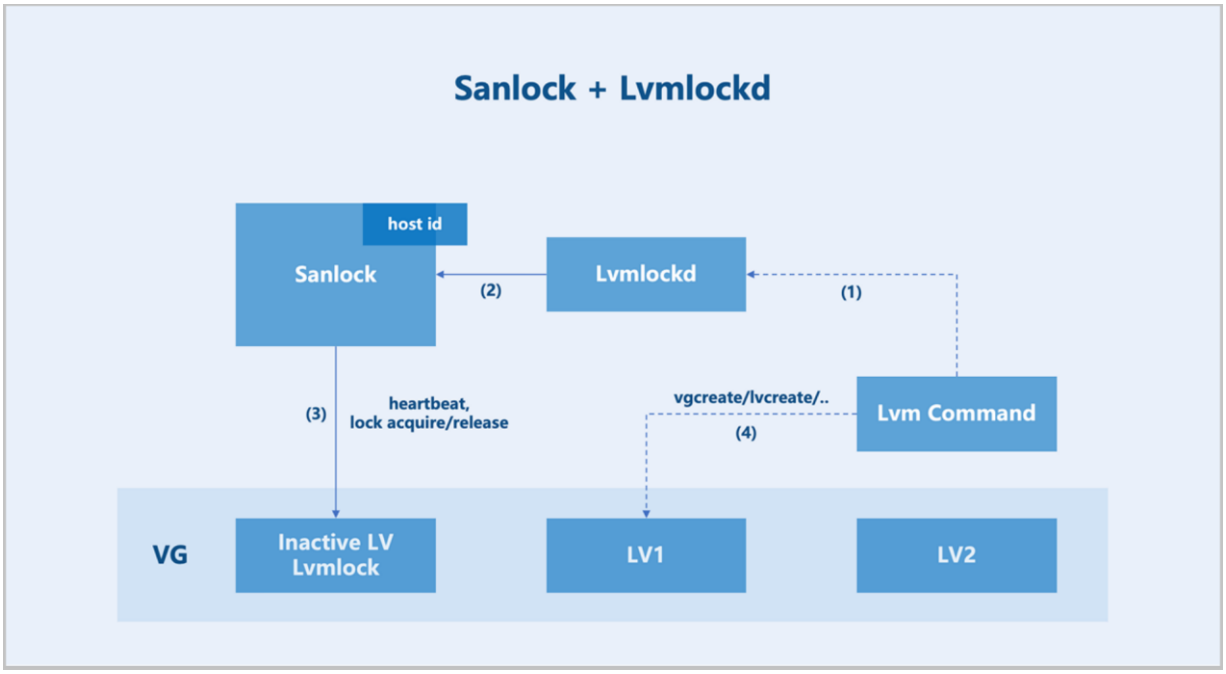
Sanlock + Lvmlockd

Sanlock (Shared storage lock manager) is essentially a lease management mechanism built upon Lamport's algorithms: Delta Paxos and Disk Paxos. This system handles cluster membership management, heartbeat maintenance, arbitration, and message passing. The next step involves translating this lease mechanism into locks usable by LVM: Lvmlockd.

With Sanlock and Lvmlockd, most LVM commands from sever are first sent to Lvmlockd, which translates LVM operations into lock operations recognizable by Sanlock.

Different resource operations require different lock permissions. Resource operations proceed only after obtaining the appropriate locks, ensuring the security of metadata and data. This entire process requires no intermediate node coordination or centralized operations, forming a fully distributed architecture.

Figure 4-13: Sanlock + Lvmlockd



4.1.2.2.2 Distributed Storage

NexaVM NSSV supports integration with various distributed storage systems. This section primarily introduces the Self-Developed Distributed Storage. It provides comprehensive resource management capabilities for servers, hard disks, data disks, block storage volumes, storage

pools, buckets, and more. You can create storage pools with different data redundancy types and configure buckets with specific storage policies and access permissions. This enables rapid and cost-effective deployment of a simple, stable, secure, and highly efficient storage infrastructure.

Functional advantages of Self-Developed Distributed Storage:

- Elastic Scalability: Supports horizontal storage expansion. Under the "Internet Plus" trend, emerging services proliferate and data grows exponentially, posing significant challenges to traditional centralized storage. Distributed storage allows cluster capacity and performance upgrades by simply adding new nodes to the cluster.
- High Availability: Supports multiple data redundancy policies, including replication and erasure coding. With replication, you can online adjust the number of replicas in a storage pool. Data is duplicated across multiple nodes, so even if one or several nodes fail, data remains accessible from other nodes, ensuring the availability of the storage cluster. Combined with flexible fault domain policies, the storage service stays reliable even during unexpected events such as hardware failures or network outages.
- Performance Optimization: Optimizes data processing performance through proprietary cache acceleration technology. It utilizes enterprise-grade NVMe SSDs and SATA SSDs to accelerate read and write operations for backend HDD devices, effectively reducing latency and improving the I/O performance of the storage cluster. Furthermore, it employs load balancing technology to evenly distribute storage I/O loads across all nodes, ensuring optimal utilization of system resources.
- Easy Management: Provides a unified, user-friendly management interface and standard RESTful APIs, enabling administrators to easily perform basic lifecycle management and maintenance of physical resources, as well as data operations and recovery tasks.
- High Cost-Effectiveness: Supports deployment on standard hardware, significantly reducing hardware costs. Additionally, elastic scaling capability allows dynamic storage resource adjustment based on business needs, avoiding resource waste and further lowering operational expenses.

Architecture of Self-Developed Distributed Storage:

- Hardware Layer: Supports standard x86 servers and supports integrating legacy equipments into a shared pool, helping organizations maximize the value of their current IT assets.
- Platform Layer: Copes with the consistent hashing challenges in storage services through a fault domain algorithm. Essentially, this algorithm enhances the basic hash algorithm by incorporating physical deployment logic and cluster state parameters. When cluster data

migration occurs, the algorithm minimizes data migration volume based on the actual hardware layout. For example, based on the provided physical topology, the fault domain algorithm can selectively perform migrations within a server or within the same rack/core switch, significantly reducing resource consumption caused by migration. Additionally, the algorithm distributes data based on the cluster state, ensuring full compatibility with different storage devices.

- Data Layer: Uses a proprietary caching technology that leverages enterprise-grade NVMe SSDs and SATA SSDs for read and write caching of backend HDD devices. The technology stores frequently accessed hot data on high-speed SSD devices and dynamically moves cold data to slower HDD devices using an intelligent flushing policy. This approach both enhances the overall performance of the storage cluster and provides massive storage space for upper-layer services.
- Interface Layer: Provides virtual volume devices to operating systems, cloud platform, and databases via the RBD interface.
- Service Layer: Integrates with cloud platforms to deliver cloud infrastructure services.

4.1.2.2.2.1 Data Storage

4.1.2.2.2.1.1 Cache Acceleration Solution

A cache acceleration solution typically uses SSD-based caching devices to accelerate the data access for HDD data disks. Self-Developed Distributed Storage uses high-speed SSDs through the ZAS cache module to provide I/O caching for traditional HDD devices. The ZAS cache module caches frequently accessed hot data in the high-speed SSD devices and returns it to the application, significantly improving I/O performance, especially in scenarios characterized by hot data access patterns.

The ZAS cache module supports two caching policies: Writethrough and Writeback. In Writethrough mode, data is written simultaneously to the cache and the backend storage device, ensuring data consistency. In Writeback mode, most of the cache is used to buffer write data, ensuring that dirty data is written sequentially to the backend storage device. The system disables the Writeback policy by default but you can switch caching policies at runtime.

The ZAS cache module has the following characteristics:

- Flexible Cache Data Migration

  The ZAS cache module automatically migrates data from slow disks to the SSD cache based on data access frequency and heat to improve access speed for that data. Simultaneously,

based on cache usage and available cache space, the module automatically migrates less frequently used data from the SSD cache to slow disks to free up cache space.

- Data Protection

  If a data I/O error occurs on the flash device, the ZAS cache module first attempts to read from the disk to recover the data or marks the cache entry as invalid. For unrecoverable errors, such as those involving metadata or dirty data, the cache module automatically disables caching.

4.1.2.2.2.1.2 Automatic Thin Provisioning

Before data is written to the logical volume, thin provisioning provides upper-layer applications with more virtual storage space than what is physically available in the storage cluster. This offers convenient scalability and improves storage space utilization efficiency.

4.1.2.2.2.1.3 Storage Pool Recovery QoS

QoS (Quality of Service) is a technology used to address I/O resource allocation issues. It helps you throttle I/O read and write speeds, facilitating the rational allocation of resources.

Within data nodes, an **`op_shardedwq`** queue handles various I/O requests from the upper level. This is a composite queue, typically containing several sub-queues. After I/O requests are dequeued, they interact with the disk via the ObjectStore interface. I/O types fall into two main categories: business read/write I/O requests from clients, and I/O generated by internal activities of the storage system, including I/O requests between data nodes, SnapTrim, Scrub, and Recovery.

The Self-Developed Distributed Storage uses a weighted priority queue (wpq) to categorize and store the aforementioned I/O types into corresponding sub-queues. Each priority (prior) queue is created when its first request is enqueued. During dequeue, a weighted probability method determines the prior level. The priority (prior) of each queue serves as its weight. The probability of a prior queue being selected equals the ratio of its priority weight to the total weight of all queues. Even if selected, a prior queue is not guaranteed to dequeue a request; this also depends on the size of the request about to be dequeued.

You can set a recovery QoS for a storage pool with different recovery policies for various business scenarios.

- Low-Speed Recovery: Low-Speed Recovery gives a higher priority to the business bandwidth . The recovery time is relatively long. Any hardware failures during the recovery may reduce the data security level. We recommend that you choose Low-Speed Recovery in a production environment.

- Medium-Speed Recovery: Mid-Speed Recovery gives the same priority to the business bandwidth and recovery bandwidth. The recovery time is medium. A saturated performance may increase the I/O latency.
- High-Speed Recovery: High-Speed Recovery gives a higher priority to the recovery bandwidth. The recovery time is relatively short. A saturated performance may affect business performance.

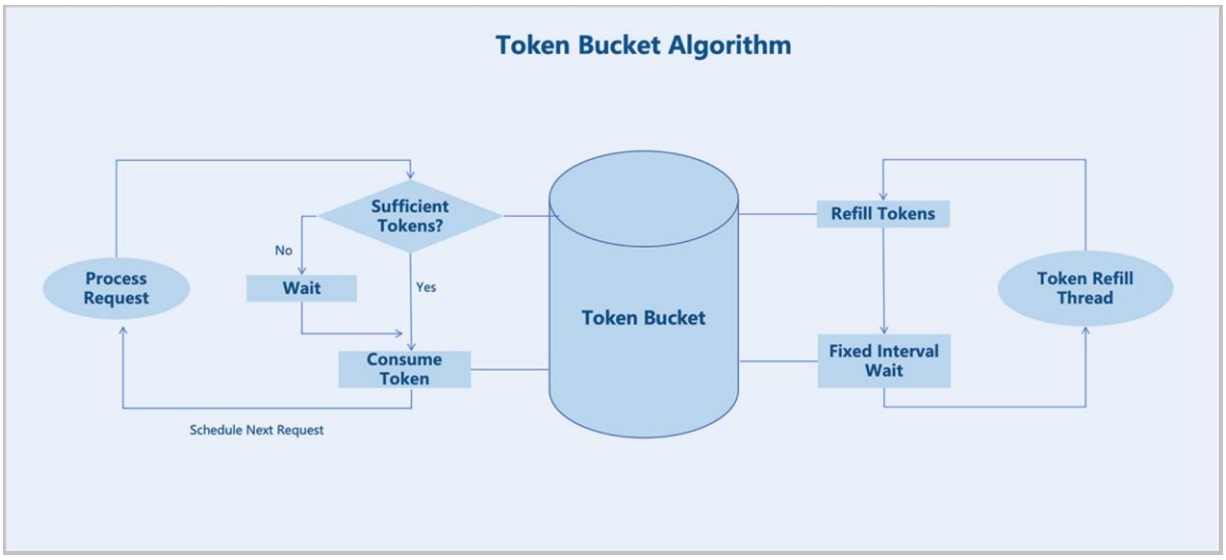
4.1.2.2.2.1.4 Volume Business QoS

The Self-Developed Distributed Storage supports configuring business QoS for block storage volumes, including maximum IOPS and maximum read/write bandwidth. By setting a business QoS, you can control the performance of different block storage volumes to meet diverse business performance requirements.

The Self-Developed Distributed Storage uses a token bucket algorithm for I/O traffic control:

- The system generates tokens at a fixed rate and places them into a bucket. Every I/O request must acquire a token from this bucket.
- Requests that fail to acquire a token must queue to obtain one, thereby limiting the average data flow rate into the system.
- When the token distribution rate is slower than the token generation rate, tokens can accumulate in the bucket. This allows direct consumption of tokens from the bucket during short-term traffic bursts.

Figure 4-14: Token Bucket Algorithm



4.1.2.2.2.2 Data Protection

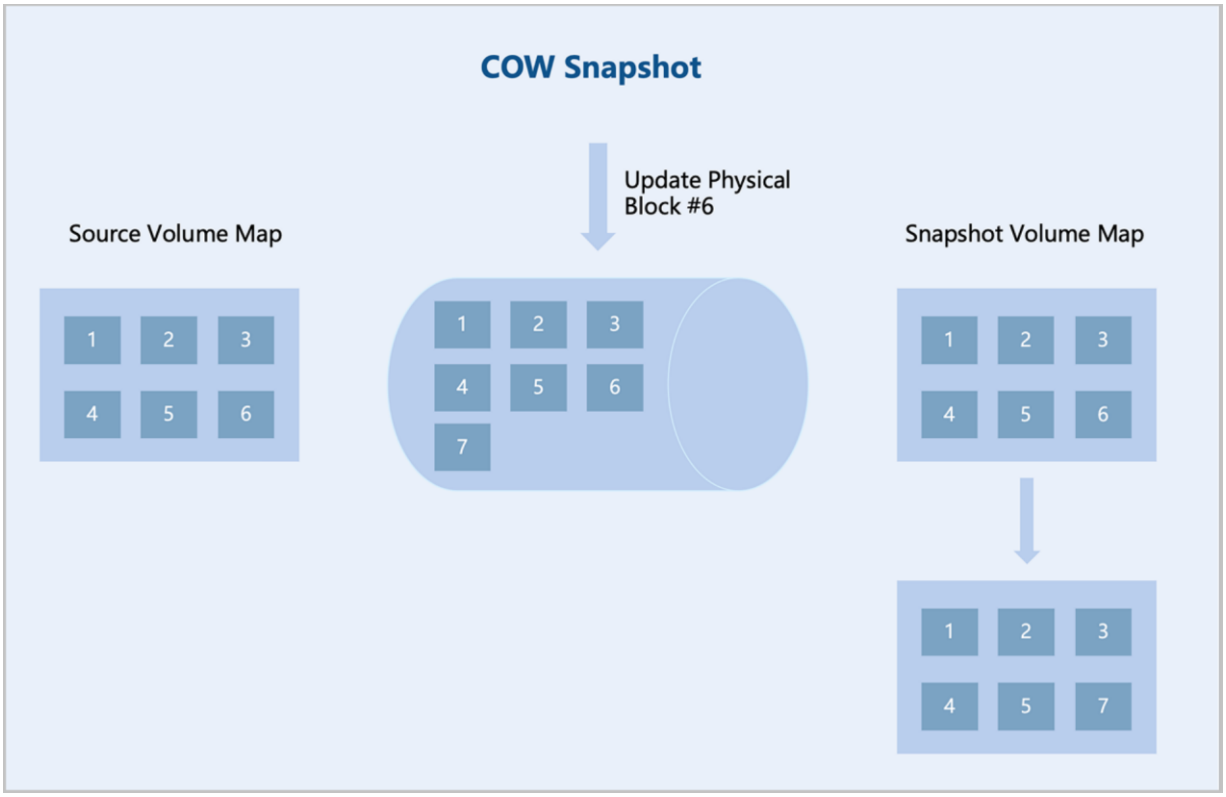
4.1.2.2.2.2.1 Volume Snapshot Protection

Snapshots serve as an important data protection mechanism in the Self-Developed Distributed Storage. Snapshots allow users to create a complete state copy of a block storage volume at a specific point in time without interrupting services or impacting the production environment.

Snapshot Creation

The Self-Developed Distributed Storage uses a COW (Copy-On-Write) snapshot technology. After creating a COW snapshot, the system creates a read-only copy of the original block device within the storage cluster. Data blocks are duplicated only when modifications occur in either the original device or the snapshot. This preserves the state of the original data in the snapshot while avoiding unnecessary data redundancy. The system also records the relationship between the original block device and the snapshot, as well as metadata information for each snapshot, such as creation time, size, and parent snapshot. Recording this metadata enables the system to accurately locate the specific snapshot state during data recovery.

Figure 4-15: COW Snapshot



Data Rollback and Recovery

The Self-Developed Distributed Storage supports creating new block storage volumes from snapshots, also known as cloning, where the new volume starts from the state of a specific snapshot. Additionally, in backup recovery or data rollback scenarios, you can revert a block storage volume to the state of a specified snapshot.

4.1.2.2.2.2.2 Data Redundancy

Overview

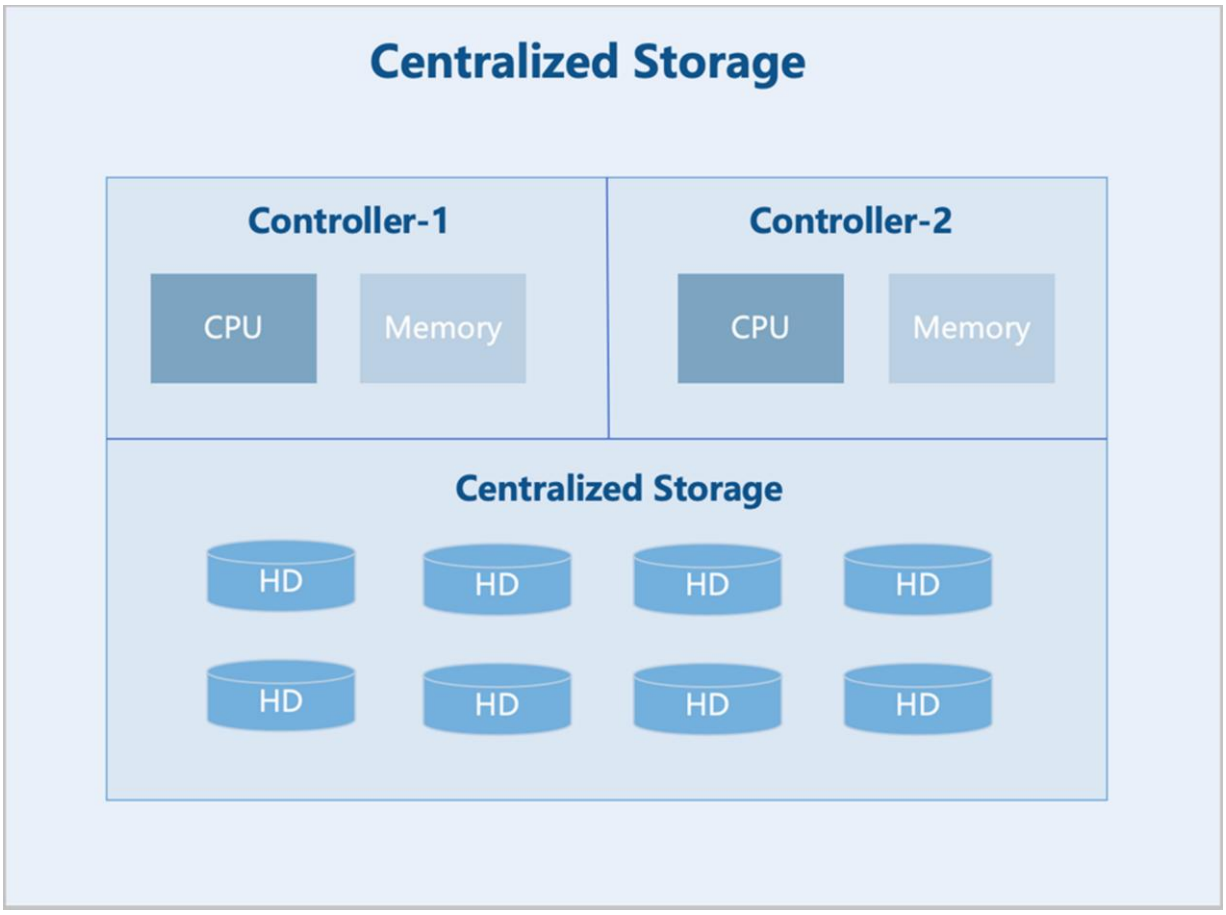
Data redundancy is the cornerstone for storage systems to achieve high availability, data protection, and business continuity. Faced with increasing data volumes, complex application scenarios, and potential risks such as hardware failures, network interruptions, and human errors, appropriate data redundancy policies provide you with a robust data protection barrier, ensuring business continuity and maximizing data value. This section briefly compares the characteristics of redundancy techniques in centralized storage and distributed storage, followed by a detailed explanation of the two core data protection policies adopted by the Self-Developed Distributed Storage: replication and erasure coding (EC).

Centralized Storage vs. Distributed Storage

Centralized Storage

Traditional centralized storage uses controllers and disk enclosures to provide data management and read/write capabilities. It typically employs dual controllers for redundancy, while some high-end storage uses multiple controllers. Storage space is provided through the controller's built-in drive bays or externally connected expansion enclosures. Traditional centralized storage commonly uses RAID to protect data, such as RAID 5, RAID 6, or RAID 10.

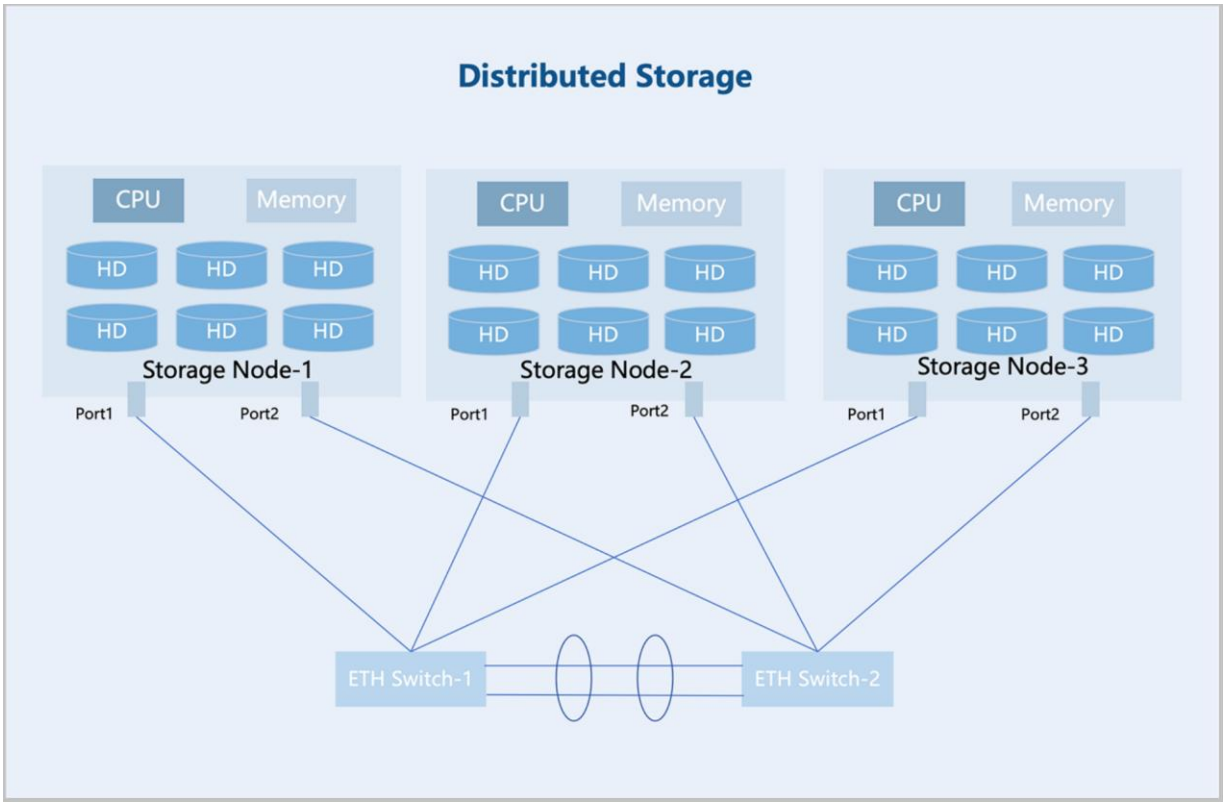
Figure 4-16: Centralized Storage



Distributed Storage

Distributed storage adopts a decentralized architecture where each storage node provides computing and storage resources, enabling more flexible scalability and larger storage capacity. Storage nodes are interconnected via standard Ethernet switches and managed by the distributed storage software to provide a unified storage resource pool to upper-layer services. Furthermore, distributed storage supports horizontal scaling. A single cluster can expand to thousands of nodes to provide EB-level capacity, making it suitable for massive data storage scenarios.

Figure 4-17: Distributed Storage



Centralized Storage vs. Distributed Storage

- **Cross-Node Redundancy**: Distributed storage supports redundancy across multiple nodes. For example, a three-replica policy can tolerate two nodes to fail simultaneously without data loss, whereas RAID only provides redundancy within a single node.
- **Global Hot Spares and Data Recovery**: Unlike RAID, which relies on the dedicated hot spare disk, distributed storage uses all available disks for data recovery, significantly improving

  efficiency. Additionally, distributed storage requires no additional hardware support, whereas RAID typically needs a dedicated RAID card.

Replication

Definition

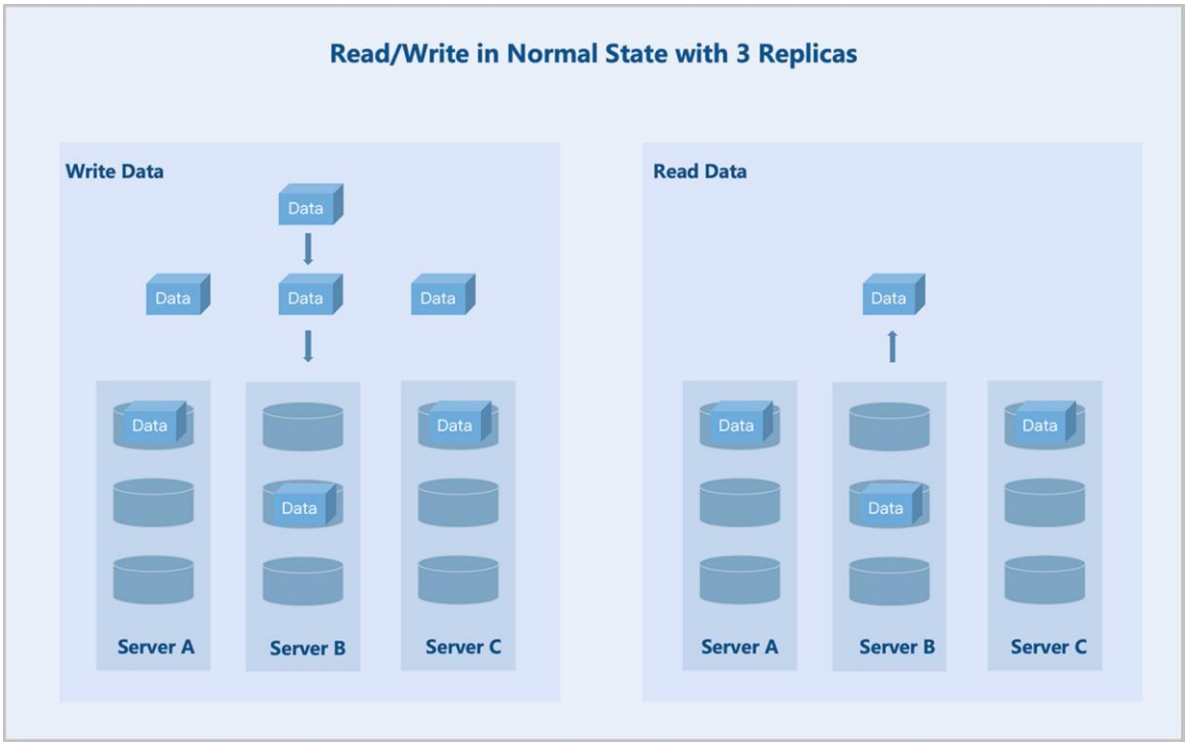
Replication is a data protection technique that achieves data redundancy and high availability by storing the copies of the same data across different nodes. If a node fails, data can be recovered from the replicas on other nodes. You can configure 2 to 6 replicas, with 3 replicas recommended for production environments.

Read/Write Principles

- **Read/Write in Normal State**

  Taking a 3 replicas at the server level as an example: During data write, the system copies the data into three identical replicas and stores each on data disks in three different servers. During data read, the system reads the data from any one of the servers and returns it to the user.

Figure 4-18: Read/Write in Normal State with Three Replicas

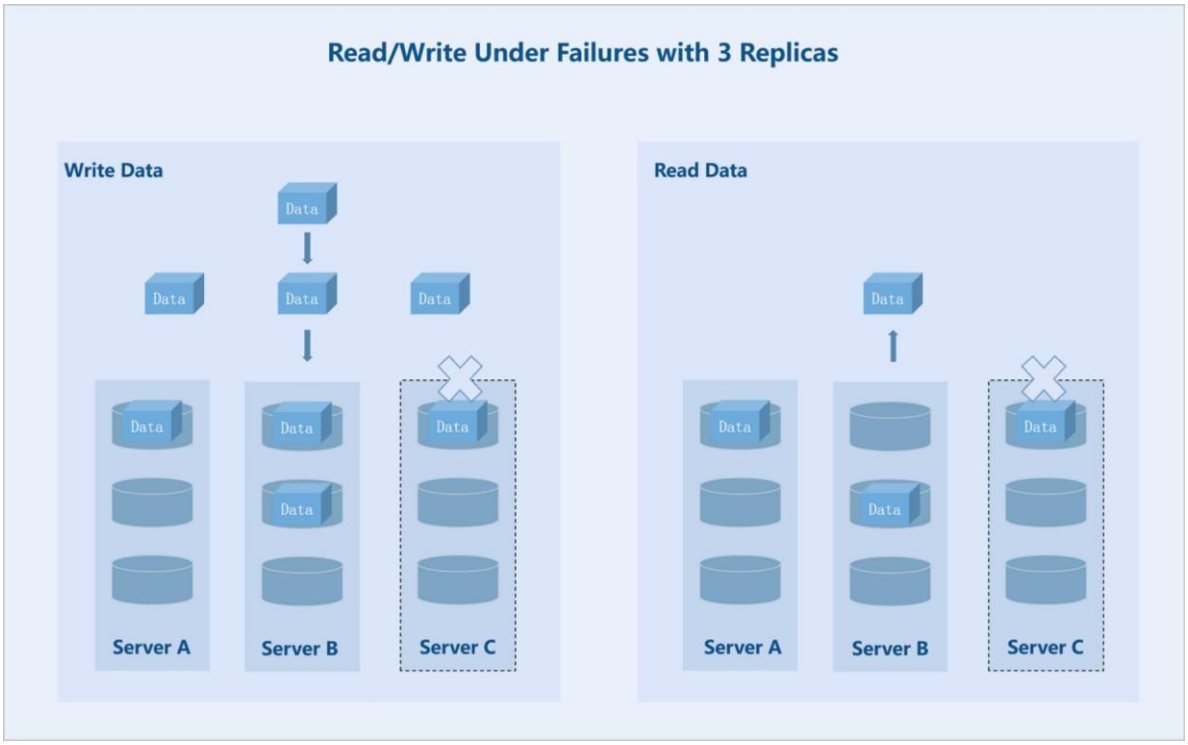


- **Read/Write Under Failures**

  Taking a 3 replicas at the server level as an example: If server C fails, the system stores

replicas on the remaining two servers. During data read, the system reads one replica from either of the remaining two servers and returns it to the user.

Figure 4-19: Read/Write Under Failures with Three Replicas



Erasure Coding (EC)

Overview

Erasure Coding (EC) is a data protection technique that splits data into K data blocks and generates M parity blocks through a parity algorithm to achieve data correction and recovery. Compared to traditional replication, EC saves storage space and network bandwidth while ensuring data reliability.

- Standard EC (K+M): K indicates the number of data blocks, and M indicates the number of parity blocks. This configuration tolerates simultaneous failures of M fault domains without affecting data availability.
- Folded EC (K+M:B): K indicates the number of data blocks, and M:B indicates the number of parity blocks. This configuration tolerates simultaneous failures of M hard disks or B fault domains without affecting data availability.

Policy Types

The following table lists the EC policies provided by the Self-Developed Distributed Storage.

EC Policy		Storage Efficiency
Recommended Values	2+1	66.67%
	4+2	66.67%
	8+3	72.73%
	4+2:1	66.67%
	8+2:1	80.00%
	16+2:1	88.89%
Custom		K/(K+M)

Standard EC

Read/Write Principles

- Read/Write in Normal State

  Standard EC (K+M): Taking a server-level 4+2 EC policy as an example: During data write, the system splits the data into 4 equally sized data blocks and generates 2 parity blocks of the same size through the parity algorithm. The system randomly stores these 6 blocks across 6 servers. If any 2 servers fail, data remains accessible. During data read, the system reads data blocks from different data disks on 4 servers, assembles the 4 data blocks into complete data, and returns it to the user.

Figure 4-20: Data Write in Normal State with EC (4+2)

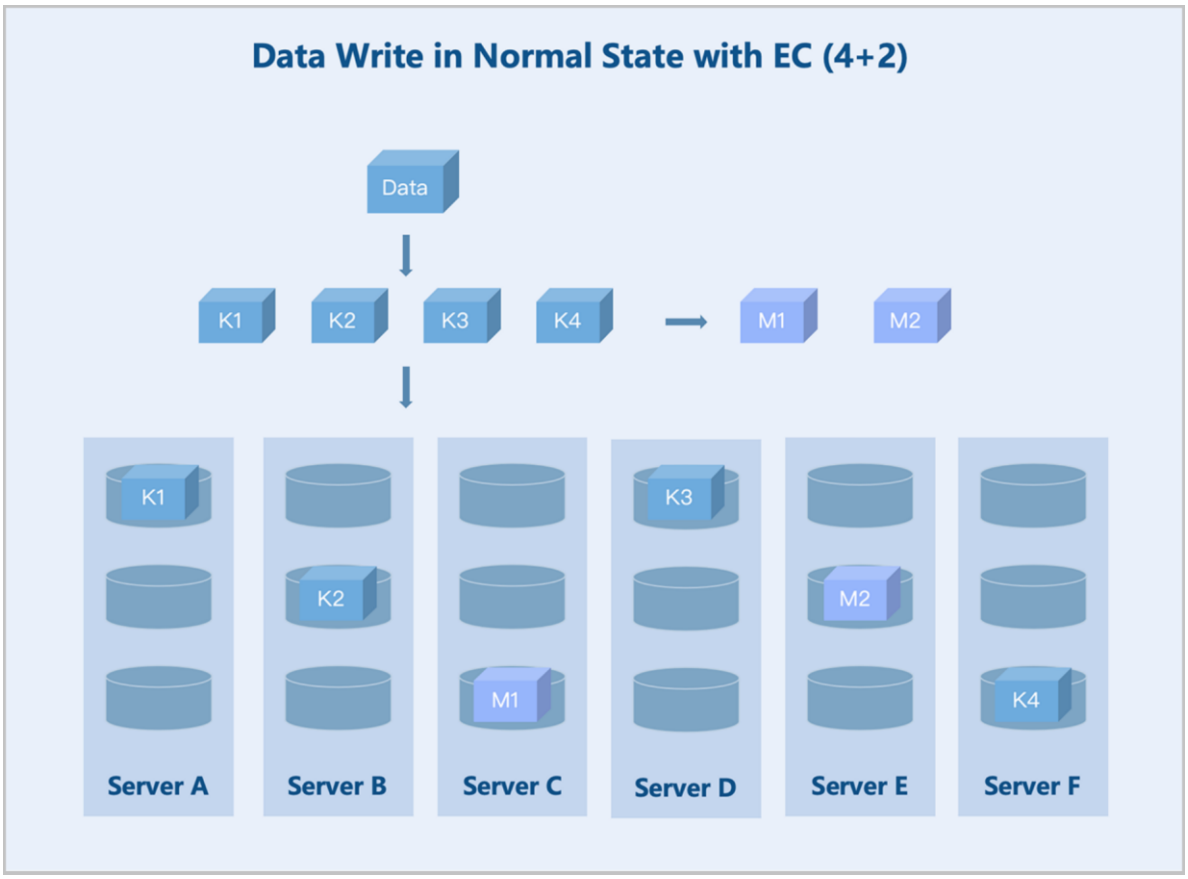
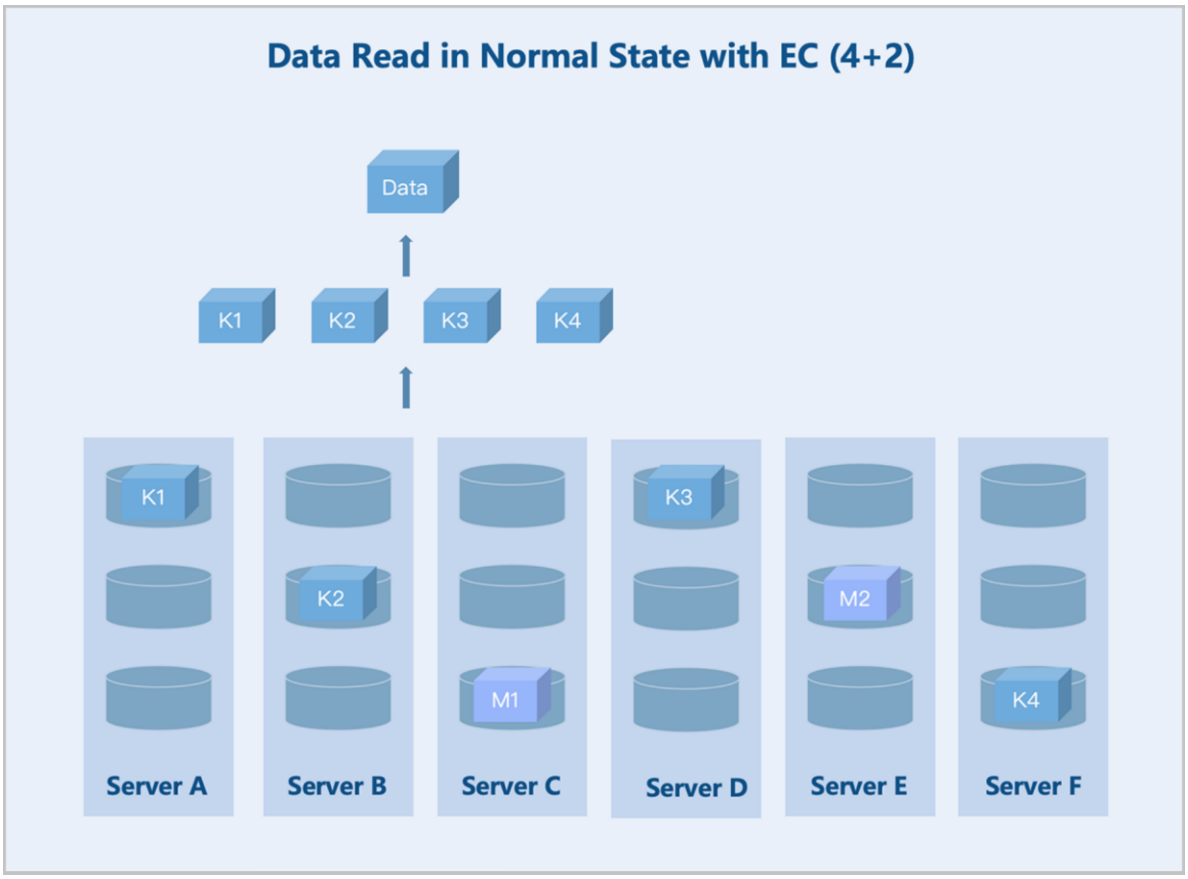


Figure 4-21: Data Read in Normal State with EC (4+2)



- Read/Write Under Failures

  Standard EC (K+M): Taking a server-level 4+2 EC policy as an example: If the number of available servers drops below K+M due to failures, the system stores newly written data on the remaining servers before the recovery. This ensures I/O continuity without reducing the reliability. Once the failed servers recover, the data redundancy policy reverts to K+M. During data read, the system reads data from the remaining healthy servers and recovers the data using the parity algorithm before returning it to the user.

Figure 4-22: Data Write Under Failures with EC (4+2)

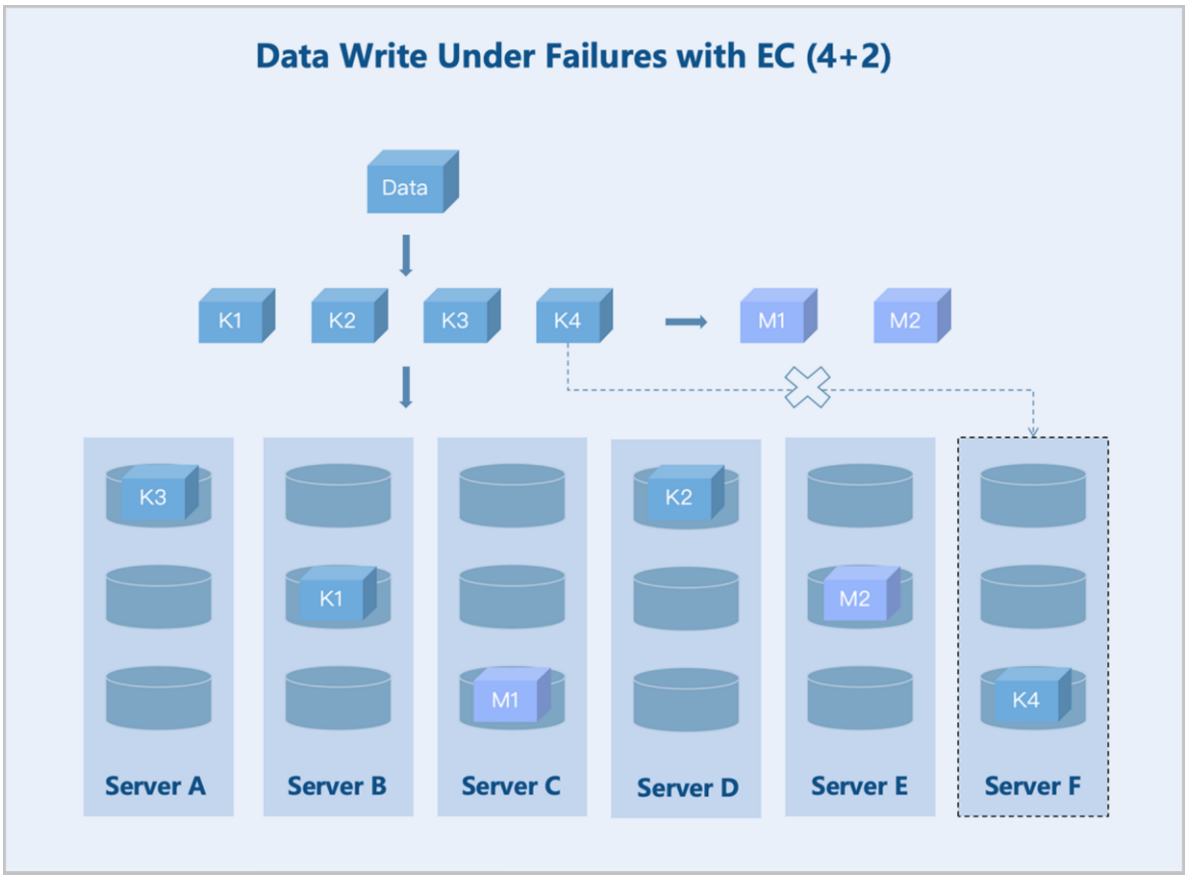
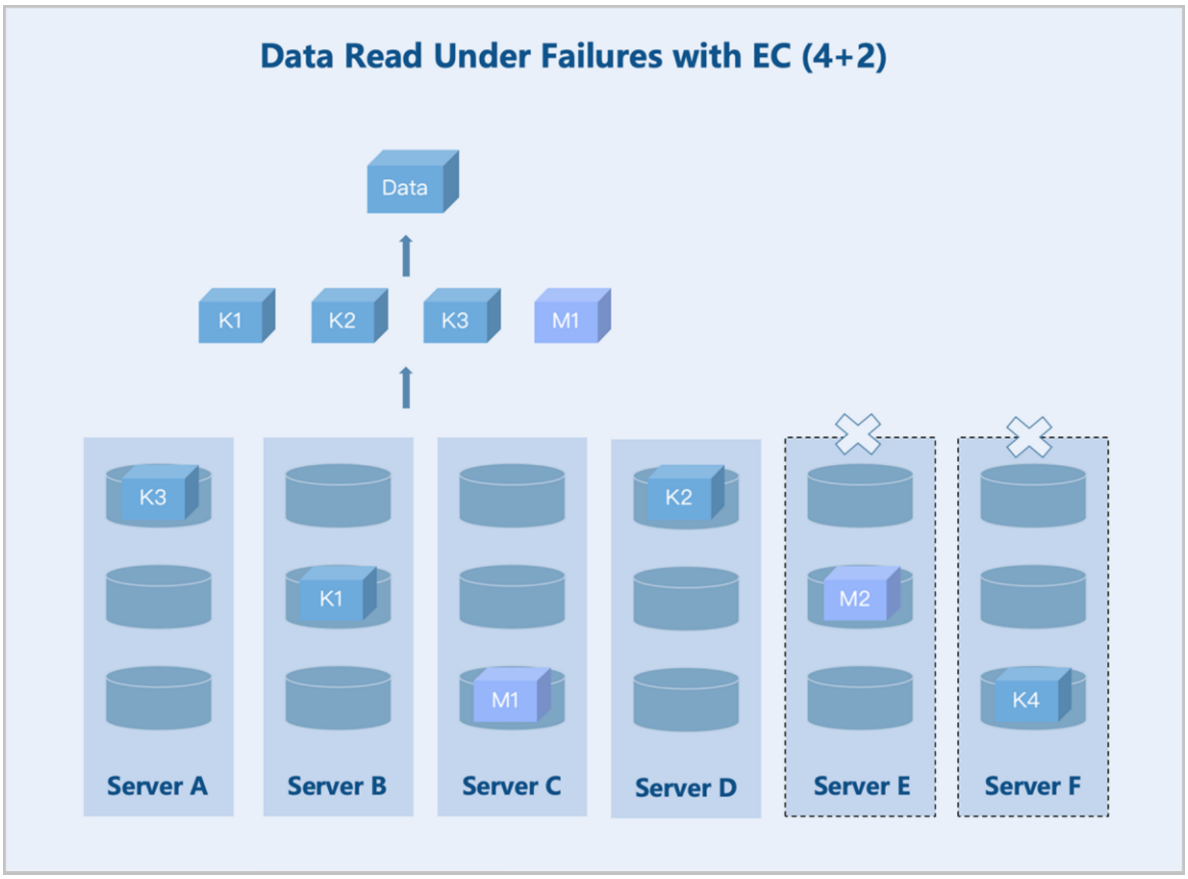


Figure 4-23: Data Read Under Failures with EC (4+2)



Folded EC

Folded EC, also known as sub-node EC, is another common data redundancy technique. Unlike the standard K+M EC configuration, the folded EC typically follows a K+M:B format, where B is usually set to 1. The folded EC maintains high data reliability while delivering improved storage efficiency.

For example, while a standard EC requires a minimum of 6 nodes, as its smallest fault domain is the storage node. A folded EC with 4+2:1 configuration can achieve data redundancy with as few as 3 storage nodes.

Furthermore, the folded EC supports scaling in and out just like standard EC. You can converts a folded EC to a standard EC as long as the fault domain requirements are met.

Read/Write Principles

- Read/Write in Normal State

  Folded EC (K+M:B): Taking a server-level 4+2:1 EC policy as an example: During data write, the system splits the data into 4 equally sized data blocks and generates 2 parity blocks of the same size through the parity algorithm. The system randomly stores these 6 blocks across 5 servers. If any one server fails, data remains accessible. During data read, the system reads data blocks from different data disks on 3 servers, assembles the 4 data blocks into complete data, and returns it to the user.

Figure 4-24: Data Write in Normal State with EC (4+2:1)

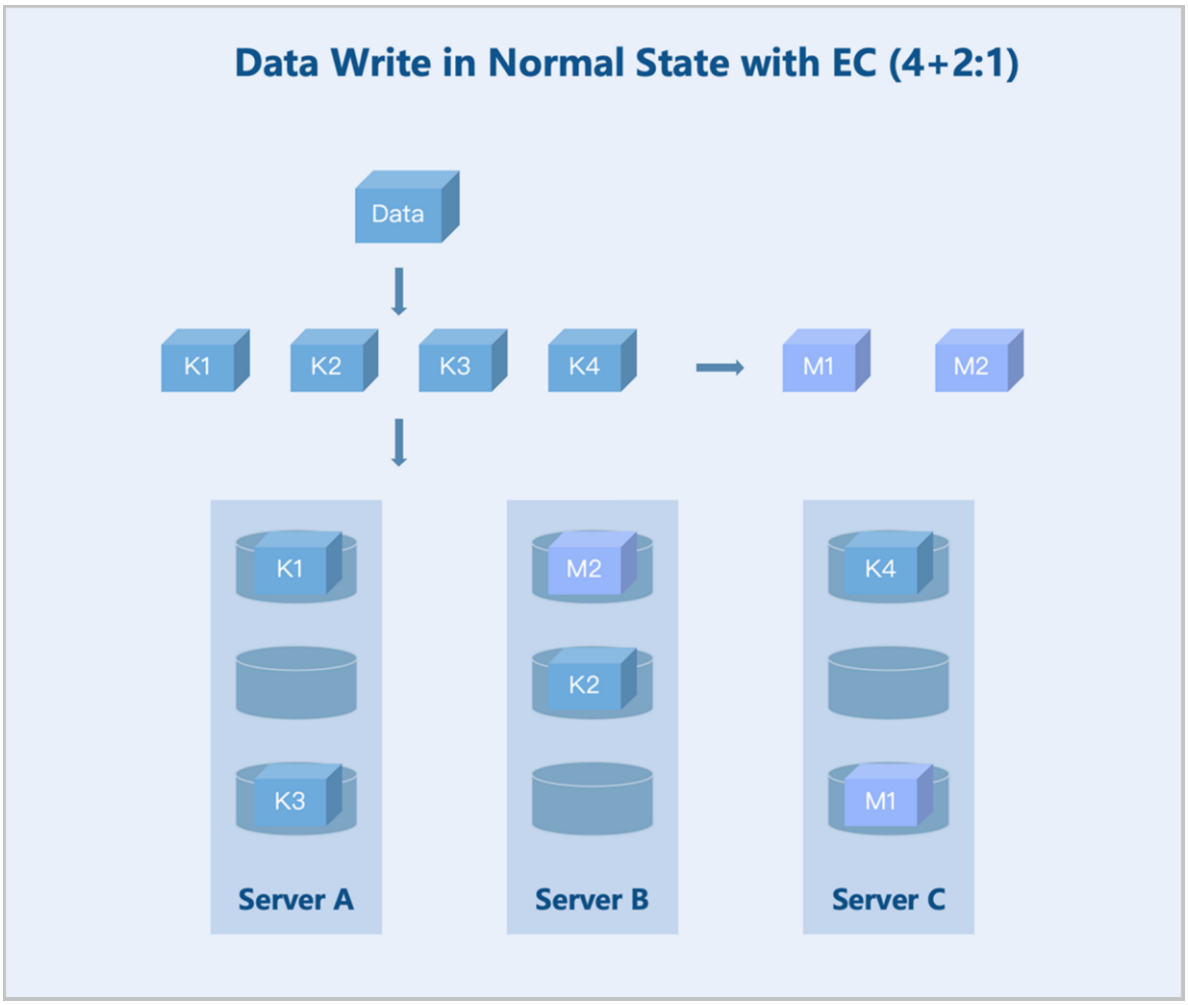
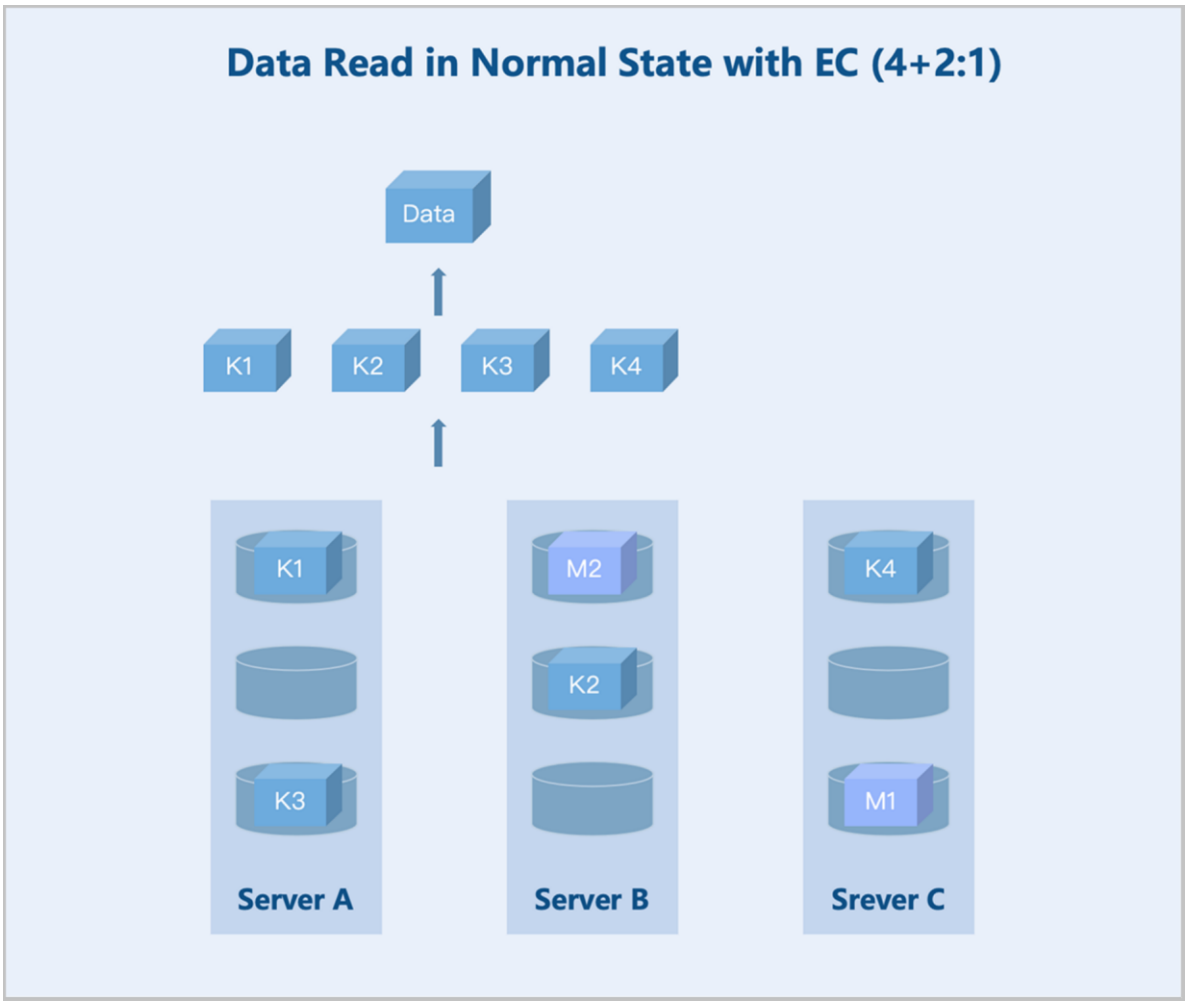


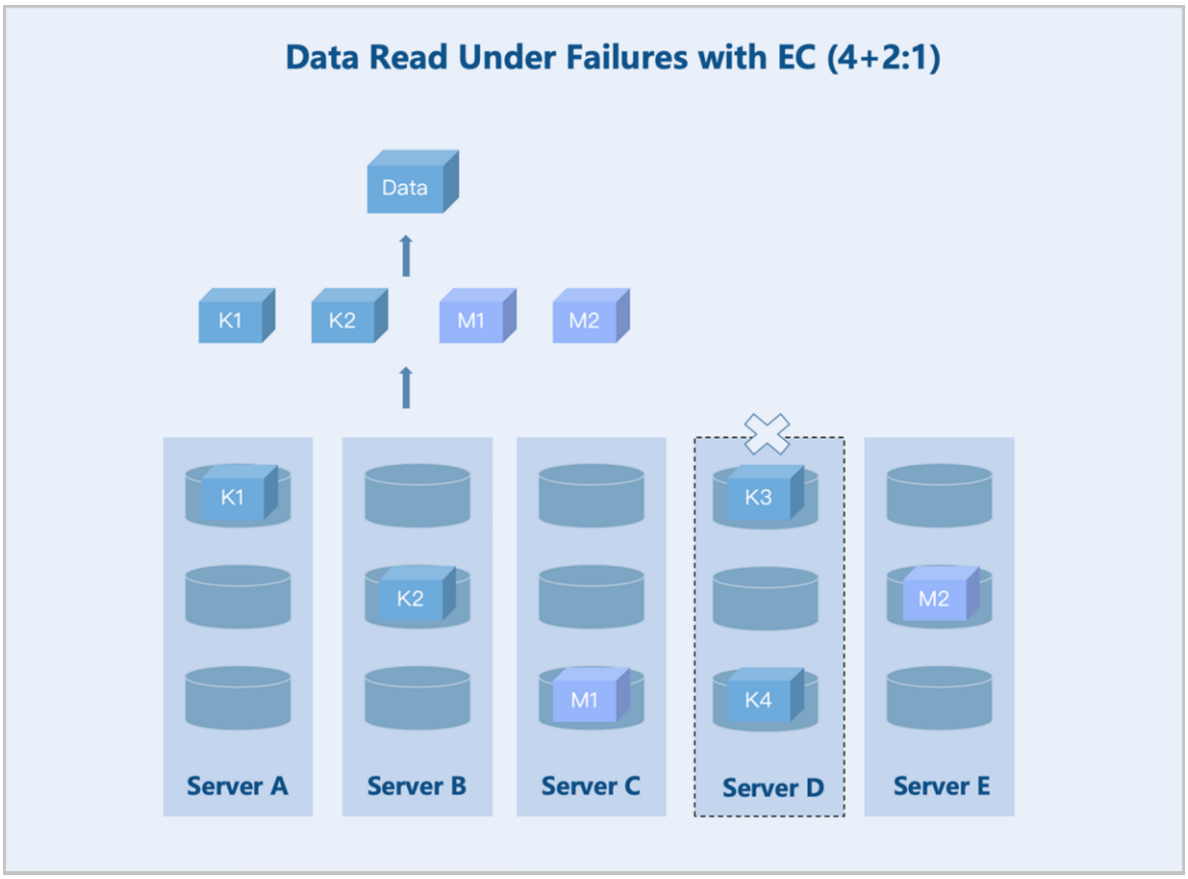
Figure 4-25: Data Read in Normal State with EC (4+2:1)



- Read/Write Under Failures

  Folded EC (K+M:B): Taking a server-level 4+2:1 EC policy as an example: If one server or M hard disks fail, the system continues writing data and parity blocks to the remaining healthy servers according to the K+M configuration. During data read, the system reads data from other healthy servers and recovers the data using the parity algorithm before returning it to the user.

Figure 4-26: Data Read Under Failures with EC (4+2:1)



Replication vs. EC

You can choose between replication and erasure coding (EC) based on actual business requirements, as each has its advantages in different scenarios:

- **Storage Efficiency**: EC holds a significant advantage. For example, a 4+2 EC policy offers approximately 66% effective storage, while 3 replicas policy achieves only 33.3%.
- **Read/Write Performance**: Performance varies significantly in small I/O scenarios. The gap narrows in large I/O scenarios. EC involves data validation during writes, which may introduce write amplification. During reads, performance can be impacted if any of the multiple nodes encountered high latency. In contrast, replication only needs to read one complete copy without any data reassembly

- **Rebuild Performance**: Replication generally outperformed EC in rebuild speed. Replication involves simple data copying without validation, resulting in faster rebuilds. EC rebuild requires reverse parity calculation, demanding more data I/O and higher CPU consumption.
- **Fault Tolerance**: Both policies have strengths and weaknesses. Replication allows up to (number of replicas − 1) simultaneous non-monitoring nodes failures without data loss. For EC, a 4+2 policy allows 2 simultaneous non-monitoring nodes failures without data loss.

4.1.2.2.2.2.3 Fault Domain Isolation

A fault domain is the smallest unit for data distribution in a cluster. When storing data, different replicas or blocks of data are stored in different fault domains. Depending on the configured data redundancy policy, a certain number of fault domain failures are allowed without data loss, ensuring data security. Three data redundancy levels are supported:

- Server-Level: Each server in the cluster acts as a fault domain. Different replicas or blocks of data are stored on different servers.
- Rack-Level: Each rack in the cluster acts as a fault domain. Different replicas or blocks of data are stored in different racks. Recommended for clusters of larger scale with more racks.
- Room-Level: Each room in the cluster acts as a fault domain. Different replicas or blocks of data are stored in different rooms. Recommended for very large clusters spanning multiple rooms.

Figure 4-27: Server-Level Fault Domain

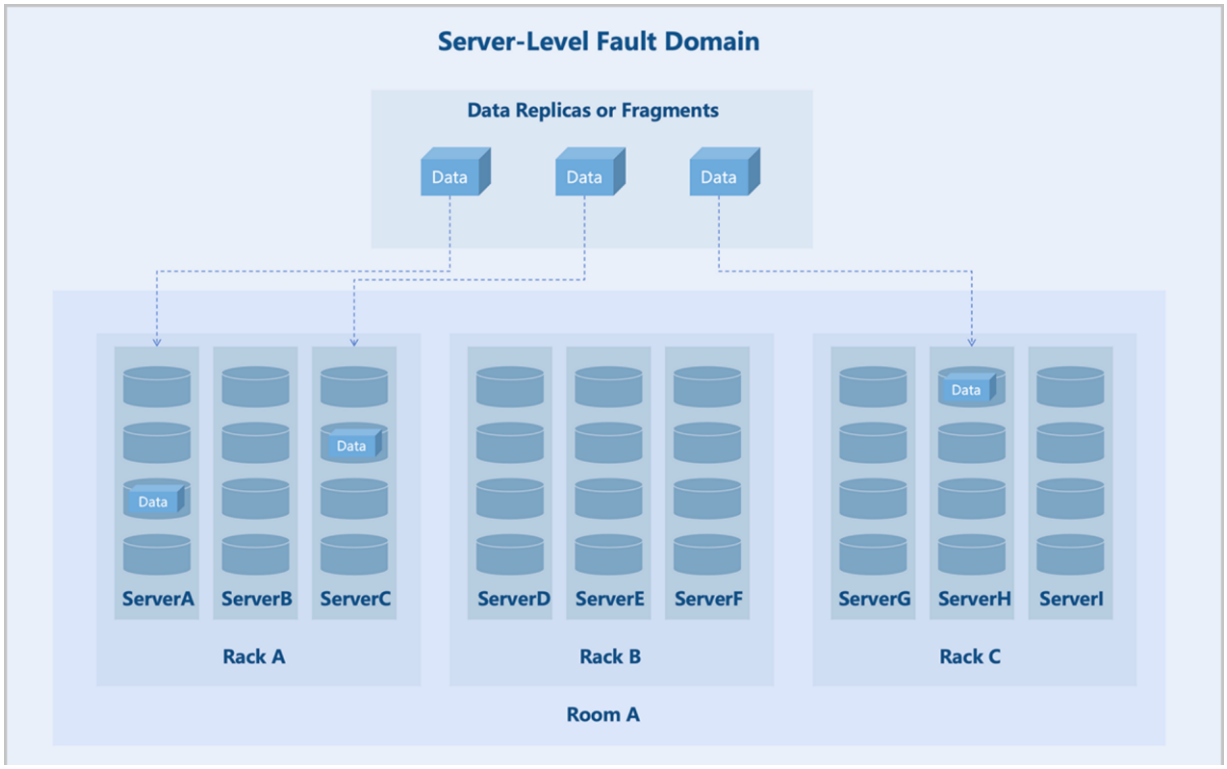


Figure 4-28: Rack-Level Fault Domain

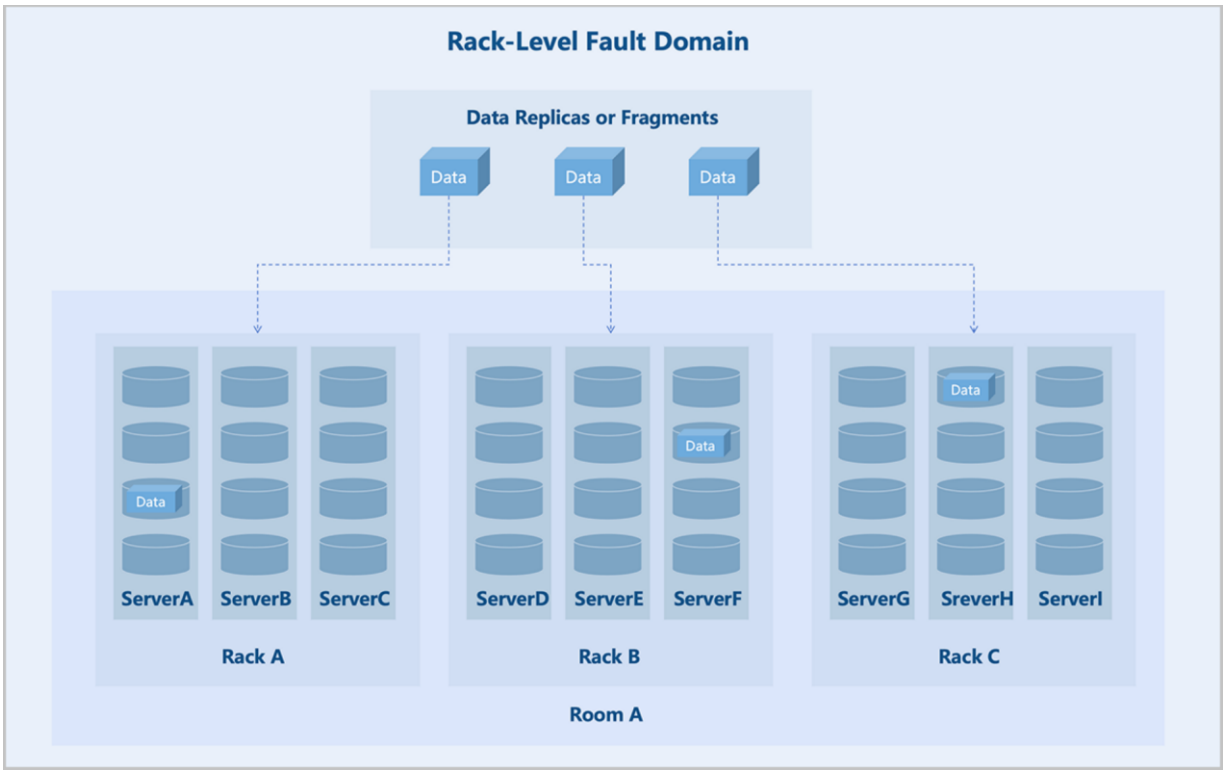
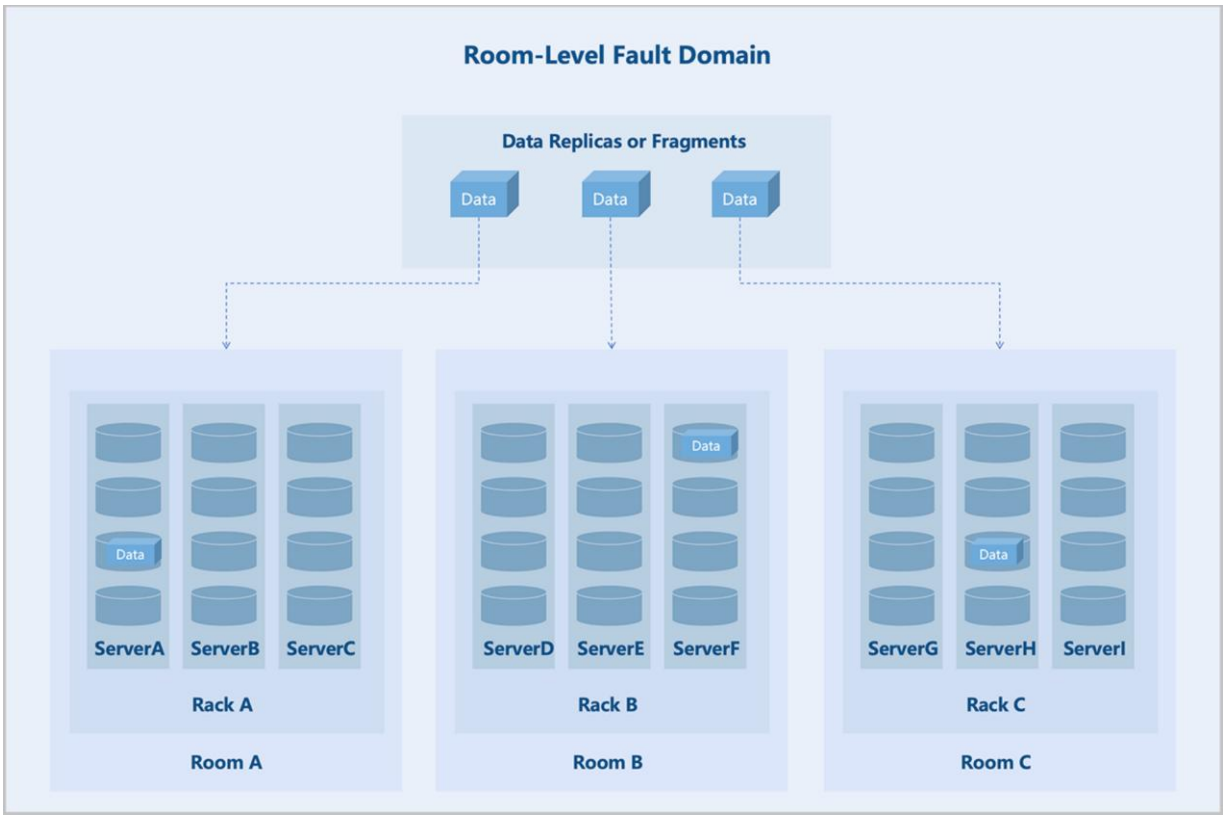


Figure 4-29: Room-Level Fault Domain



Fault domain isolation helps contain the impact of failures to a specific scope, preventing a domino effect and thereby enhancing business continuity.

By leveraging fault domain-aware scaling along with rational storage policies, newly added nodes can form an independent disk pool without requiring data migration. This enables seamless capacity expansion, shielding applications from underlying storage changes and eliminating the need for application-level adjustments traditionally associated with storage updates. As a result, both operational and administrative workloads are significantly reduced, while system reliability and performance are enhanced.

4.1.2.2.2.2.4 Data Consistency Check

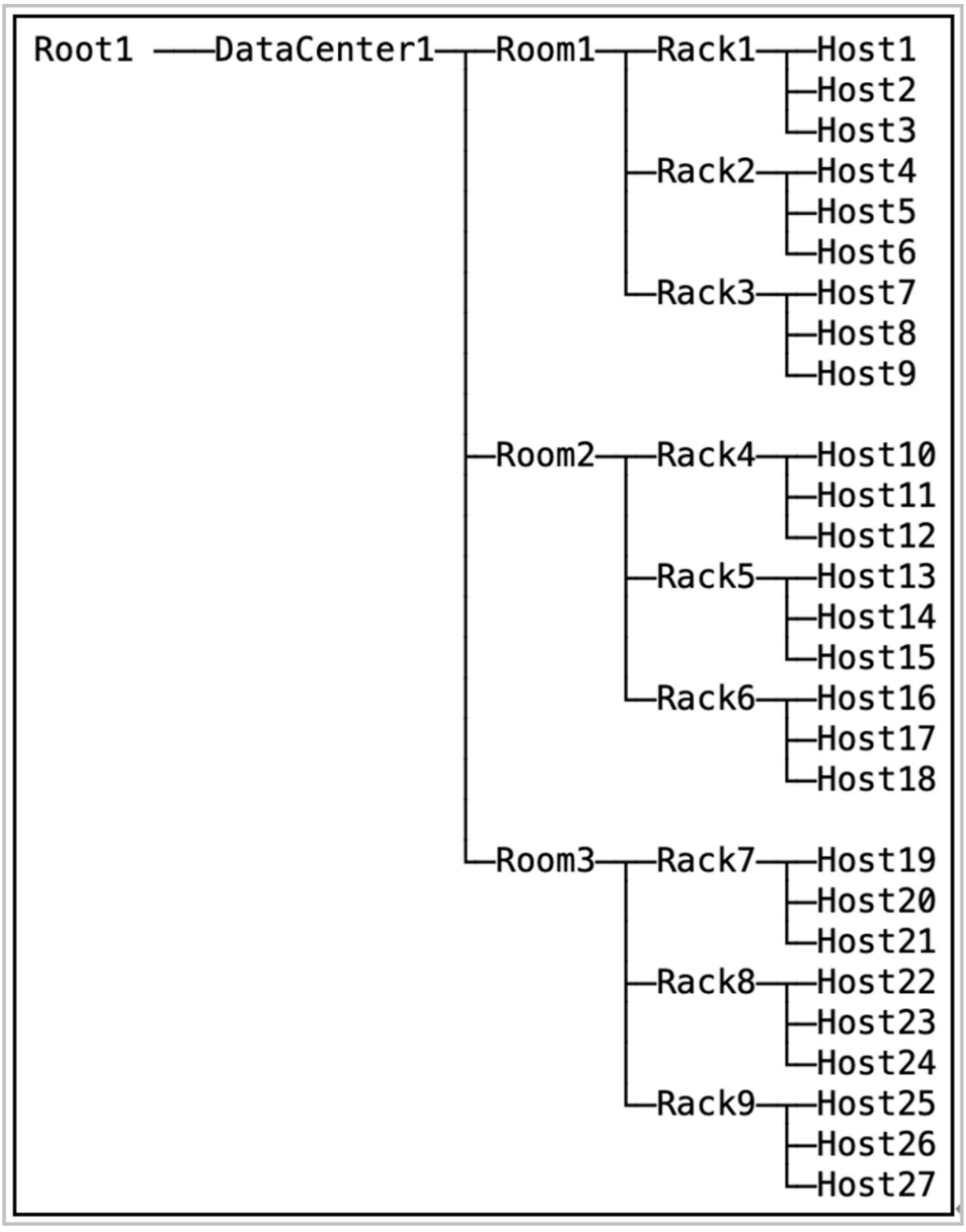
The Self-Developed Distributed Storage uses a Scrub mechanism to perform background scans for data consistency check. Data consistency checks runs periodically and comes in two types: Scrub and Deep-Scrub.

- Scrub Check: Focuses on metadata. It completes quickly and runs frequently. It is recommended to perform Scrub daily. You can customize the schedule.
- Deep-Scrub Check: Focuses on data. It takes longer to complete and may impact I/O performance. It is recommended to run during off-peak hours. An alert is triggered if no Deep-Scrub has been completed within 30 days.

4.1.2.2.2.2.5 Cluster Hardware Topology

The cluster topology provides a visual representation of the actual deployment of the cluster's physical resources. The topology includes logical entities such as data centers, rooms, racks, and servers, organized hierarchically in a tree structure to illustrate the distribution relationship from rooms down to servers. Each tree structure has a root node, and the cluster topology supports multiple root nodes. After planning the topology, you can select the corresponding level of data redundancy policy when creating a storage pool.

Figure 4-30: Topology Hierarchy



You can plan a topology through the Web interface. The following table lists the quantity limits for each topology object.

Topology Object	Quantity Range
Data Center	0~2
Room	0~100 (Per Data Center)
Rack	0~100 (Per Room)
Server	0~20 (Per Rack)

4.1.2.2.2.3 Convenient O&M

4.1.2.2.2.3.1 Multiple Resource Pools

The Self-Developed Distributed Storage supports multiple resource pools, helping you utilize storage media with different performance characteristics and achieve fault isolation.

Each resource pool has distinct attributes and performance, such as the number of replicas, data redundancy level, and storage media. You can flexibly allocate and manage resources based on actual requirements to improve storage efficiency and performance.

Resource pools are isolated from each other. You can implement data isolation management across multiple pools. Besides, failures in a single resource pool does not affect other resource pools, effectively safeguarding data security and storage reliability.

4.1.2.2.2.3.2 Hard Disk Light

The Self-Developed Distributed Storage provides a visual interface to light on the hard disk for rapid locating. When you need to maintain or replace a disk, click Disk Light in the UI. The corresponding disk's LED indicator lights up in the physical environment, guiding you to quickly and accurately identify the device, thereby improving O&M efficiency.

4.1.2.2.2.3.3 Disk S.M.A.R.T. Monitoring

The Self-Developed Distributed Storage supports S.M.A.R.T. (Self-Monitoring, Analysis, and Reporting Technology) monitoring to track the health status, temperature, firmware, and total bytes written of disks. Upper-layer services trigger relevant alerts based on I/O errors and disk status information returned by the S.M.A.R.T. data.

4.1.2.2.2.3.4 Data Disk Maintenance Mode

The Self-Developed Distributed Storage supports maintenance mode for data disks. To maintain servers or disks, you can put the corresponding disks into maintenance mode on the

UI. A disk in this mode stops all services and data access, and the data on the disk does not undergo

rebalancing.

4.1.2.2.2.3.5 Automatic Failure Detection and Alerting

The Self-Developed Distributed Storage supports automatic failure detection and alerting. This mechanism monitors the storage platform system and individual storage servers. Upon detecting a failure, the system automatically sends alert messages to the platform. You can also add email endpoints to receive alarms, enabling timely response and recovery.

When a failure occurs, the system supports automatic service restart and data migration. This maximizes data reliability and availability, forming a highly reliable and highly available distributed storage system.

4.1.2.2.2.3.6 Data Rebalancing

The Self-Developed Distributed Storage supports data rebalancing to evenly distribute data across data disks under all storage servers in the cluster. This enhances storage system performance and reliability.

Automatic Data Rebalancing:

- Based on storage pool configurations and storage server load, the system automatically migrates data from overloaded nodes to those with lower load to achieve load balancing.
- When a server fails or a new server is added, the system automatically migrates data to maintain consistency and reliability.

Manual Data Rebalancing:

The system also supports manual data rebalancing. You can manually initiate rebalancing operations based on the actual data distribution.

4.1.3 Network Virtualization

4.1.3.1 Overview

Network virtualization is a technology that abstracts network capabilities away from dedicated network hardware. The underlying hardware only needs to provide basic packet forwarding services. Network virtualization provides various network services, including data switching, routing, security groups, and more, achieving an experience comparable to a physical network.

NexaVM NSSV abstracts the network model into distributed switches and distributed port groups. A distributed switch spans multiple hosts within a data center, providing the ability to deploy, manage, and monitor virtual networks across the entire data center. A distributed port group is a collection of virtual ports on a distributed switch. Virtual machines in the same port group share a uniform network configuration. This enables virtual machines to migrate between different hosts. Additionally, the platform provides a distributed DHCP service on distributed port groups. You can also configure DNS for virtual machines. These features meet the requirements of different network scenarios in the data center.

4.1.3.2 Key Features

4.1.3.2.1 Distributed Switch

Switches identify hosts using MAC addresses or IP addresses to forward and control network traffic between hosts, and between hosts and external networks. In traditional data centers, host locations are fixed, and positional relationship between switches and hosts remain constant, meaning switch configurations rarely require changes.

In virtualized data centers, switches often face new challenges, such as:

- You cannot pre-determine virtual machine MAC addresses or IP addresses in advance.
- During virtual machine migration, the VM configuration on physical switch ports also needs to migrate, which may cause migration failures.
- When source and destination virtual machines are on the same host, network traffic might bypass the switch, preventing the switch from performing traffic control.

NexaVM NSSV introduces distributed switches that provide traffic forwarding and control between virtual machines, and between virtual machines and external networks, offering you convenient, flexible, and feature-rich network capabilities.

A distributed switch is a logical switch distributed across each host, consisting of virtual switches, virtual machine NICs, and host NICs.

- The virtual switch forwards network traffic based on MAC addresses.
- Virtual machine NICs connect virtual machines to the virtual switch, providing data channels for virtual machine traffic.
- Host NICs serve as uplink ports for the virtual switch, forwarding cross-host virtual machine traffic to physical switches.

4.1.3.2.2 Security Group

In traditional data centers, networks divide into trusted zones, DMZ zones, and untrusted zones. Perimeter firewalls enforce traffic control to ensure network security. In virtualized data centers, perimeter firewalls may struggle to handle scenarios such as network isolation between different tenants and access control between different services within the same tenant flexibly. Therefore, NexaVM NSSV introduces a new component: security group. A security group is a distributed firewall focused on controlling east-west traffic and supports inbound and outbound traffic control at the virtual machine NIC level.

A security group consists of two logical sets:

- The set of security group rules: supports the addition, deletion, and modification of rules to control rule actions.
- The set of associated NICs: supports the binding of virtual machine NICs to apply security group rules to them.

Characteristics of security groups:

- Default policies: If no rules are added, except communication between group members, the security group denies all inbound traffic by default, and allows all outbound traffic.
- Rule flexibility: Rules support on-demand modification, including source IP, destination IP, destination port, protocol type, and priority.
- Dynamic priority adjustment: When adding a rule, you can insert it at a specific priority. When removing a rule, priorities automatically adjust to remain consecutive.
- Multiple security groups per NIC: A virtual machine NIC supports associating to multiple security groups. You can dynamically adjust the priority of these groups. By default, rules in the first-associated security group take effect first.

How It Works

Security groups have the following key points:

- Composition of a security group:
  - Source: Supports source data (for inbound direction) and destination data (for outbound direction).
  - Address Type: Supports IPv4 and IPv6.
  - Protocol Type and Port: Supports ICMP, TCP, UDP, and more.
  - Action: Deny or allow.

- Priority of security group rules:
  - Rule priority is a consecutive and unique (priority 0 indicates the highest priority by default). Lower priority numbers indicate higher precedence.
  - By default, newly added rules have the lowest priority. When inserting a rule at a specified priority, subsequent rule priorities adjust automatically to ensure uniqueness.
  - When traffic passes through a virtual machine NIC, matching starts from the rule with the highest priority. If a match succeeds, the rule action executes. Otherwise, matching continues with the next rule.
- Import and export of security group rules:
  - Existing rules support one-click export.
  - Exported rules supports re-importing into other security groups with rule validation provided during import.
- Associating multiple security groups to a NIC:
  - A virtual machine NIC allows associating to multiple security groups. The associated groups support priority ordering. A lower number indicates a higher group priority.
  - When traffic passes through the virtual machine NIC, matching starts from the security group with the highest priority. If a rule within that group match succeeds, the rule action executes. Otherwise, matching proceeds to the next security group.
- NIC default policy:
  - By default, the NIC's default inbound policy is Allow, and the default outbound policy is Deny . These default policies support modification.
  - When traffic does not match any security group rule, the NIC's default policy executes.

Applications

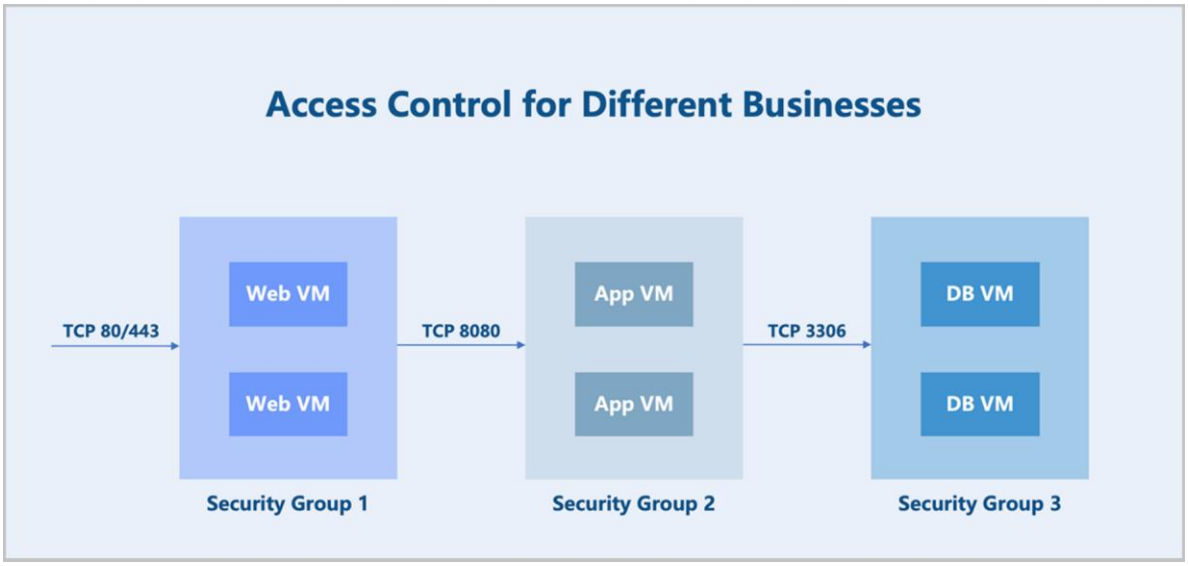
1. Access control among different businesses.

   Take a typical web application as an example.

   - Allow any IP address to access TCP ports 80 and 443 on the Web virtual machines.
   - Allow the Web virtual machines to access TCP port 8080 on the App virtual machines.
   - Allow the App virtual machines to access TCP port 3306 on the DB virtual machines.

   Create separate security groups for the three business types (Web, App, DB) and configure corresponding rules to achieve access control among the different businesses.

Figure 4-31: Access Control for Different Businesses



2. Coexistence of multiple businesses.

   In addition to providing service capabilities, the above business virtual machines also need to allow operational control (assumed to use the SSH protocol). There are two implementation options:

   - Option 1: Add a new rule to each of the three security groups.
   - Option 2: Create a new security group and associate it to the NICs of all the above business virtual machines.

   NexaVM NSSV supports associating multiple security groups to a virtual machine NIC. Option 2 is the preferred method for its better flexibility and controllability. You can configure two security groups for each business virtual machine NIC mentioned above and adjust the security group priority and default rules as needed.

3. Disabling communication within a security group.

   By default, virtual machines within the same security group can communicate with each other without any rule restrictions. In certain scenarios where communication between virtual machines within the same group must be prohibited, you can disable the default rule: **Allow for communication within the group**. Note that this default rule cannot be deleted.

4.1.4 Virtual Resources Management

4.1.4.1 Virtual Machine Management

4.1.4.1.1 VM Scheduling Policy

A VM scheduling policy is a resource orchestration policy based on which virtual machines are assigned hosts to achieve the high performance and high availability of businesses. You can add virtual machines to a VM scheduling group and associate a scheduling policy to this group to implement VM scheduling.

Function Principles

NexaVM NSSV supports adding a virtual machine to a virtual machine scheduling group and binding scheduling policies to the group to achieve virtual machine scheduling.

- If a mutually exclusive virtual machine or aggregated virtual machine policy is bound, no host scheduling group needs to be specified, and the virtual machine is assigned to a host according to the policy and its execution mechanism.
- If a virtual machine host affinity or virtual machine mutually exclusive host scheduling policy is bound, the corresponding host scheduling group must be specified, and the virtual machine is assigned to a host according to the policy and its execution mechanism.

The following explains the working principles of the four types of scheduling policies through four scenarios:

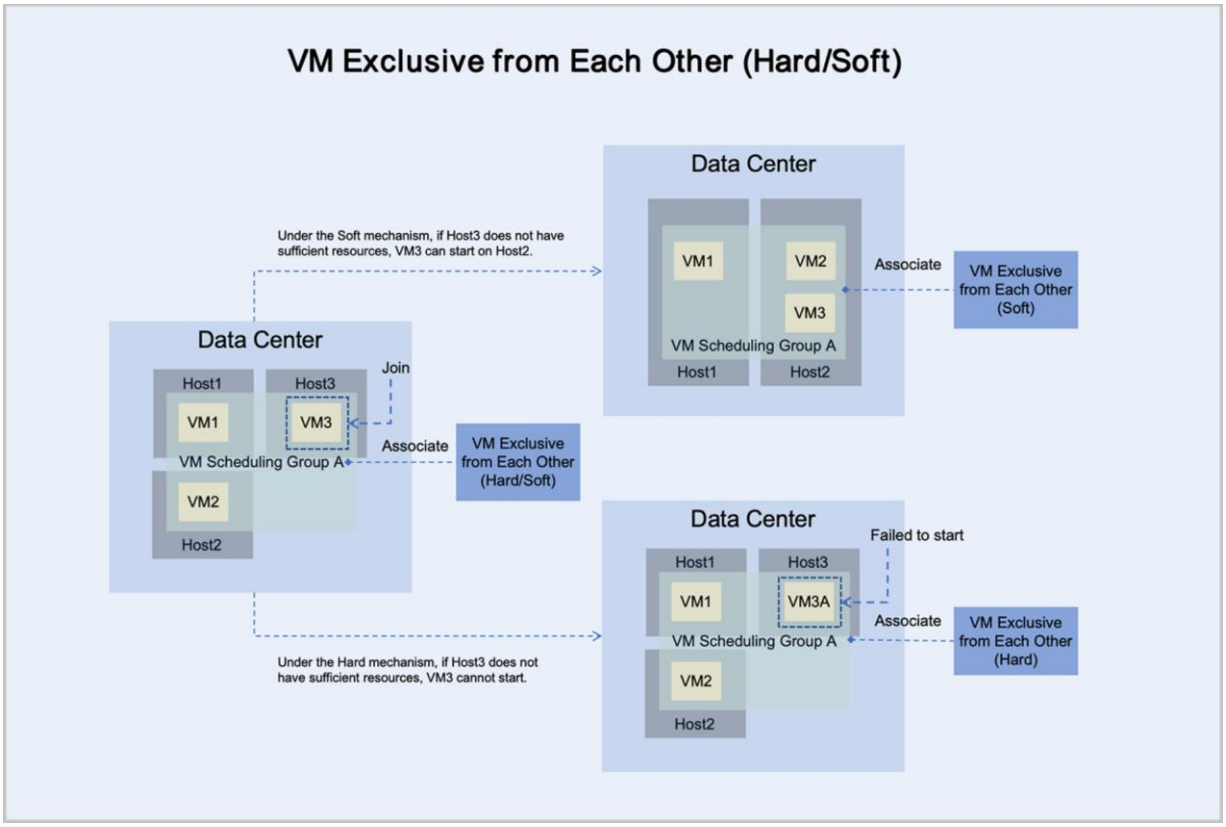
Scenario 1: Assume there are three hosts Host1, Host2, and Host3 in the data center. Virtual machine scheduling group A is bound to the **VM Exclusive from Each Other** scheduling policy, and virtual machines VM1 and VM2 have joined this scheduling group and are running on hosts Host1 and Host2, respectively. At this point, virtual machine VM3 joins this scheduling group.

Under different execution mechanisms, the behavior of virtual machine VM3 is as follows:

- Under the mandatory mechanism, virtual machine VM3 adheres to the principle of mandatory mutual exclusion with other virtual machines in the group:
  - If Host3 has sufficient resources, it can normally start and run on Host3.
  - If Host3 does not have sufficient resources, it cannot start on Host3.

- Under the preferred mechanism, virtual machine VM3 adheres to the principle of trying to mutually exclude other virtual machines in the group, prioritizing starting on Host3:
  - If Host3 has sufficient resources, it can normally start and run on Host3.
  - If Host3 does not have sufficient resources, VM3 can attempt to start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

Figure 4-32: VM Exclusive from Each Other (Hard/Soft)

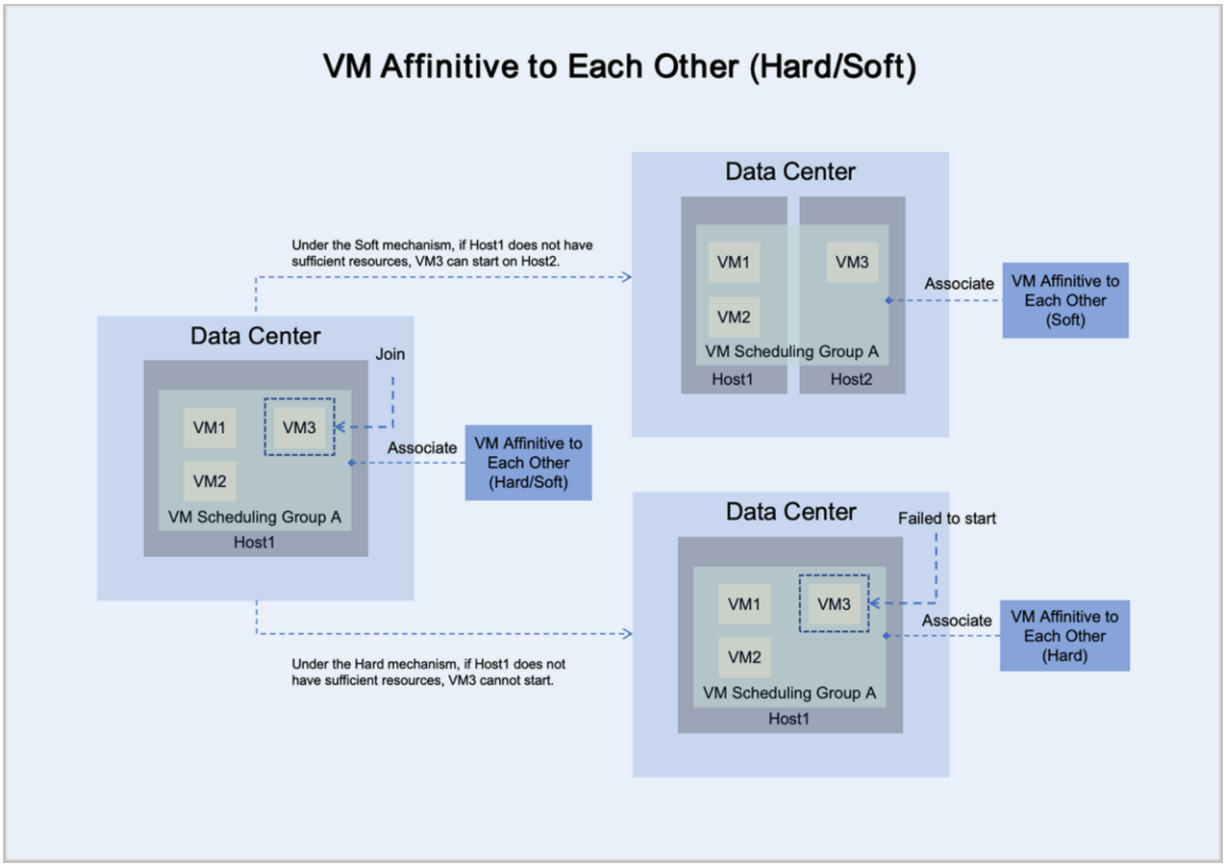


Scenario 2: Assume there are two hosts Host1 and Host2 in the data center. Virtual machine scheduling group A is bound to the **VM Affinitive to Each Other** scheduling policy, and virtual machines VM1 and VM2 have joined this scheduling group and are running on host Host1. At this point, virtual machine VM3 joins this scheduling group. Under different execution mechanisms, the behavior of virtual machine VM3 is as follows:

- Under the mandatory mechanism, virtual machine VM3 adheres to the principle of mandatory aggregation with other virtual machines in the group:
  - If Host1 has sufficient resources, it can normally start and run on Host1.
  - If Host1 does not have sufficient resources, it cannot start on Host1.

- Under the preferred mechanism, virtual machine VM3 adheres to the principle of trying to aggregate with other virtual machines in the group, prioritizing starting on Host1:
  - If Host1 has sufficient resources, it can normally start and run on Host1.
  - If Host1 does not have sufficient resources, VM3 can attempt to start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

Figure 4-33: VM Affinitive to Each Other (Hard/Soft)



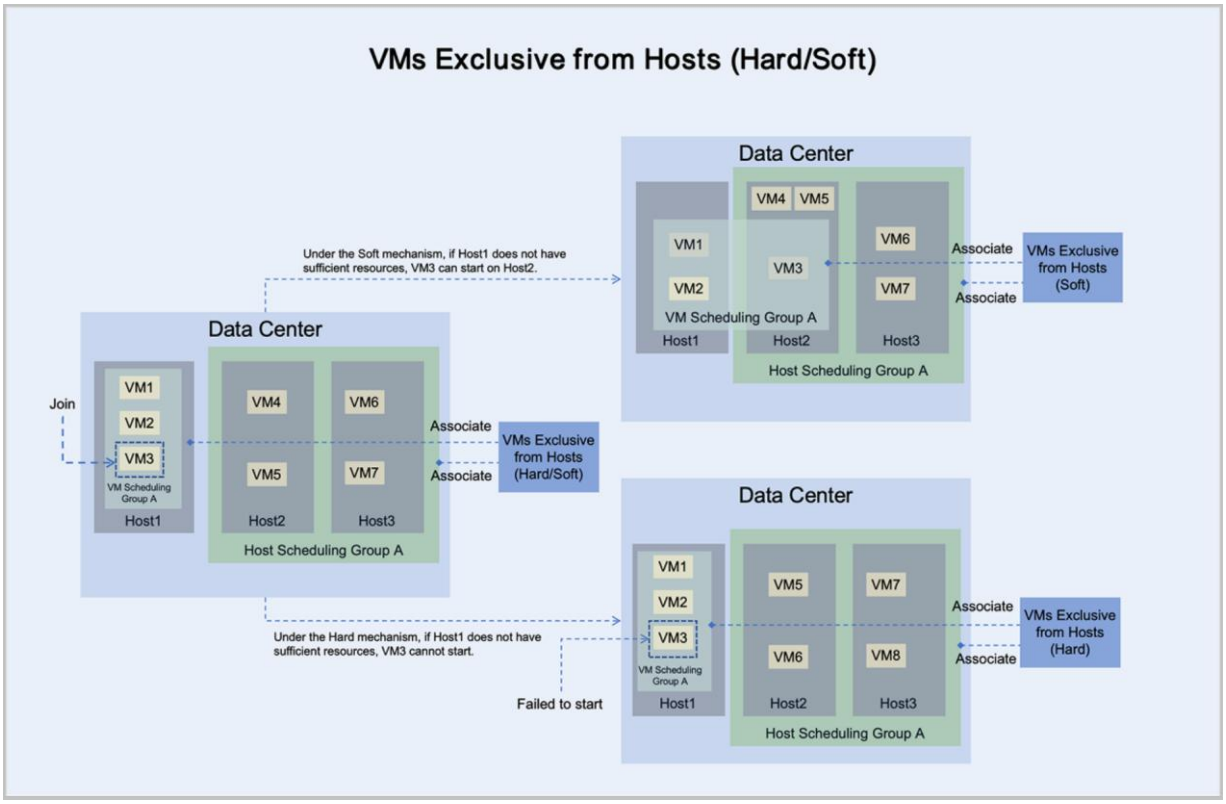
Scenario 3: Assume there are three hosts Host1, Host2, and Host3 in the data center. Virtual machine scheduling group A is bound to the **VMs Exclusive from Hosts** scheduling policy, and virtual machines VM1 and VM2 have joined this scheduling group and are running on host Host1. Host scheduling group A is also bound to the **VMs Exclusive from Hosts** scheduling policy, and hosts Host2 and Host3 have joined this scheduling group, each running two virtual machines. At this point, virtual machine VM3 joins virtual machine scheduling group A. Under different execution mechanisms, the behavior of virtual machine VM3 is as follows:

- Under the mandatory mechanism, virtual machine VM3 adheres to the principle of mandatory mutual exclusion with hosts in host scheduling group A:

- If Host1 has sufficient resources, it can normally start and run on Host1.
- If Host1 does not have sufficient resources, it cannot start on Host1.

- Under the preferred mechanism, virtual machine VM3 adheres to the principle of trying to mutually exclude hosts in host scheduling group A, prioritizing starting on Host1:
  - If Host1 has sufficient resources, it can normally start and run on Host1.
  - If Host1 does not have sufficient resources, VM3 can attempt to start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

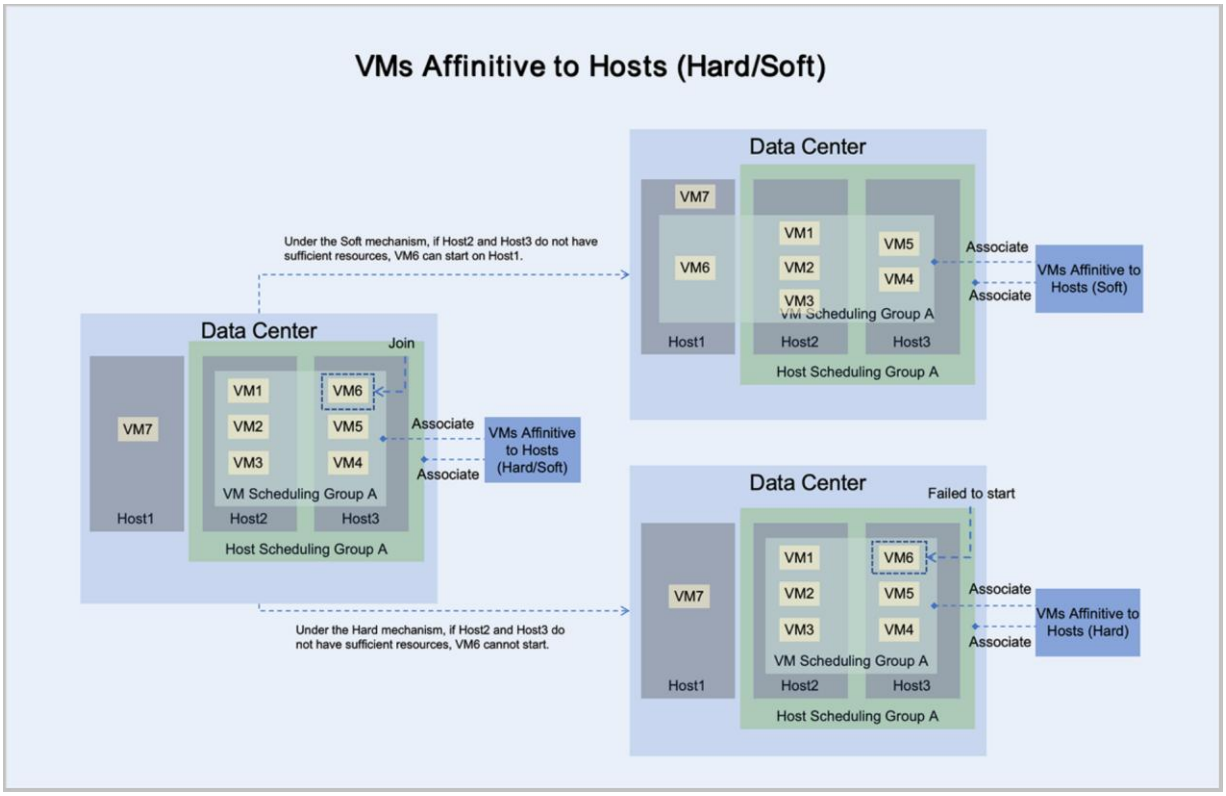
Figure 4-34: VMs Exclusive from Hosts (Hard/Soft)



Scenario 4: Assume there are three hosts Host1, Host2, and Host3 in the data center. Virtual machine scheduling group A is bound to the **VMs Affinitive to Hosts** scheduling policy, and virtual machines VM1 through VM5 have joined this scheduling group and are running on hosts Host2 and Host3. Host scheduling group A is also bound to the **VMs Affinitive to Hosts** scheduling policy, and hosts Host2 and Host3 have joined this scheduling group. At this point, virtual machine VM6 joins virtual machine scheduling group A. Under different execution mechanisms, the behavior of virtual machine VM6 is as follows:

- Under the mandatory mechanism, virtual machine VM6 adheres to the principle of mandatory aggregation with hosts in host scheduling group A:
  - If Host2 or Host3 has sufficient resources, it can normally start and run on Host2 or Host3.
  - If Host2 and Host3 do not have sufficient resources, it cannot start on Host2 or Host3.
- Under the preferred mechanism, virtual machine VM6 adheres to the principle of trying to aggregate with hosts in host scheduling group A, prioritizing starting on Host2 or Host3:
  - If Host2 or Host3 has sufficient resources, it can normally start and run on Host2 or Host3.
  - If Host2 and Host3 do not have sufficient resources, VM6 can attempt to start on another host with sufficient resources. In this scenario, VM6 starts and runs on Host1.

Figure 4-35: VMs Affinitive to Hosts (Hard/Soft)



4.1.4.1.2 Virtual Machine Clone

NexaVM NSSV provides two cloning methods to meet different service needs: full clone and instant full clone.

Full Clone

Full clones do not share data with the source VM. Instead, full clones create a new virtual machine by first converting the source VM into an image file. The source VM image is a complete file containing all necessary files and configurations for the VM and is stored in the image storage. This image file is then pushed to the data storage to serve as an image cache for the cloned virtual machine, thereby creating a new virtual machine. Full cloned VMs are not restricted by the data storage. The cloned VMs can be configured to start on a data storage different from that of the source VM.

Full clone virtual machines are completely isolated from and independent of the source VM. This independence ensures that the new VM's performance is in no way affected. Full clones typically require more storage. However, NexaVM NSSV optimizes storage during batch full cloning operations. The image cache for new VMs is stored only once on the data storage. This avoids excessive consumption of data storage resources and also accelerates the startup of batch-cloned virtual machines.

Instant Full Clone

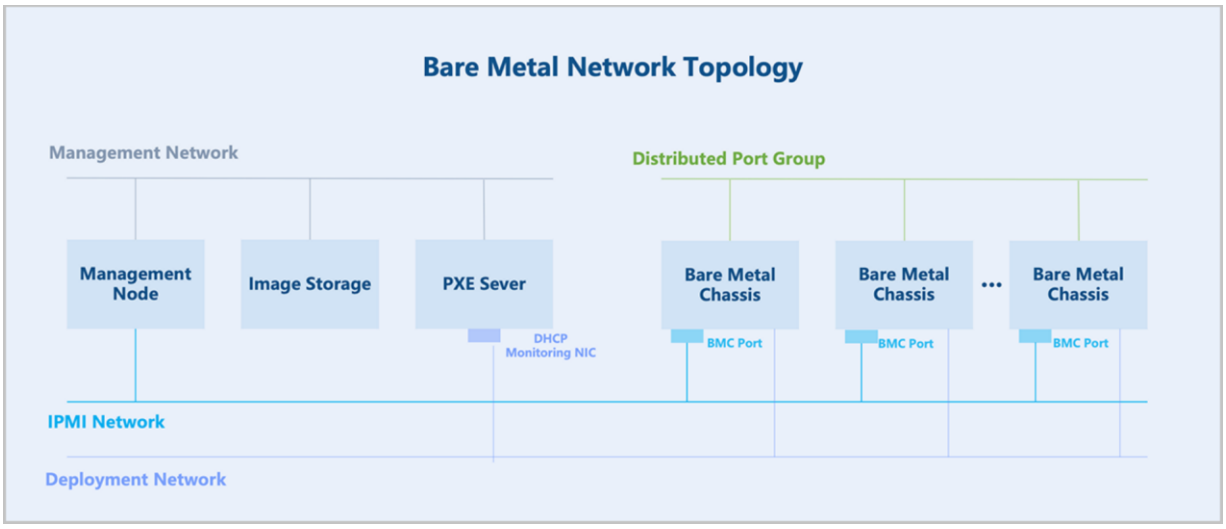
When you perform instant full cloning, the process initially uses linked cloning technology to create a new VM that depends on the source VM, ensuring fast VM provisioning. Subsequently, a background task initiates a snapshot merge operation that completely separates the cloned VM's data from the source VM. As a result, VMs created through fast full cloning benefit from rapid boot times and ultimately achieve complete data independence, with no performance impact after the cloning process completes.

4.1.4.2 Bare Metal Management

The network traffic model for bare metal management includes the management network, distributed port group, IPMI network, and deployment network.

- Management Network: Manages hardware resources on NexaVM NSSV.
- Distributed Port Group: Serves as the service network for bare metal instances to provide application services.
- IPMI Network: Used by the management node to perform remote operations on bare metal chassis and bare metal instances, such as powering on/off, rebooting, and retrieving hardware information.
- Deployment Network: Used by the PXE sever to assign IP addresses via the DHCP service and to transfer images via the TFTP service.

Figure 4-36: Bare Metal Network Topology



The core of the bare metal management: hardware information retrieval and unattended deployment of bare metal chassis.

Bare metal management workflow:

1. The management node directs the bare metal chassis to perform a PXE boot via the PXE server.
2. The PXE server encapsulates DHCP, TFTP, and image storage services. After the bare metal chassis boots via PXE, it obtains an IP address through DHCP, downloads **pxelinux.0** and

   boot files from the TFTP server, loads the kernel into memory for execution, and boots into a LiveCD system.
3. In the LiveCD system, a detection script runs and reports the hardware information of the bare metal chassis back to the management node.
4. Based on the returned hardware information, a preconfigured template is applied to the bare metal chassis. This template includes partition information, NIC bonding, IP address, and more.
5. Select an OS ISO for installation to deploy the bare metal instance.
6. The bare metal chassis is rebooted for a PXE boot. The PXE server pre-downloads the target OS ISO, and the bare metal chassis performs an unattended deployment according to the

preconfigured template. After deployment, the chassis automatically configures the NIC and other settings based on the preconfigured template, completing the bare metal instance configuration.

7. For better O&M of bare metal instances, the PXE server supports deploying a bare metal monitoring service. This service enables real-time monitoring of internal data within the bare metal instances, including metrics for CPU, memory, disk capacity, disk I/O, NICs, and more.

4.2 Data Protection

4.2.1 Snapshot Management

NexaVM NSSV supports both Redirect-On-Write (ROW) and Copy-On-Write (COW) snapshot mechanisms.

- For centralized storage: Local storage, NFS, and SAN storage use QCOW2 external snapshot, which is a type of ROW snapshot mechanism.
- For distributed storage: Ceph enterprise edition use ROW snapshots. Self-Developed Distributed Storage uses COW snapshots.

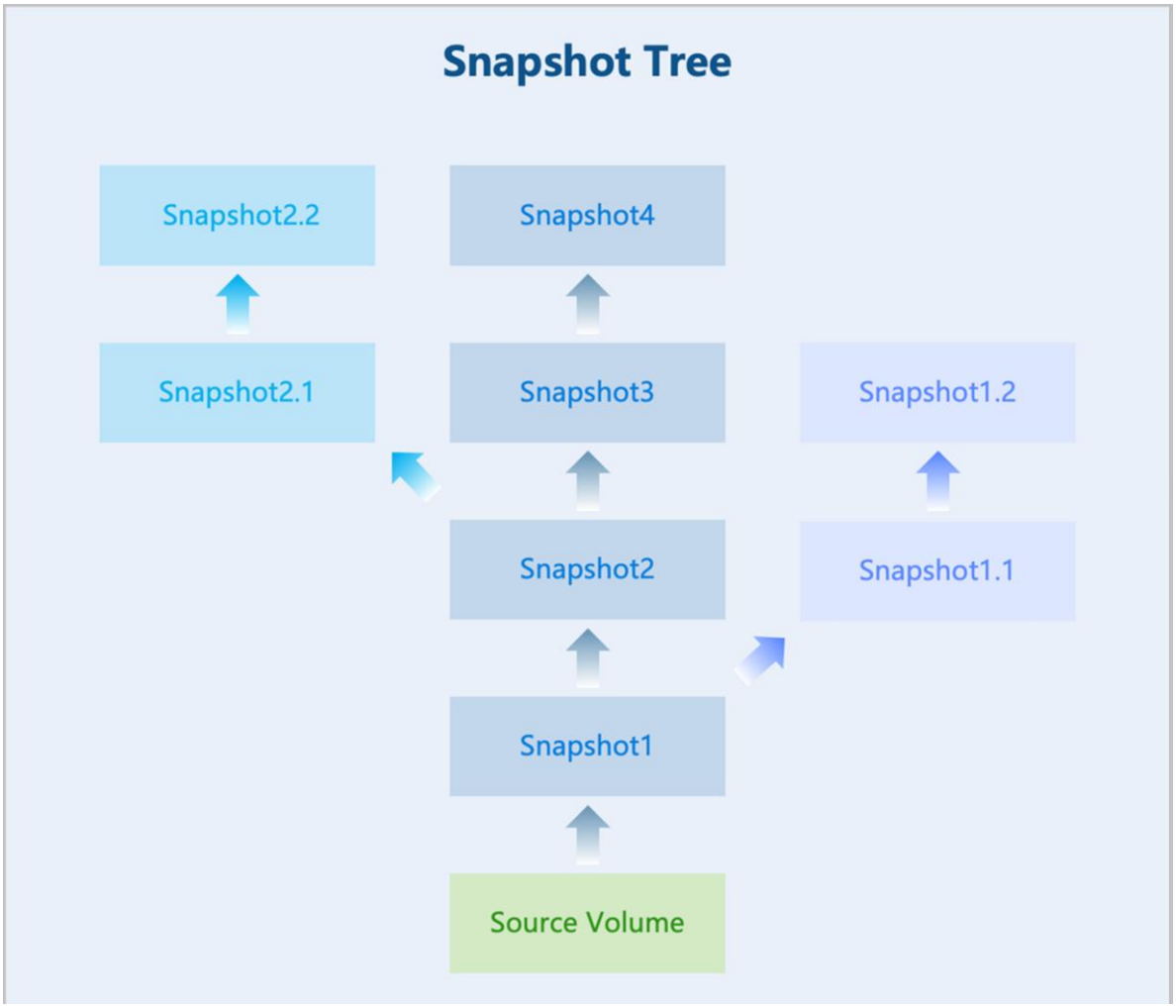
Centralized Storage Snapshot Mechanism

This section introduces QCOW2 external snapshots.

1. Snapshot Chain and Snapshot Tree

Typically, a single disk corresponds to one snapshot chain. NexaVM NSSV supports creating a snapshot tree for a disk, where each branch of the tree represents a distinct snapshot chain.

Figure 4-37: Snapshot Tree



A snapshot tree includes the following information:

- Snapshot Chain: A relational chain composed of a set of snapshots for a disk. Each branch of the snapshot tree is a snapshot chain.
- Snapshot Node: An individual node in a snapshot chain, representing a single snapshot of the disk.
- Snapshot Size: The storage space consumed by a snapshot. You can view the total size of all snapshots in the snapshot tree and the size of an individual snapshot node.

Note:

- For non-distributed storage, the system allows a maximum of 128 nodes per snapshot chain by default. For distributed storage, the maximum number of snapshots per disk is 32, including both manual and automatic snapshots.

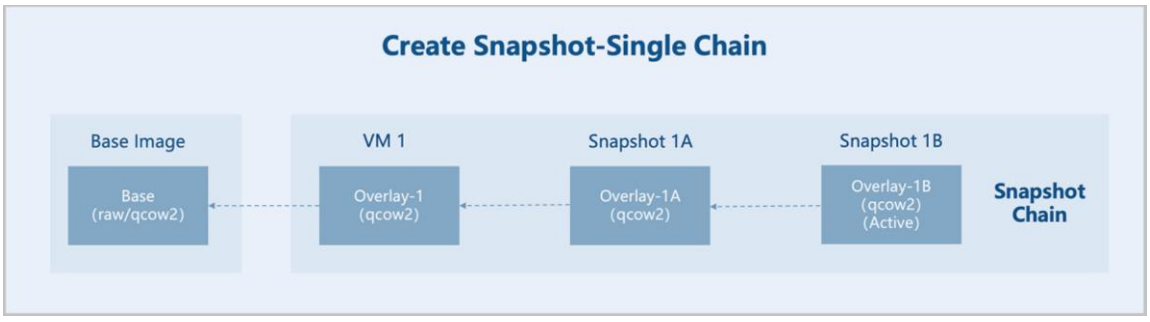
- When a snapshot chain reaches its maximum length:
  - If you continue to create automatic snapshots, the system automatically deletes the earliest automatic snapshot.
  - If you continue to create manual snapshots, you must manually delete unneeded snapshots.
- In production, we recommend that you keep the number of snapshots for an individual virtual machine below 5. Excessive snapshots can affect the VM performance, data security, and data storage capacity.

2. Create Snapshots

When an external snapshot is created, a new empty QCOW2 file is generated. This new file has its backing file pointed to the original QCOW2 file. The original QCOW2 file is set to read-only, effectively turning it into a snapshot itself. Subsequent data writes are directed only to the new QCOW2 file. The creation of an external snapshot involves creating a new blank qcow2 file.

- Create a single snapshot chain based on a backing file.

Figure 4-38: Create Snapshot-Single Chain



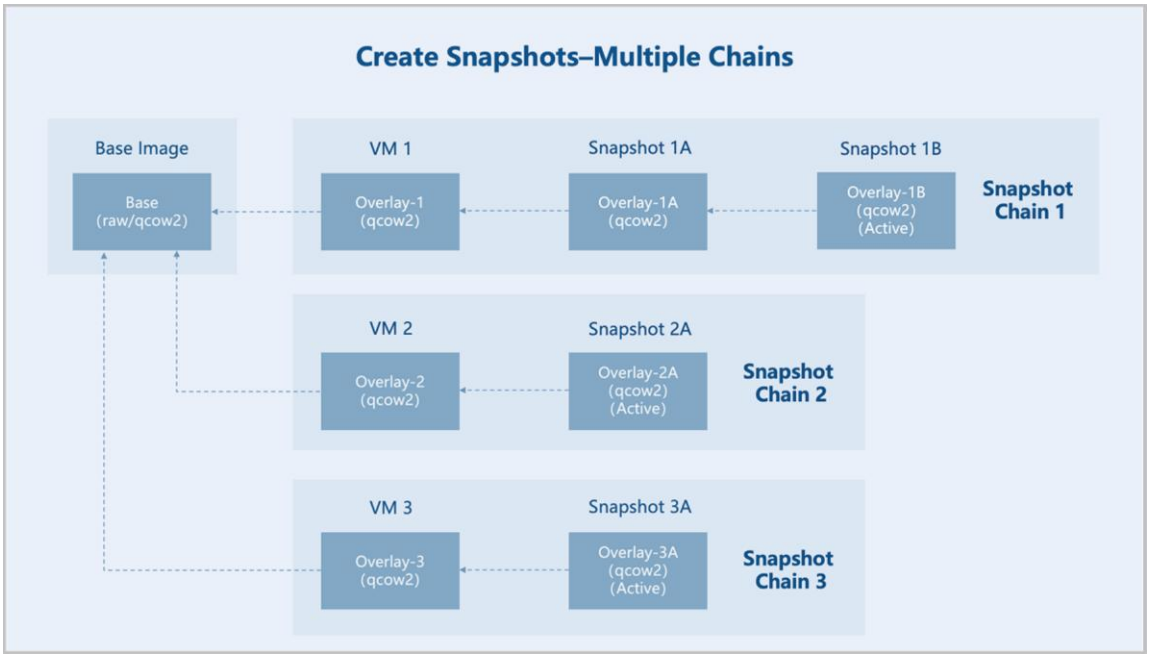
Assume there is an original base image (Base). A virtual machine 1 is created using this base image as a template. Then, snapshot 1A and 1B are created sequentially for virtual machine 1.

- Base Image: A pre-made disk image file containing a complete operating system and bootloader, serving as the Base (read-only).

- VM 1: A new empty file, Overlay-1, is created. Its backing file points to Base. Base remains read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1.
- Snapshot 1A: A new empty file, Overlay-1A, is created. Its backing file points to Overlay -1. Overlay-1 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1A.
- Snapshot 1B: A new empty file, Overlay-1B, is created. Its backing file points to Overlay -1A. Overlay-1A is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1B. virtual machine 1 uses the disk file corresponding to the last snapshot 1B in the snapshot chain, and snapshot 1B is Active.

- Create multiple snapshot chains based on backing files.

Figure 4-39: Create Snapshot-Multiple Chains



Assume there is an original base image (Base). virtual machine 1, virtual machine 2, and virtual machine 3 are created using this base image as a template. Then, snapshot 1A and snapshot 1B are created sequentially for VM 1, snapshot 2A is created for VM 2, and snapshot 3A is created for VM 3.

- Base Image: A pre-made disk image file containing a complete operating system and bootloader, serving as the Base (read-only).

- Snapshot Chain 1:
  - Virtual Machine 1: A new empty file, Overlay-1, is created. Its backing file points to Base. Base remains read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1.
  - Snapshot 1A: A new empty file, Overlay-1A, is created. Its backing file points to Overlay-1. Overlay-1 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1A.
  - Snapshot 1B: A new empty file, Overlay-1B, is created. Its backing file points to Overlay-1A. Overlay-1A is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1B. VM 1 uses the disk file corresponding to the last snapshot 1B in snapshot chain 1, and snapshot 1B is Active.
- Snapshot Chain 2:
  - Virtual Machine 2: A new empty file, Overlay-2, is created. Its backing file points to Base. Base remains read-only. Subsequent data is written only to Overlay-2.
  - Snapshot 2A: A new empty file, Overlay-2A, is created. Its backing file points to Overlay-2. Overlay-2 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-2A. VM 2 uses the disk file corresponding to the last snapshot 2A in snapshot chain 2, and snapshot 2A is Active.
- Snapshot Chain 3:
  - Virtual Machine 3: A new empty file, Overlay-3, is created. Its backing file points to Base. Base remains read-only. Subsequent data is written only to Overlay-3.
  - Snapshot 3A: A new empty file, Overlay-3A, is created. Its backing file points to Overlay-3. Overlay-3 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-3A. VM 3 uses the disk file corresponding to the last snapshot 3A in snapshot chain 3, and snapshot 3A is Active.

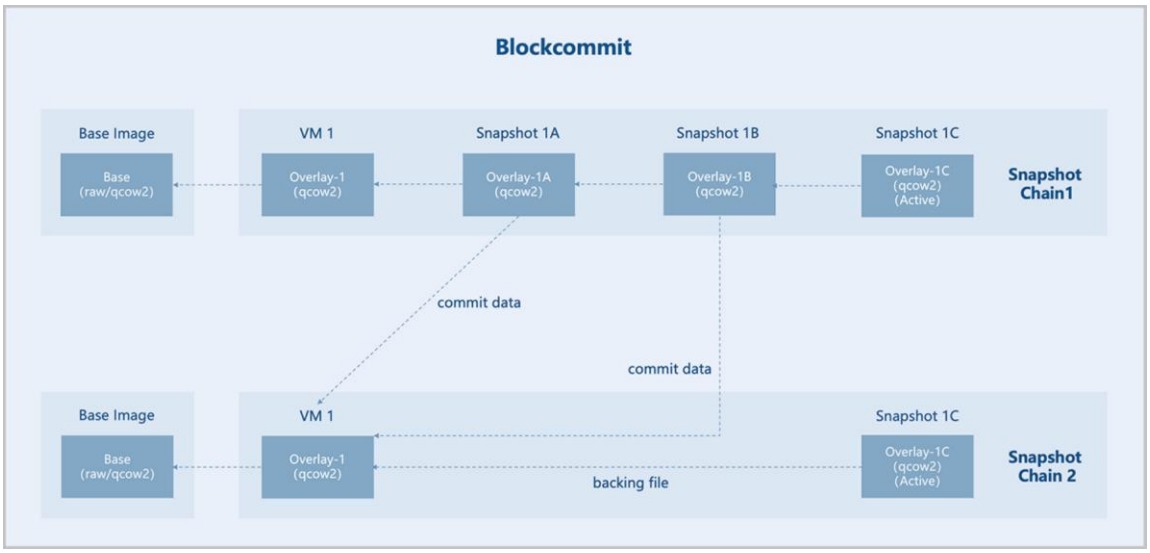
3. Merge Snapshots

External snapshots are interdependent, where each overlay file depends on its backing file. Each snapshot preserves its corresponding data, preventing the direct deletion of an individual snapshot to shorten the chain length. The chain length of external snapshots can be reduced through two methods: downward merging (Blockcommit) or upward merging (Blockpull).

- Blockcommit

  Within the same snapshot chain, you can merge overlays to backing files.

Figure 4-40: Blockcommit

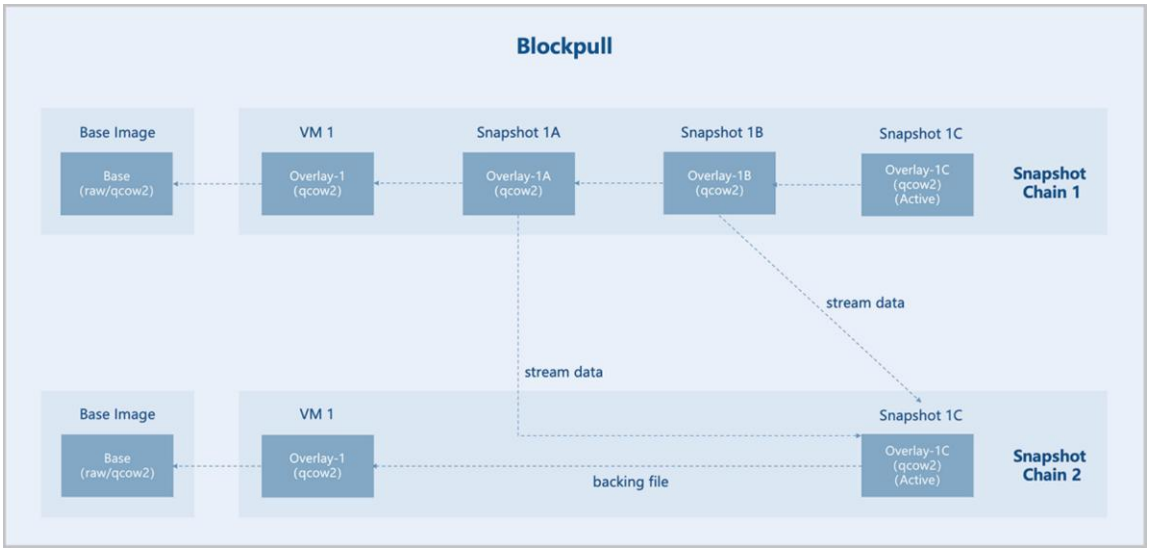


Assume there is an original base image (Base). Virtual machine 1 is created based on Base , and three interdependent external snapshots are created for virtual machine 1: Snapshot 1A, Snapshot 1B, and Snapshot 1C. Now, Snapshot 1A and Snapshot 1B are merged downward into virtual machine 1. As a result, the backing file of Snapshot 1C (Active) points directly to virtual machine 1, so the snapshot chain is shortened. Snapshots 1A and 1B are no longer useful and can be deleted.

- Blockpull

  Within the same snapshot chain, you can merge backing files to overlays.

Figure 4-41: Blockpull



Assume there is an original base image (Base). Virtual machine 1 is created based on Base , and three interdependent external snapshots are created for virtual machine 1: Snapshot 1A, Snapshot 1B, and Snapshot 1C. Now, Snapshot 1A and Snapshot 1B are merged upward into Snapshot 1C (Active). As a result, the backing file of Snapshot 1C (Active) points directly to virtual machine 1, so the snapshot chain is shortened. Snapshots 1A and 1B are no longer useful and can be deleted.

Distributed Storage Snapshot Mechanism

Ceph enterprise edition uses ROW snapshot technology. Self-Developed Distributed Storage uses COW snapshots. For more information, see *Volume Snapshot Protection*.

4.2.2 Backup Management

4.2.2.1 Data Backup

The backup service supports data backup based on the QEMU block device layer. virtual machines on all types of data storage support backup operations. Backup types include full backup and incremental backup. A full backup contains a complete data set, while an incremental backup contains only the data updated since the last backup. Both full and incremental backups only backup actual used data.

By default, the backup policy is configured to automatically perform a full backup after every 63 incremental backups, starting after the initial full backup. Due to the dependencies between incremental backups, a new full backup must be created before previous incremental backups can be deleted. In practice, the system employs more intelligent and flexible strategies to determine the appropriate backup method, ensuring the safety and reliability of the backup data.

The data backup process consists of three parts: data replication, data transfer, and data storage.

4.2.2.1.1 Data Replication

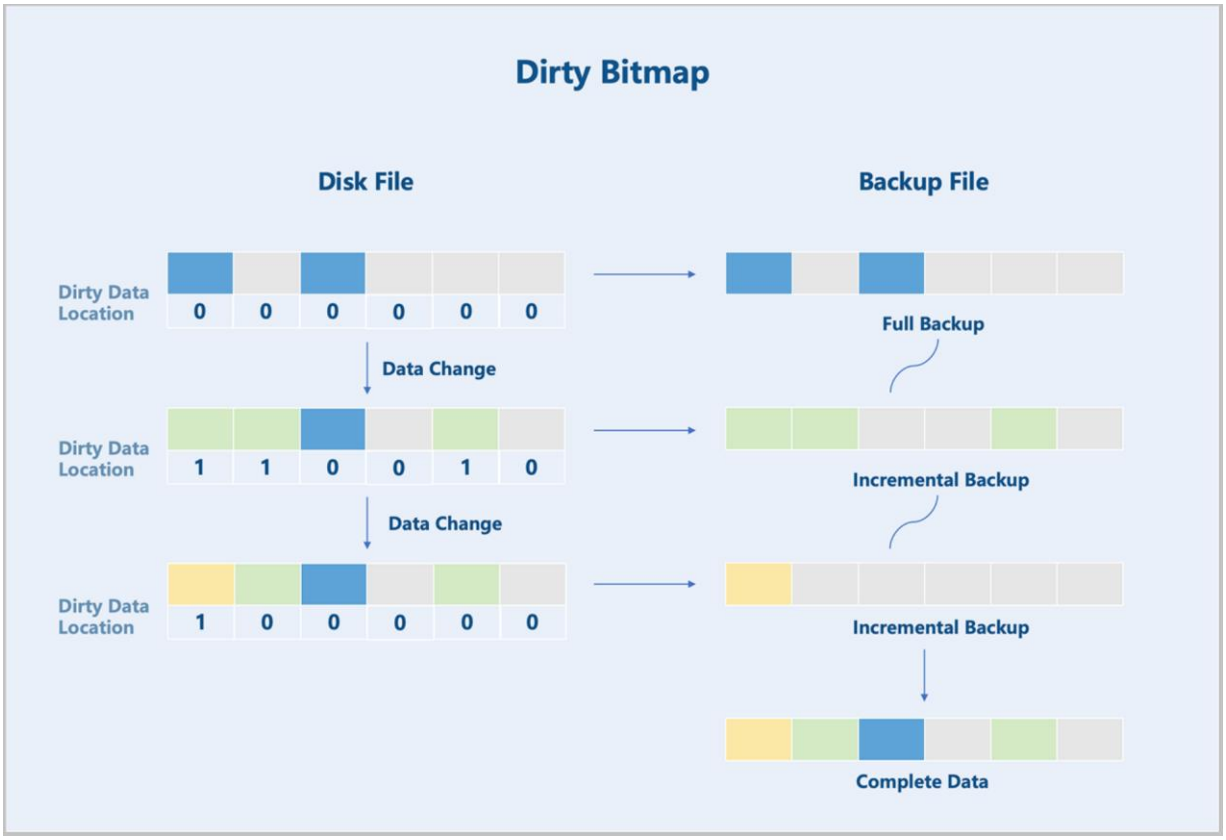
The Backup Service module utilizes the dirty data tracking functionality (Dirty Bitmap) at the QEMU block device layer to track and export backup data.

Locations in the virtual machine's disk where data changes occur are called dirty data locations. The Dirty Bitmap records all such locations in the virtual disk file that have been modified since the last backup. Based on these records, all data modified since the last backup (incremental backup data) can be exported.

Ultimately, the full backup file and subsequent incremental backup files form a complete backup chain to preserve the entire data set. Since the Dirty Bitmap resides in the memory of the

QEMU process, the tracking information is lost after a VM reboot. Therefore, NexaVM NSSV automatically exports a full backup data after a VM reboot.

Figure 4-42: Dirty Bitmap

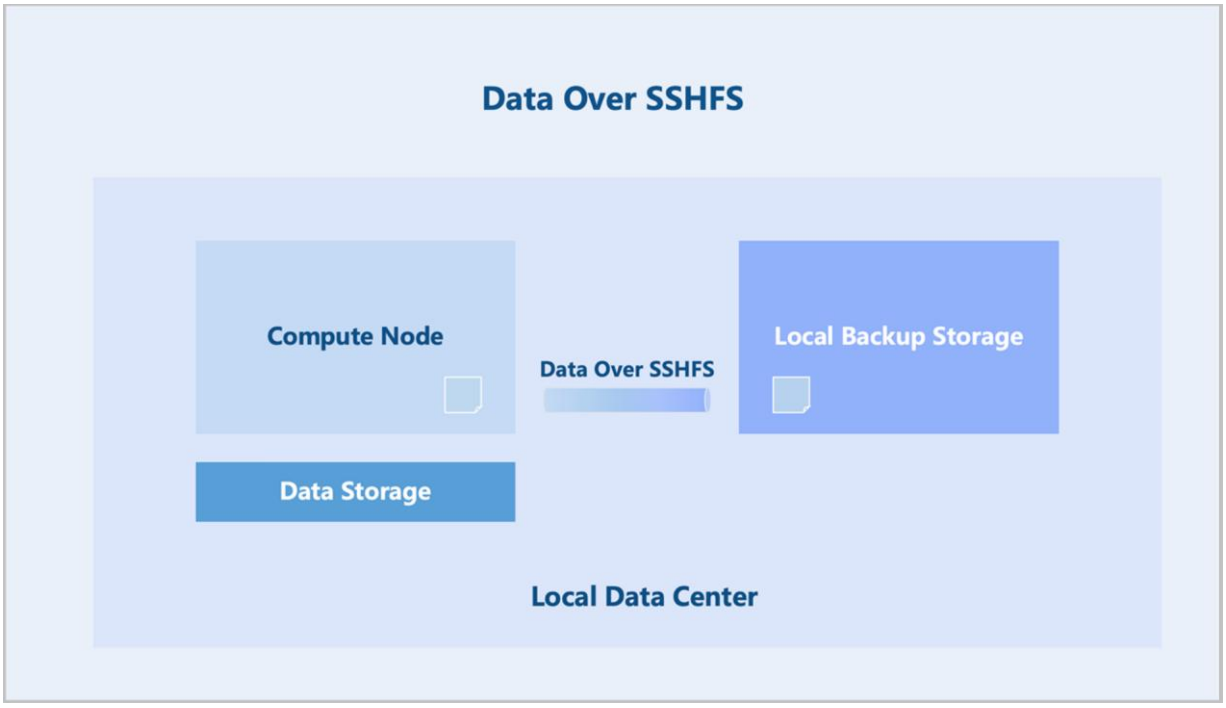


4.2.2.1.2 Data Transfer

Two different implementation schemes are supported for different virtualization component versions, differing primarily in how backup data is transferred.

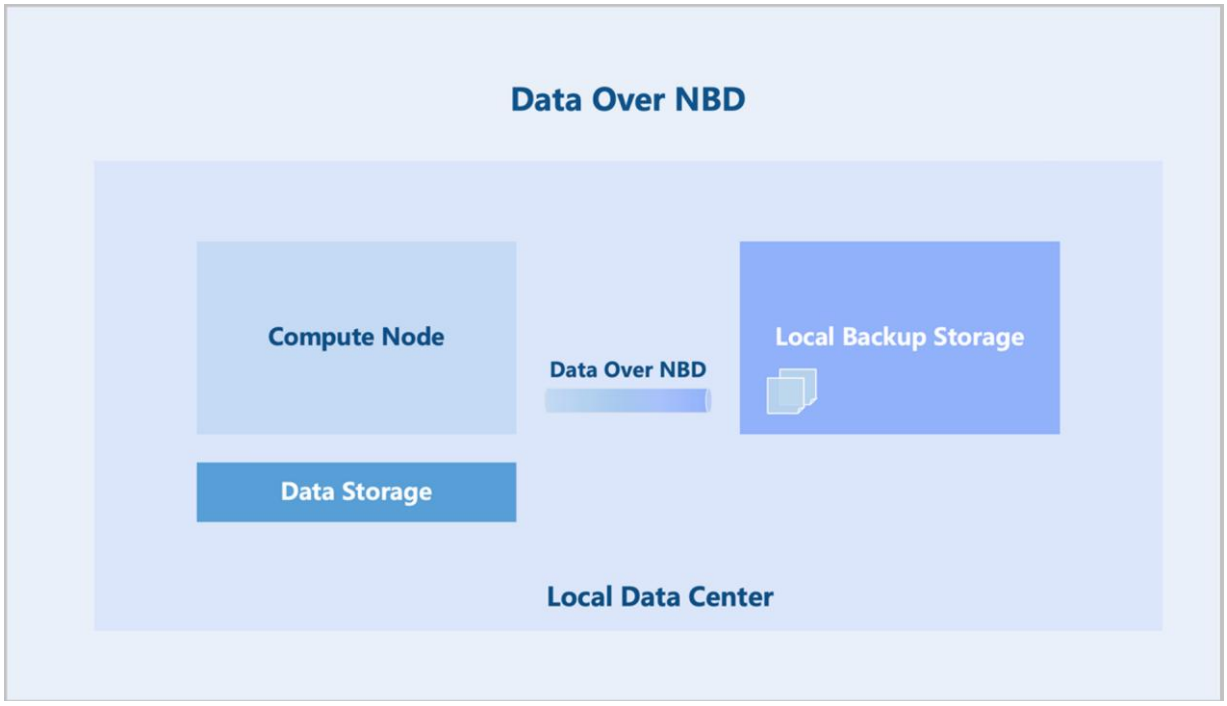
Scheme 1: Data Over SSHFS. This scheme uses SSHFS to mount the backup directory of the remote backup storage on the compute node and imports the backup data into the backup storage. SSHFS is a simple FUSE Over SSH solution. The data channel is encrypted by the SSH session, and each backup job uses a dedicated SSHFS link.

Figure 4-43: Data Over SSHFS



Scheme 2: Data Over NBD. This scheme uses the NBD module on the backup storage to export a backup disk. Then, the compute node utilizes a QEMU block device job (Block-job) to write backup data directly to this disk.

Figure 4-44: Data Over NBD

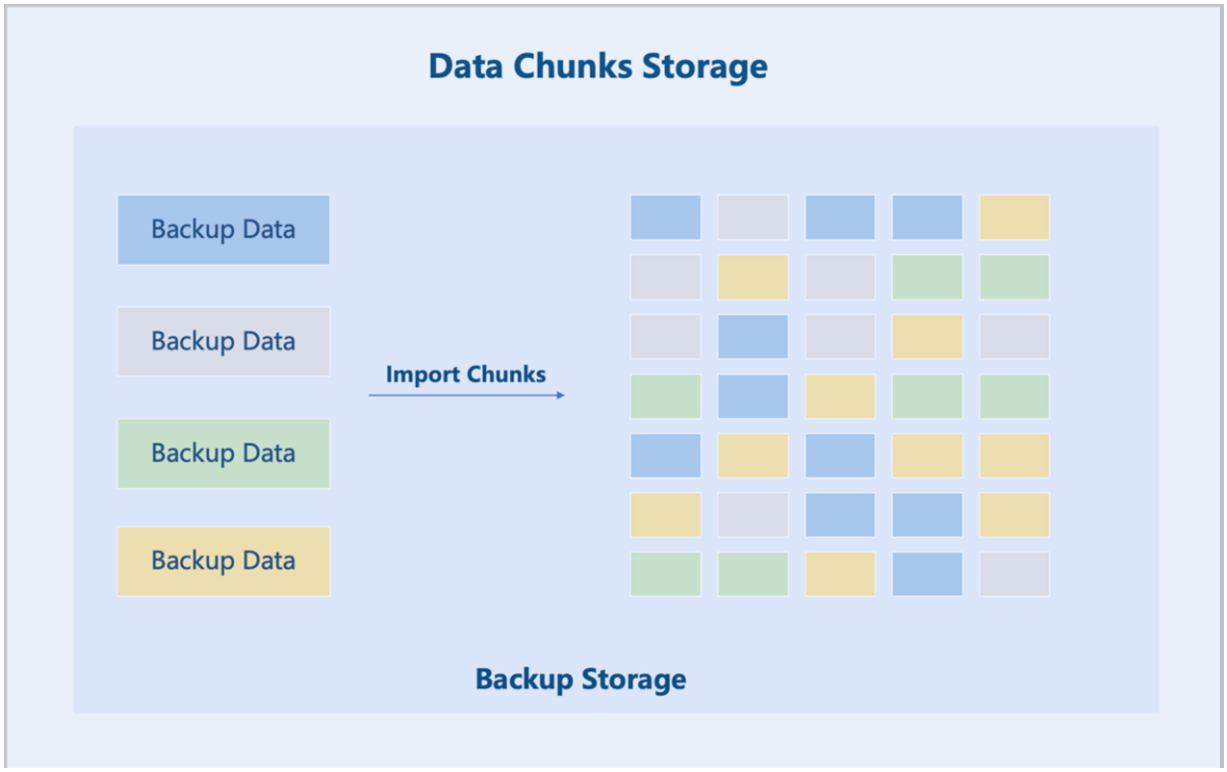


4.2.2.1.3 Data Storage

The backup storage supports various storage media types, including SAN, NAS, disk arrays, and tape libraries.

The backup storage stores data in a chunked and deduplicated format. The backup data is split into 64MB chunks. A hash is calculated for each chunk, and an index is created. Data chunks with identical has values are stored only once.

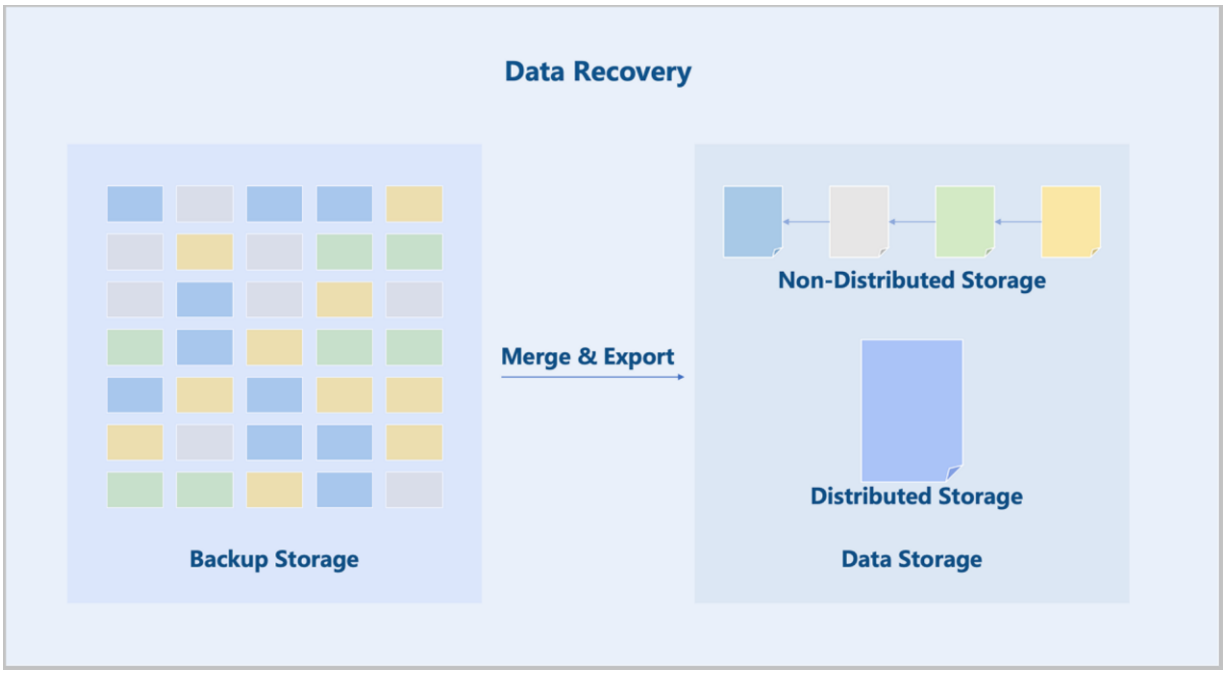
Figure 4-45: Data Chunks Storage



4.2.2.2 Data Recovery

When restoring a virtual machine or volume from a local backup, the chunked data on the backup storage is merged and imported into the data storage. For non-distributed storage, the restored backup data is stored as a disk chain. For distributed data storage, the disk chain is merged into a single disk file for storage. For a new recovery, the disk data restored to the data storage serves as an image cache to create a new virtual machine or volume. For an overwrite recovery, the system updates the database record of the current virtual machine or volume with the new disk path on the data storage and then deletes the old virtual machine or volume files.

Figure 4-46: Data Recovery



4.3 O&M Management

4.3.1 Management Node Monitoring

In a dual-management node HA scenario, you can intuitively view the health status of each management node.

The management node monitoring supports displaying the management node IP, node status, VIP, and management service status for multiple management nodes. The main management services include:

- Arbiter Gateway Reachable: Monitors whether the arbitration IP of the active-standby management node is reachable. If unreachable, it may cause the high availability feature of the management node to fail.
- Peer MN Reachable: Monitors whether the standby management node is reachable. If the standby management node is unreachable, communication with the standby node will not be possible.
- VIP Reachable: Monitors whether the VIP is reachable. If the VIP is unreachable, the primary management node cannot access the UI interface via the VIP.

- Database Status: Monitors the status of the database. If the database is abnormal, there may be a risk of data loss. Please restore the fault promptly.

4.3.2 Monitoring and Alarm

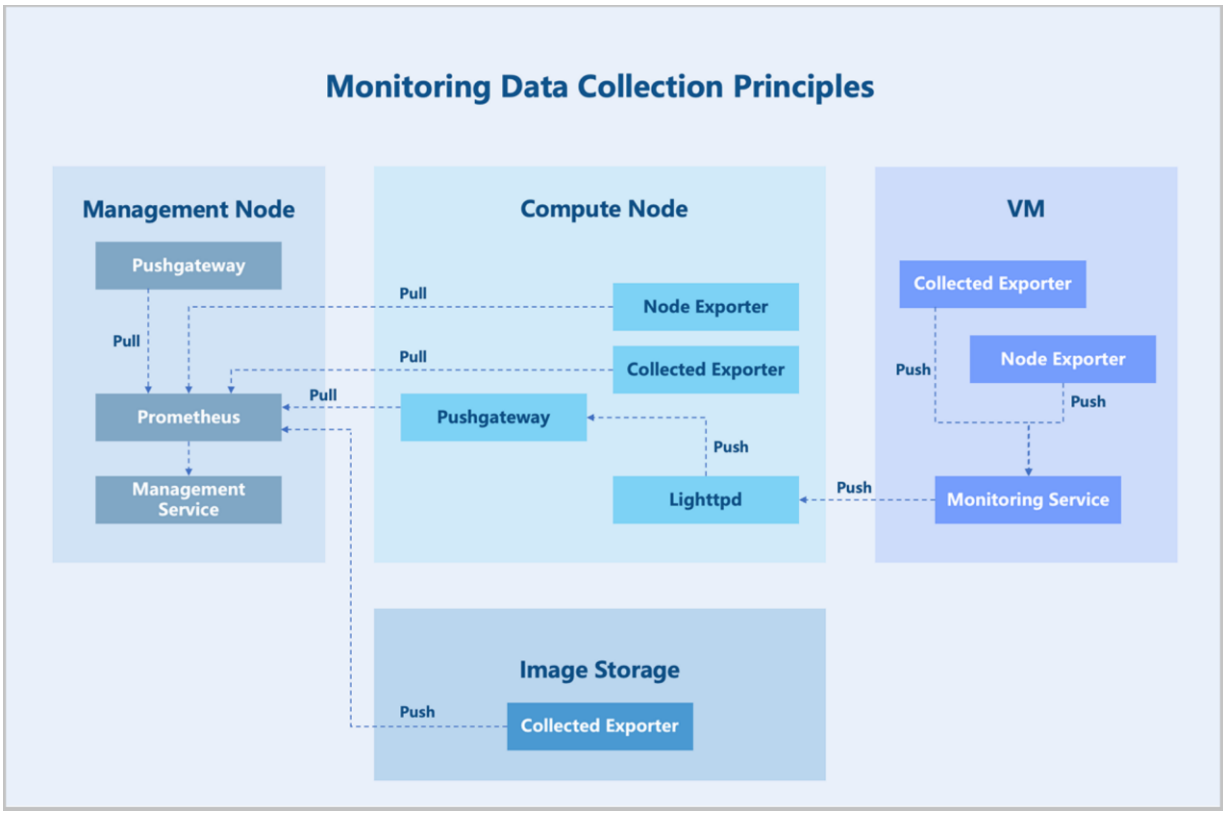
Monitoring and alarm supports monitoring time-series data (such as resource load and capacity data) and predefined system events. It pushes alarms to specified endpoints through the SNS.

The supported alarm types include resource alarm, event alarm, and extended alarm. Supported endpoint types include system, email, DingTalk, WeCom, Lark, Webhook, SMS, Microsoft Teams, and SNMP Trap. Some resource alarms require an installed agent to function.

Prometheus provides time-series monitoring data. When monitoring business data, Prometheus uniformly collects different data.

In the Prometheus architecture, the Prometheus server does not directly monitor specific targets. Its core functions are data collection, storage, and providing external data query support. Therefore, to acquire sample data, such as the host CPU usage, an Exporter is used to periodically gather monitoring samples. NexaVM NSSV employs both pull and push modes to collect monitoring data from different monitoring targets. When hosts or external VM metrics are the monitoring targets, the Prometheus service periodically uses the pull mode to collect data gathered by the Exporter on the hosts. Additionally, due to network or security constraints, Prometheus might not be able to directly access the interiors of VMs. In such cases, a Pushgateway acts as an intermediary to facilitate data transfer. The data collection agent still uses an Exporter to gather monitoring metrics but employs a push mode to periodically send this data to the Pushgateway. Prometheus then uses its pull mode to collect the data from the Pushgateway, thereby achieving unified data collection.

Figure 4-47: Monitoring Data Collection Principles



5 Security

5.1 Compute Security

5.1.1 HTTPS-encrypted UI Login

The system uses HTTPS by default for accessing the UI management interface, providing enhanced security.

- **HTTPS Protocol**: NexaVM NSSV automatically redirects to the HTTPS address on port 443 by default. You can access the UI management interface by entering the management node IP address in your browser, for example, *https://management_node_ip*.
- **HTTP Protocol**: HTTP protocol is disabled by default, but you can configure it manually if needed.
- The system supports **PKCS12** format certificates by default. Currently, only PKCS12 and JKS format certificates are supported. If you are using certificates in other formats, convert them to a supported format.

5.1.2 VM Console

The VM console provides users with a streamlined entry point for monitoring and managing virtual machines. You must have appropriate permissions to access the VM console. Two authentication methods are supported for console login: SSH key authentication and username/password.

- SSH Key Authentication
  - You can use SSH key authentication to log in to Linux virtual machines.
  - An SSH key is a pair of cryptographic keys generated by an algorithm: a **public key**, which is shared openly, and a **private key**, which is kept secure by the user.
  - After a public key is attached to a virtual machine, you can use the corresponding private key to SSH into the virtual machine from another virtual machine without requiring a password.
  - To attach a public key during the virtual machine creation, ensure that the VM image has cloud-init pre-installed. The recommended cloud-init versions are 0.7.9, 17.1, 19.4, or later.

- To attach a public key after the virtual machine creation, ensure that the virtual machine is running and has QEMU Guest Agent (QGA) installed and running. You can install QGA by installing the VMTools. If you install QGA by using other methods, install version 2.5 or later.

- Username/Password
  - You can log into virtual machines using a username and password.
  - The fixed username for Linux virtual machines is **root**, and the fixed username for Windows virtual machines is **administrator**.
  - After a password is injected into a virtual machine, you can use the username or password to SSH into the virtual machine from another virtual machine.
  - Ensure that the VM image has cloud-init pre-installed. The recommended cloud-init versions are 0.7.9, 17.1, 19.4, or later.

5.1.3 High Availability

VM HA

Virtual machines support the high availability (HA) mode. This policy can trigger automatic VM restart when a VM is stopped due to routine maintenance (planned) or unexpected failures (unplanned), thereby improving VM availability.

NeverStop VM HA Mechanism:

- The system uses polling and trigger-based mechanisms to monitor the virtual machine status. If the virtual machine is confirmed to be stopped, a VM configured with HA will be automatically restarted.
- The system uses polling and trigger-based mechanisms to monitor the virtual machine status. If the VM status cannot be definitively determined, the following detection process is initiated:

  1. Based on the existing network configuration, select the most accurate method to probe the status of the host where the virtual machine resides.
  2. If the status of the host is abnormal, the HA-enabled virtual machine will attempt to restart automatically.

5.1.4 IP/MAC/ARP Spoofing Protection

In traditional networks, IP/MAC/ARP spoofing has always been a severe challenge. Through IP/MAC/ARP spoofing, attackers can disrupt the network environment and intercept network secrets.

The system isolates abnormal protocol access initiated by virtual machines at the host's data link layer and blocks virtual machine MAC/ARP spoofing. It also prevents virtual machine IP spoofing at the host's network layer.

5.1.5 Images and Snapshots

Images

You can create images from virtual machines. An image contains a complete data information of a virtual machine. You can use images to quickly replicate corresponding resources.

NexaVM NSSV provides protection for image integrity and security:

- Images use encryption algorithms to protect integrity. When an image is downloaded from the image storage to a data storage, it must pass encryption algorithm verification. The download only proceeds if the verification is successful.
- Image files are stored in slices in the image storage. The sliced image files must be reassembled by NexaVM NSSV before you can read their specific content, thereby ensuring the image data security.

Snapshots

You can create snapshots for virtual machines. A snapshot is essentially a data state file of a disk at a specific time. Before performing important operations, creating a snapshot for a virtual machine can retain the data state (including the memory state) at that specific time, facilitating quick rollback in case of failures. For long-term backup, it is recommended that you use the backup service.

The snapshot feature is applied in the following scenarios:

- Quick Failure Recovery: If an unexpected failure occurs in the production environment, you can use the snapshot rollback feature to quickly restore the environment to the normal state. This method is a temporary solution. For comprehensive long-term data protection, it is recommended that you use the backup service.
- Data Development: By creating snapshots of production data, you can acquire near real-time authentic production data for applications such as data mining, report query, and development testing.
- Improve Operation Fault Tolerance: Before major operations such as system upgrades or business data migration, we recommend that you create one or more snapshots. If any problem occurs during the upgrade or migration process, you can use snapshots to restore the normal system data state in time.

5.1.6 Encrypted Password Storage

NexaVM NSSV supports encrypted storage of all plaintext passwords to protect the privacy and autonomy of user data.

Supported scenarios for encrypted password storage include, but are not limited to:

- Host passwords: Not displayed in plaintext.
- Data storage passwords: Not displayed in plaintext.
- Database passwords: Encrypted and stored by using keys and hidden from users directly.
- Log passwords: All platform log passwords are either not displayed in plaintext or are hidden from users.

5.1.7 Resource Deletion Protection

Deletion Policy

NexaVM NSSV supports configuring deletion policies for critical resources to reduce the risk of accidental deletion.

The current deletion policies include Immediate Deletion, Delayed Deletion, and Never Delete.

- Immediate Deletion: Resources are physically deleted directly and removed from the database . Deleted resources cannot be recovered.
- Delayed Deletion: Resources are first marked as deleted in the database but are not physically deleted. Within a certain period, you can recover resources from the recycle bin in the UI or using APIs. During this period, resources still exist physically and occupy physical space (for example, disk space). After a certain period, resources are physically deleted and cannot be recovered.
- Never Delete: Resources are marked as deleted in the database but are never physically deleted. They occupy physical space all the time.

Resources that currently support deletion policy include virtual machines, disks, images, and bare metal instances.

- VM Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.
- Disk Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.

- Image Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.
- Bare Metal Instance Deletion Policy: Immediate Deletion and Never Delete. The default policy is Never Delete.

UI Deletion Reminder

The UI provides a protection mechanism for deleting important resources. The system displays the consequences of deleting the resource and shows the number of directly associated virtual machines and disks. You must confirm the deletion to proceed, reducing the risk of accidental operation.

5.1.8 Monitoring and Alarm

The monitoring and alarm feature is primarily delivered through an monitoring system and the notification system. The monitoring system monitors time-series data and events, and the notificati on system pushes alarms to specified endpoints.

The monitoring system provides monitoring data metrics, including system performance and resource utilization, in forms such as large-screen monitors, dashboards, graphical charts, and banner notifications, allowing you to fully understand platform resource utilization, operational status, and health indicators. You can also customize alarms and endpoints to achieve flexible and fine-grained monitoring, promptly discover and diagnose related issues.

Characteristics of the monitoring system:

- Time-Series Monitoring: The system currently supports monitoring two types of time-series data.
  - Resource Load Data: For example, virtual machine CPU utilization and host memory utilization.
  - Resource Capacity Data: For example, the number of available IP addresses and the total number of running virtual machines.
- Event Collection: Collects predefined events that occur in NexaVM NSSV, such as host disconnection and virtual machine high availability activation.
- Alerting: Generates alarms for time-series data or events and provides global notifications for important resources, such as available physical capacity of data storage.
- Auditing: Records all operations and provides search functionality.
- Customization: Allows you to customize alarms and alert message templates.

Characteristics of the notification system:

- Pushes alarm messages to specified endpoints.
- The system provides a system endpoint by default. You can set other types of endpoints.

5.2 Network Security

5.2.1 Security Group

Security groups for virtual machines provide control over TCP/UDP/ICMP and other data packets , enabling effective filtering and allowing you to enforce specific security rules for traffic on designated NICs.

5.3 Access Control Security

5.3.1 Two-Factor Authentication

NexaVM NSSV supports two-factor authentication (2FA) as an additional layer of security beyond static passwords. When 2FA is enabled, you must correctly enter a 6-digit dynamic security code from your authenticator app during each login attempt to gain access.

After 2FA is enabled and you successfully log in for the first time, the login QR code is no longer displayed. This helps prevent malicious login attempts and further enhances system security.

5.3.2 AccessKey Authentication

NexaVM NSSV supports AccessKey authentication.

Local AccessKey, consisting of AccessKey ID and AccessKey Secret, is a secure credential issued by NexaVM NSSV. Local AccessKey authorizes third-party users to call the NexaVM NSSV's APIs and access its resources. These credentials must be kept strictly confidential.

5.3.3 Task and Event

NexaVM NSSV provides unified log management, recording user logins and resource operations performed under various accounts. The logs capture details such as operation description, task result, operator, client IP, task creation and completion time, and operation return details. Through operation log auditing, you can meet requirements for security analysis, intrusion detection, resource change tracking, and compliance auditing.

6 Openness and Compatibility

6.1 Open API

NexaVM NSSV provides comprehensive and developer-friendly API support. We offer native RESTful API support, enabling developers to utilize standard HTTP methods and adhere to REST architectural principles for integration. Additionally, we provide both Java SDK and Python SDK options. These SDKs allow developers to programmatically call NexaVM NSSV APIs to implement corresponding functionalities.

The API Reference provides detailed specifications for using the RESTful API and SDKs, along with comprehensive definitions for all available APIs. To further reduce the learning curve for developers, we also offer a visual API Inspector. This tool allows developers to quickly view the invocation details of query and operation APIs directly within the UI, putting all API information at their fingertips. Additionally, developers can copy API call statements with a single click for use in third-party debugging tools, significantly improving efficiency and enhancing the overall developer experience.

6.2 Hardware Compatibility

NexaVM NSSV supports loosely coupled deployment of software and hardware. You can plan hardware configurations and platform software components according to your actual requirements, as NexaVM NSSV imposes no restrictions or specific requirements on hardware brands. If you have existing server equipment intended for reuse, NexaVM NSSV supports legacy hardware compatibility and enables unified management through the platform.