Kunpeng BoostKit for SDS

Technical White Paper

Issue 07

Date 2025-09-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Overview

With the explosive growth of applications such as IoT, big data, and mobile Internet, more and more data is generated, and the total storage market volume increases year by year. As distributed storage becomes increasingly popular, some applications that require high performance, such as databases of financial systems, also use distributed storage. To achieve high performance, storage systems usually need to use SSDs as the main storage medium. Therefore, more powerful CPUs are required to fully utilize the performance of the SSDs. Kunpeng servers are powered by Huawei-developed Kunpeng series processors. The Kunpeng 920 processor provides up to 64 cores@2.6 GHz, which outperforms mainstream platforms in the industry. Boosted by Kunpeng processors, the Kunpeng software-defined storage (SDS) system has distinguished performance advantages over distributed storage systems based on traditional hardware platforms.

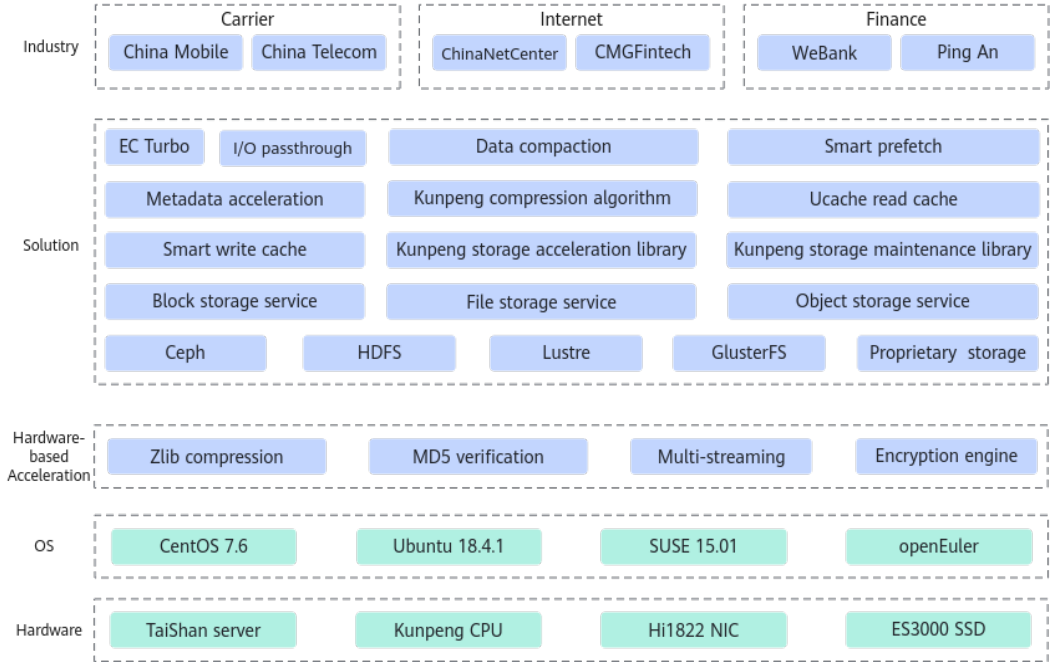
Kunpeng BoostKit for SDS is based on the Kunpeng hardware platform. With Huawei-developed leading processors, NICs, SSDs, management chips, and AI chips and open-source Ceph distributed storage software, it provides customers with a distributed storage solution covering block storage, file storage, and object storage services.

2 Architecture

The Kunpeng BoostKit for SDS consists of the hardware platform, OS, middleware, and distributed storage software. The distributed storage software only supports the open source Ceph.

Figure 2-1 shows the overall architecture of the Kunpeng BoostKit for SDS.

Figure 2-1 Kunpeng BoostKit for SDS architecture



3 SDS Cluster Architecture

3.1 Ceph Storage Cluster

3.1.1 Introduction

Ceph provides a scalable, reliable storage service for storage clusters based on Reliable, Autonomic Distributed Object Store (RADOS).

A Ceph storage cluster contains two types of daemons:

- Ceph monitor

  The Ceph monitor maintains the primary copy of the cluster map. The monitor cluster ensures high availability when a monitor fails. The storage cluster client requests the latest copy of the cluster map from the Ceph monitor.

- Ceph object storage daemon (OSD)

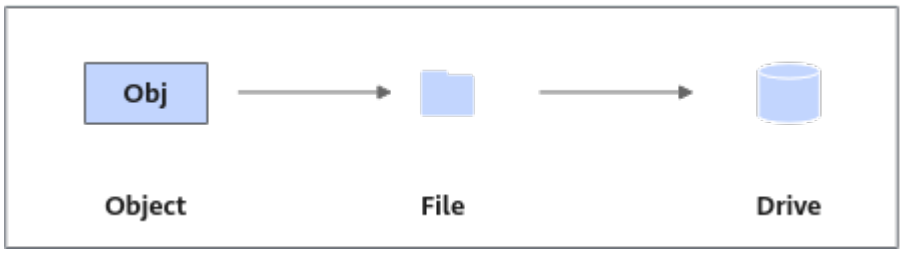
  The Ceph OSD checks its own status and the status of other OSDs and reports the status to the monitor.

3.1.2 Data Storage

The storage cluster clients and each Ceph OSD use the Controlled Replication Under Scalable Hashing (CRUSH) algorithm to efficiently calculate the data location without relying on a centralized query table. Its advanced functions include librados-based storage interfaces and multiple librados-based service interfaces.

The Ceph storage cluster receives data from the Ceph client and stores the data as objects, regardless of whether the data comes from the Ceph block device, Ceph object storage, Ceph file system, or librados-based customized implementation. Each object is a file in the file system and is stored on object storage devices. The Ceph OSDs process read and write operations on storage devices.

Figure 3-1 Data storage example



The Ceph OSDs store all data as objects in a flat namespace without a directory hierarchy. An object contains an identifier, binary data, and metadata composed of name/value pairs. The metadata semantics depend entirely on Ceph clients.

3.1.3 Ultra-large Clusters

In many cluster architectures, the main purpose of cluster members is to let the centralized interface know which nodes it can access, and then the central interface provides services for the client through two-level scheduling. In a petabyte- to exabyte-scale system, the scheduling system is the biggest bottleneck.

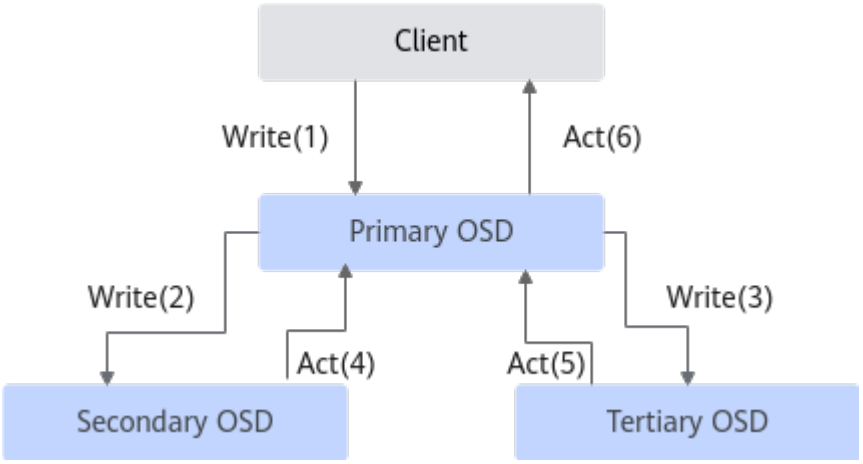
Ceph eliminates this bottleneck. Both the OSDs and clients can sense the cluster. For example, Ceph clients and OSDs know other OSDs in the cluster, and each OSD can directly communicate with other OSDs and monitors. In addition, Ceph clients can directly interact with OSDs.

Ceph clients, monitors, and OSDs can interact directly with each other, which means that OSDs can use the CPU and memory of the local node to perform tasks that may overwhelm the central server. This design balances computing resources and brings the following benefits:

- **OSDs directly serve clients**. All network devices have a limit on concurrent connections. When the number of concurrent connections is large, the physical limitations of centralized system are evident. Ceph allows clients to directly communicate with OSD nodes, eliminating single points of failure (SPOF) and improving performance and system capacity. Ceph clients can maintain sessions with a certain OSD on demand, rather than a central server.
- **OSD members and status**: After being added to the cluster, the Ceph OSD continuously reports its status. At the low level, the OSD status is up or down, indicating whether it is running and whether it can provide services. If the status of an OSD is **down** and **in**, the OSD may be faulty. If an OSD is not running (for example, the OSD has crashed), it cannot report to the monitor that it is down. The Ceph monitor periodically pings OSDs to ensure that they are running. In the meantime, it also authorizes each OSD to check whether neighboring OSDs are down, update the cluster map, and report to the monitor. This mechanism means that the monitor is still a lightweight process.
- **Data scrubbing**: As part of maintaining data consistency and cleanliness, OSDs can scrub objects in a placement group (PG). Ceph OSDs can compare object metadata with metadata of copies stored on other OSDs to catch OSD defects or file system errors daily. Ceph OSDs can also perform deep scrubbing weekly by comparing object data by bit to identify bad sectors that are not detected during light scrubbing.

- **Replication**: Like Ceph clients, OSDs also use the CRUSH algorithm. However, the OSDs use it to calculate where copies are stored (also used for rebalancing). In a typical write scenario, a client uses the CRUSH algorithm to calculate where an object should be stored, maps the object to a storage pool and a PG, and then searches the CRUSH map to determine the primary OSD of the PG.
- The client writes the object to the primary OSD of the target PG. The primary OSD uses its CRUSH map copy to find the secondary and tertiary OSDs for storing the object copies, and copy the data to the secondary and tertiary OSDs corresponding to the appropriate PG (the number of OSDs is the same as the number of copies). The data storage success information is fed back to the client.

Figure 3-2 Ceph three-copy write



With the capability of making copies, OSDs can reduce the replication pressure of clients and ensure the high reliability and security of data.

3.2 Dynamic Cluster Management

- Autonomous, self-repairing, and intelligent OSD daemons are the key design of Ceph. The following describes how Ceph dynamically implements data mapping, rebalancing, and data consistency.
- The Ceph storage system supports the concept of pool, which is a logical partition of a storage object.
- The Ceph client obtains a cluster map from the monitor and writes the object to the storage pool. The size or number of copies, CRUSH rule set, and number of PGs of a storage pool determine how Ceph stores data.

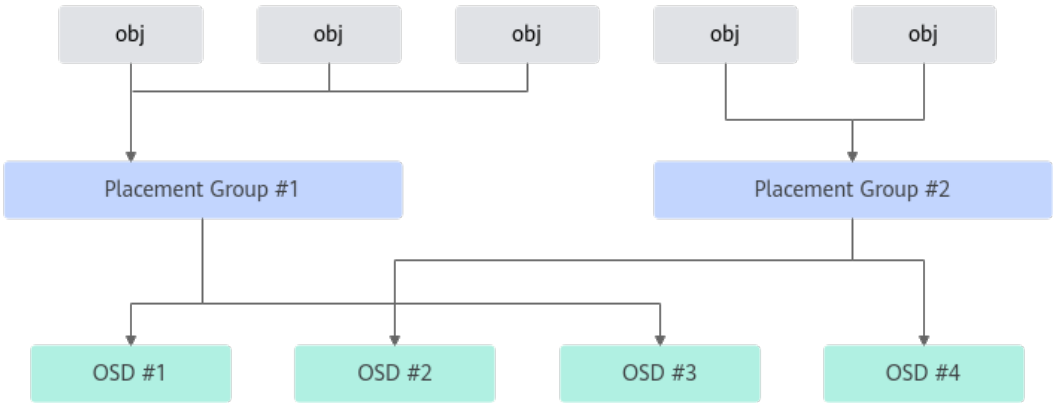
3.2.1 Mapping an Object to an OSD

Each storage pool has many PGs. CRUSH dynamically maps them to OSDs. When the Ceph client saves objects, CRUSH maps the objects to a PG.

An indirect layer between OSDs and clients is created when objects are mapped to PGs. The Ceph cluster must be able to increase and reduce the size and dynamically rebalance. If a client knows what objects are stored in each OSD,

clients and OSDs are tightly coupled. In contrast, the CRUSH algorithm maps objects to PGs, and then maps each PG to one or more OSDs. This indirect layer enables Ceph to dynamically rebalance when OSDs and underlying devices go online. The following figure shows how CRUSH maps objects to PGs and then maps PGs to OSDs.

Figure 3-3 CRUSH mapping



With the cluster map copy and CRUSH algorithm, the client can accurately calculate the target OSD for object read and write.

When a Ceph client is bound to a monitor, the client obtains the latest copy of the cluster map. With this copy, the client can know all monitors, OSDs, and metadata servers in the cluster. However, it does not know the object locations.

The object locations are calculated.

The client only needs to enter the object name and storage pool name, which is simple. Ceph stores data in a storage pool such as liverpool. When the client wants to save a named object (such as john, paul, george, or ringo), it uses the object name, hash value, number of PGs in the storage pool, and the storage pool name to calculate the PG. Ceph calculates the PG ID as follows:

1. The client enters the storage pool name and object name (for example, **pool="liverpool"** and **object-id="john"**).
2. CRUSH obtains the object name and hashes it.
3. CRUSH performs a modulo operation on the hash value by using the number of PGs (for example, 58), and obtains the PG ID.
4. CRUSH obtains the storage pool ID (for example, **liverpool = 4**) based on the storage pool name.
5. CRUSH adds the storage pool ID before the PG ID (for example, 4.58).

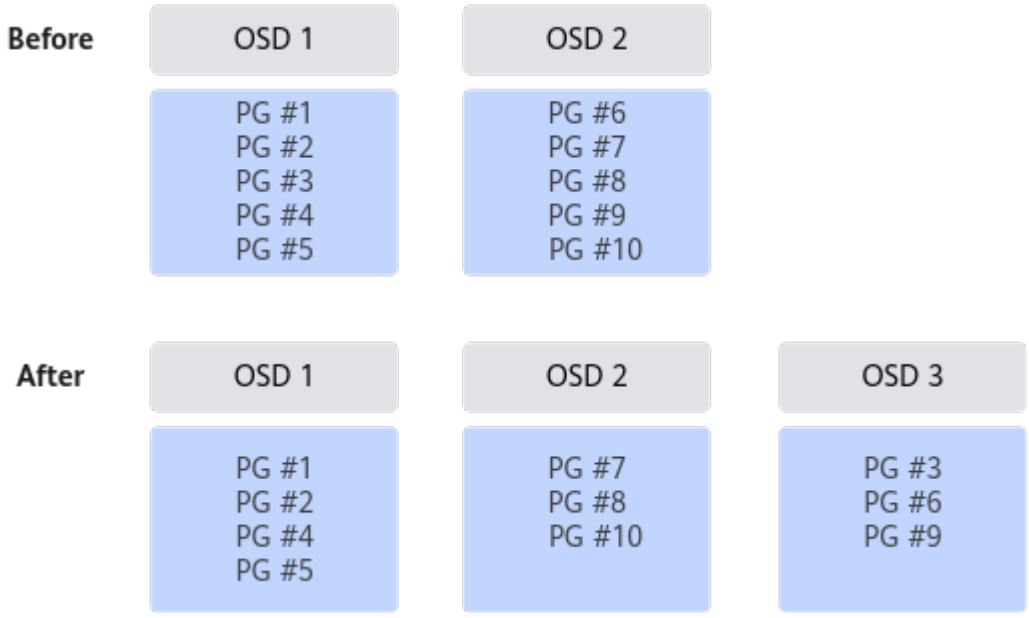
Calculating the object location is much faster than querying the location. The CRUSH algorithm allows the client to calculate where an object should be stored and allows the client to connect to the primary OSD to store or retrieve objects.

3.2.2 Data Rebalancing

When an OSD is added to the Ceph storage cluster, the cluster map needs to be updated by using the new OSD. Calculating PG IDs will alter the cluster map, which consequently changes the object placement because the input conditions

for PG ID calculation have changed. The following figure shows the rebalancing process (for reference only; the actual process is subject to the specific conditions). Some PGs instead of all PGs are migrated from the existing OSDs (OSD 1 and OSD 2) to the new OSD (OSD 3). The CRUSH algorithm is stable even during rebalancing. Most PGs retain their initial configurations, and each OSD has spare space. Therefore, load on the new OSD does not increase sharply after rebalancing is complete.

Figure 3-4 Rebalancing process



3.2.3 Data Consistency

As a part of maintaining data consistency and cleanliness, OSDs can scrub objects in PGs, that is, OSDs compare metadata of different OSD object copies in a PG. Scrubbing is usually performed daily to capture OSD defects and file system errors. OSDs also perform deeper scrubbing by comparing object data by bit to capture bad sectors on drives that are not detected during light scrubbing.

To ensure data consistency in the Ceph cluster, you can select either of the multi-copy solution and erasure coding solution.

4 Advantages

Kunpeng BoostKit for SDS is based on the Kunpeng hardware platform. It streamlines the full stack of the hardware, OS, middleware, and SDS software. This solution provides the following benefits:

High Performance

The Kunpeng BoostKit for SDS uses multi-core high-performance processors and integrates a hardware compression acceleration engine.

- The Kunpeng server (powered by two Kunpeng 920 5250 processors) delivers 20% higher service performance than mainstream high-end 2-socket servers in the industry.
- The Kunpeng server (powered by two Kunpeng 920 3210 processors) delivers 20% higher performance than mainstream 2-socket servers in data compression applications.
- The Kunpeng server (powered by two Kunpeng 920 5220 processors) delivers the same performance as mainstream mid-range 2-socket servers in the industry.
- The Kunpeng server (powered by two Kunpeng 916 5130 processors) delivers the same performance as mainstream low-end servers in the industry.

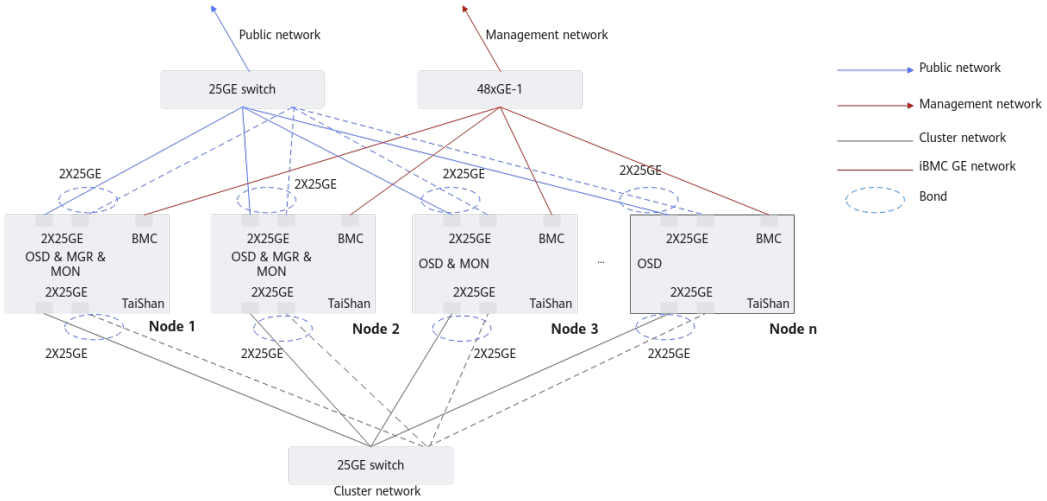
Open Ecosystem

The Kunpeng BoostKit for SDS boasts a mature ecosystem:

- Open-source Ceph distributed storage system
- Ubuntu, CentOS, and openEuler OSs

5 Networking

Figure 5-1 Kunpeng BoostKit for SDS networking



As shown in **Figure 5-1**, two 25GE network ports are bonded into one 50GE port when the bandwidth of a single network port cannot meet the customer's service requirements.

6 Features

6.1 Block Storage Service

Ceph block storage is also called a RADOS block device (RBD). Ceph introduces a new RBD protocol for block devices. RBD provides reliable, distributed, and high-performance block storage for clients. RBD blocks are striped and distributed on multiple Ceph objects. These objects are distributed in the entire Ceph storage cluster, ensuring data reliability and performance. RBD has been supported by the Linux kernel. The RBD driver has been well integrated with the Linux kernel over the past few years. Almost all Linux OS distributions support RBD. In addition to reliability and performance, RBD also supports other enterprise-level features, such as full and incremental snapshot, thin provisioning, copy-on-write clone. RBD also supports full memory caching, which can greatly improve its performance.

A Ceph RBD image can be used as a drive mapped to a physical bare-metal server (BMS), a virtual machine (VM), or another host. Industry-leading open-source hypervisors, such as KVM and Xen, fully support RBD and use RBD to provide block storage services for VMs.

Ceph block devices fully support cloud platforms such as OpenStack. In OpenStack, Ceph block devices can be used through cinder (block) and glance (image) components. In this way, the copy-on-write feature of the Ceph block storage can be used to create thousands of VMs in a short time.

6.2 File Storage Service

Physical storage resources of a distributed file system may not be connected to the local node directly. Instead, the resources are connected over a computer network. CephFS uses the Ceph cluster to provide a file system compatible with POSIX, allowing Linux to directly mount Ceph storage to the local host. Like NFS and SAMBA, it provides shared folders. Clients use the storage provided by Ceph by mounting directories.

When the CephFS is used, you need to configure the MDS node. The MDS node is similar to a proxy cache server of metadata. It provides metadata computing, cache, and synchronization for the Ceph file system.

6.3 Object Storage Service

Unlike traditional storage methods, object storage does not store data in files or blocks. Instead, it stores data as objects or key-value pairs. Multiple servers are configured with large-capacity drives and the Ceph object storage software to provide external read and write interfaces. Each object needs to store data, metadata, and a unique identifier. Object storage provides fast read/write speed like block storage and support data sharing like file storage. Therefore, object storage applies to scenarios with few updates, such as image storage and video storage. However, object storage cannot be directly accessed by the OS like a drive of a file system, and can only be accessed at an application layer by using APIs. The Ceph object gateway (also called RADOS gateway (RGW) interface) based on the Ceph RADOS layer provides interface object storage compatible with OpenStack Swift and Amazon S3, including GET, PUT, DEL, and other extensions.

RADOS is the basis of the Ceph storage cluster. In Ceph, all data is stored as objects, and RADOS object storage is responsible for storing these objects regardless of the data type. The RADOS layer ensures data consistency. The RADOS must have its own user management. The RADOS gateway provides RESTful API for users' applications to store data in the Ceph cluster. The RGW interface is compatible with OpenStack Swift and Amazon S3.

RGW uses librgw and librados to allow applications to connect to the Ceph object storage. It converts API requests into librados requests and provides RESTful API interfaces compatible with OpenStack Swift and Amazon S3.

During the internal logic processing of the RGW, the HTTP front end receives the request data and saves it in the data structure. The RESTful API common processing layer parses the HTTP semantics to obtain the OpenStack Swift or Amazon S3 data and performs a series of checks. After the checks are passed, different processing procedures are performed according to the API operation requests. To obtain data from or write data to the RADOS cluster, the RGW and RADOS API adaptation layer calls the librados API to send a request to the RADOS cluster.

You can bypass the RGW layer to access the Ceph object storage system in a more flexible and faster manner. The librados software library allows users' applications to directly access the Ceph object storage through C, C++, Java, Python, and PHP. The Ceph object storage supports the multi-site capability, that is, it can provide a solution for disaster recovery (DR). You can use RADOS or the unified gateway to configure object storage for multiple sites.

From a storage perspective, Ceph OSDs perform a mapping from objects to blocks (a task that is often performed at the Ceph client file system layer). This action allows the local entity to best determine how to store an object. Earlier versions of Ceph implemented a custom low-level file system on a local storage device named EBOFS. This system implements a non-standard API to the underlying storage that has been tuned for object semantics and other features such as asynchronous notifications submitted by drives. Currently, the B-tree file system (BTRFS) can be used on storage nodes, and has implemented some necessary functions such as embedded integrity.

6.4 Common Features

6.4.1 Bcache

Bcache Overview

Bcache is the cache of the block device layer in the Linux kernel. One or more high-speed drives (such as SSDs) can be used as the cache of low-speed drives. Bcache is incorporated into the kernel mainline from Linux 3.10.

Bcache has the following features:

- A cache device can serve as the cache for multiple devices. Caches can be dynamically added and deleted while the devices are in operation.
- Data can be recovered from an abnormal shutdown. The write completion is confirmed only after cache data is written to a back-end device.

- Write blocking and cache flushing are correctly processed.
- Write cache modes such as writethrough, writeback, and writearound are supported.
- Sequential I/Os are detected and bypassed (with a configurable threshold or the option to disable this function).
- When the SSD latency exceeds the configured threshold, the SSD traffic is reduced (this function is used when an SSD is used as the cache of multiple drives.)
- Prefetch is performed when cache miss occurs (disabled by default).
- High-performance writeback implementation. Dirty data is sorted and then flushed to drives. If the **writeback_percent** value is set, the background writeback process uses the PD controller to smoothly process dirty data based on the dirty data proportion.
- The Bcache random read rate can reach 1 million IOPS when an efficient B+ tree is used and the hardware devices are fast enough.
- Bcache can run stably in production.

Cache Modes of Bcache

Bcache supports three cache modes: writeback, writethrough (default), and writearound. The cache mode can be dynamically modified.

- Writeback: All data is written to the cache drive first. The system writes the data to back-end data drives later. This mode is disabled by default.
- Writethrough: Data is written to the cache drive and back-end data drives at the same time. This mode is enabled by default and applies to read-dominant workloads.
- Writearound: Data is directly written to back-end drives.

Common Tuning Methods

Step 1 Set the writeback mode to improve the write performance.

```
echo writeback > /sys/block/bcache0/bcache/cache_mode
```

Step 2 Allow cache sequential I/Os or set the sequential I/O threshold.

- Enable cache sequential I/Os.
  ```
  echo 0 > /sys/block/bcache0/bcache/sequential_cutoff
  ```
- Adjust the sequential I/O threshold.
  ```
  echo 4M > /sys/block/bcache0/bcache/sequential_cutoff
  ```

After the preceding settings are performed, the system writes I/O data directly to the back-end data drives when the sequential I/O cache size exceeds the threshold.

Step 3 Disable the congestion control function.

```
echo 0 > /sys/fs/bcache/<cache set uuid>/congested_read_threshold_us
echo 0 > /sys/fs/bcache/<cache set uuid>/congested_write_threshold_us
congested_read_threshold_us # Defaults to 2,000 µs.
congested_write_threshold_us # Defaults to 20,000 µs.
```

For more information, see the Bcache home page and manual:

- **Link 1**

- **Link 2**

----End

6.4.2 Journal

Ceph OSDs use the journal to achieve fast speed and consistency.

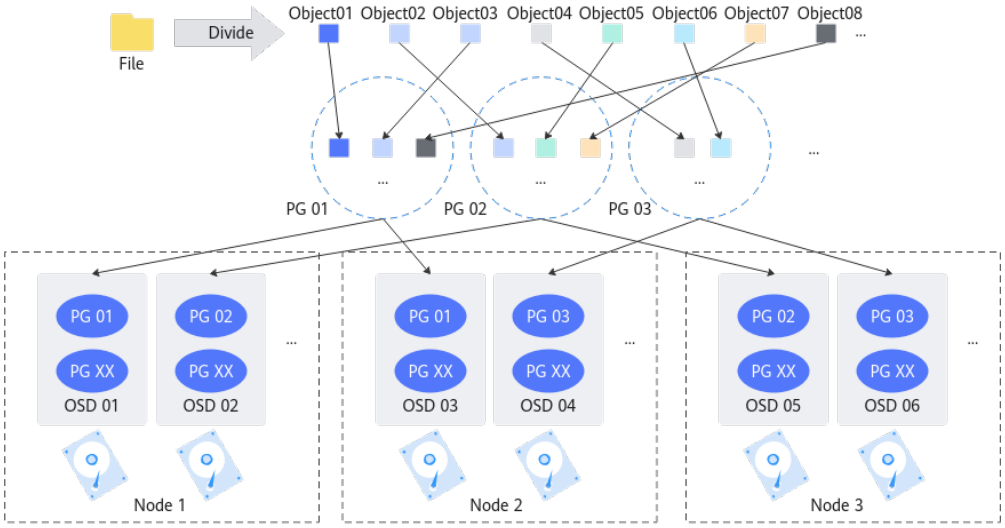
- Speed: The journal enables OSDs to quickly submit small write data blocks. Ceph writes small data blocks and random I/Os to the journal. In this way, the back-end file system can merge write operations and improve the concurrent bearing capacity. Therefore, the OSD journal can provide excellent burst write performance. Actually, data has not been written to an OSD because the file system captures the data in the journal.
- Consistency: Ceph OSDs require a file system interface that can ensure atomic operations. An OSD writes the description of an operation to the journal and then applies the operation to the file system. This requires the atomic update of an object (for example, the metadata of a PG). The OSD stops writing data and synchronizes the journal to the file system at intervals. In this way, the OSD can modify operations in the journal and reuse space. If the synchronization fails, the OSD replays the journal from the previous synchronization point.

6.4.3 Replication

The Ceph distributed storage uses the replication mechanism to ensure data reliability. Three copies are saved by default (the number of copies can be modified) Ceph uses the CRUSH algorithm to implement fast and accurate data storage in a large-scale cluster. In addition, Ceph can minimize data migration when the hardware is faulty or hardware devices are added. The working principles are as follows:

1. When users need to store data in the Ceph cluster, the data is divided into multiple objects. Each object has an object ID, and the object size can be configured. The default object size is 4 MB. An object is the minimum storage unit of the Ceph cluster.
2. There are a large number of objects. To effectively reduce object-to-OSD index tables, lower the complexity of metadata, and make read/write more flexible, PGs are introduced. PGs are used to manage objects. Each object is mapped to a PG through a hash algorithm. A PG can contain multiple objects.
3. The PGs are mapped to OSDs through CRUSH calculation. If there are three copies, each PG is mapped to three OSDs, ensuring data redundancy.

Figure 6-1 Resource allocation in the CRUSH algorithm (using two copies)



The CRUSH algorithm is affected by the following factors:

1. Current system status (cluster map)

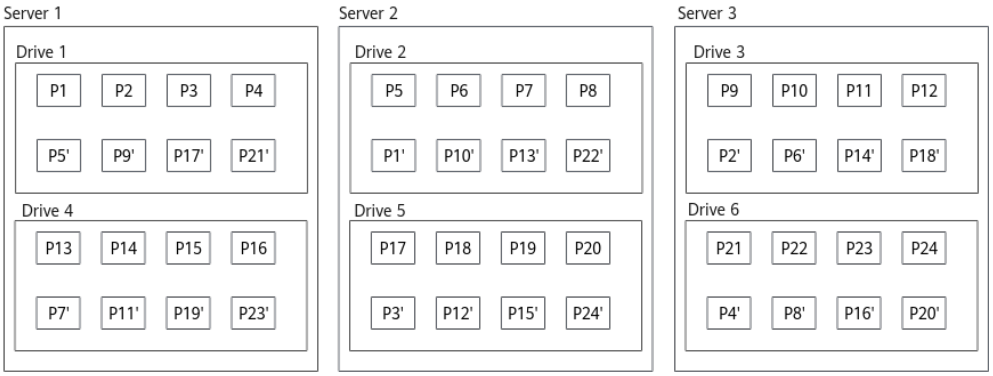
   When the OSD status and quantity are changed, the cluster map changes, which affects the mapping between PGs and OSDs.

2. Storage policy configuration (data security-related)

   The policy can allow three OSDs of the same PG to be located on different servers or racks, improving storage reliability.

   Figure 6-2 shows the multiple data copies. For data block P1 on drive 1 of server 1, its data backup is P1' on drive 2 of server 2. P1 and P1' constitute two copies of the same data block. For example, if drive 1 becomes faulty, P1' can take the place of P1 to provide storage services.

Figure 6-2 Replication in Ceph distributed storage



6.4.4 Erasure Coding

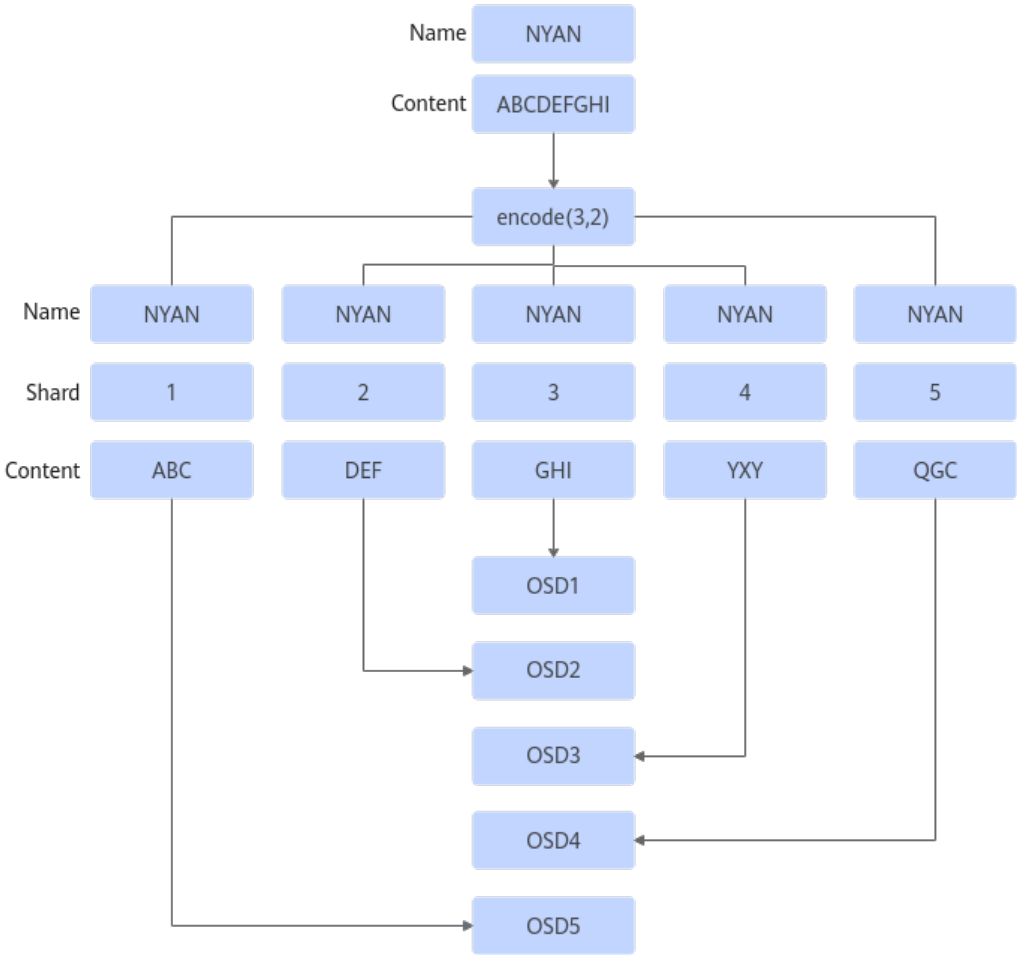
Erasure coding (EC) is a fault-tolerant coding technology. It was first used in the telecommunication industry to resolve the data loss problem during transmission. This mechanism segments the signals to be transmitted, add signal parity, and then associate the segments. Even if some signals are lost during transmission, the receive end can still calculate the complete information through an algorithm. In

data storage, the erasure coding mechanism divides data into segments, extend and encode redundant data blocks, and store them in different locations such as drives, storage nodes, or other physical locations.

The erasure code is a structure of *k* data blocks and *m* check blocks. The values of *k* and *m* can be set according to a specific rule. The formula is $n = k + m$. The variable *k* indicates the number of original data blocks. The variable *m* indicates the number of redundant data blocks for failure protection. The variable *n* indicates the total number of data blocks after the erasure coding mechanism is implemented. When less than or equal to *m* storage blocks (data blocks or parity blocks) are damaged, all data blocks can be obtained by calculating data in the remaining storage blocks, preventing data loss.

The following uses k=3 and m=2 as an example to describe how to store an object named NYAN in Ceph in erasure coding mode. Assume that content of the object is ABCDEFGHI. After NYAN is uploaded to Ceph, the client calls the erasure coding algorithm in the primary OSD to encode the data. The original data ABCDEFGHI is split into three segments, and then another two parity segments (the contents are YXY and QGC) are calculated. According to the rule specified by the CRUSH map, the five segments are randomly distributed on five different OSDs to complete the object storage, as shown in **Figure 6-3**.

Figure 6-3 Storing an object named NYAN in Ceph in EC mode



The following describes how to read data by using erasure coding. NYAN is used as an example. After the client sends a request for reading NYAN, the primary OSD (OSD1 in this example) of the PG where the object is located sends the request to other associated OSDs. If OSD4 is faulty and cannot respond to the request, only segments of OSD1 (GHI), OSD3 (YXY), and OSD5 (ABC) can be obtained. Although OSD2 receives the request and sends data, its data is received last. In this case, OSD1, as the primary OSD, decodes the data segments of OSD1, OSD3, and OSD5, and ignores the data segment of OSD2. The NYAN data (ABCDEFGHI) is restored and returned to the client.

6.5 I/O Passthrough

Overview

In balanced storage, system bottlenecks lie in HDDs. The I/O passthrough feature improves the throughput of HDDs based on how data is stored to HDDs in the Ceph cluster, thereby improving cluster performance.

Technical Principles

To ensure data reliability in the Ceph cluster, cache flush operations are performed when data is committed to drives. As a result, the write performance latency is prolonged. By bypassing such operations and allowing data to be directly persisted to drives, the data read/write efficiency can be improved.

Expected Results

In block storage where the block size is 4 KB, 16 KB, 64 KB, or 1 MB, the fio tool is used to test random mixed read/write (7:3) performance. The performance is improved by more than 15% when I/O passthrough is enabled.

6.6 EC Turbo

Overview

The EC feature of open source Ceph is mainly used for whole object read/write operations in object storage (as shown in **Figure 6-4**), and it does not support partial read/write which will cause severe read/write amplification in block storage. In this case, the read/write performance of small blocks is only a fraction of that in the replication solution. In addition, the number of cross-network accesses increases during the stripe read/write process of EC. As a result, the read/write performance deteriorates. EC Turbo supports partial read/write (as shown in **Figure 6-5**), which effectively mitigates read/write amplification and improves read/write performance.

Figure 6-4 Only full-stripe read/write allowed in open source Ceph

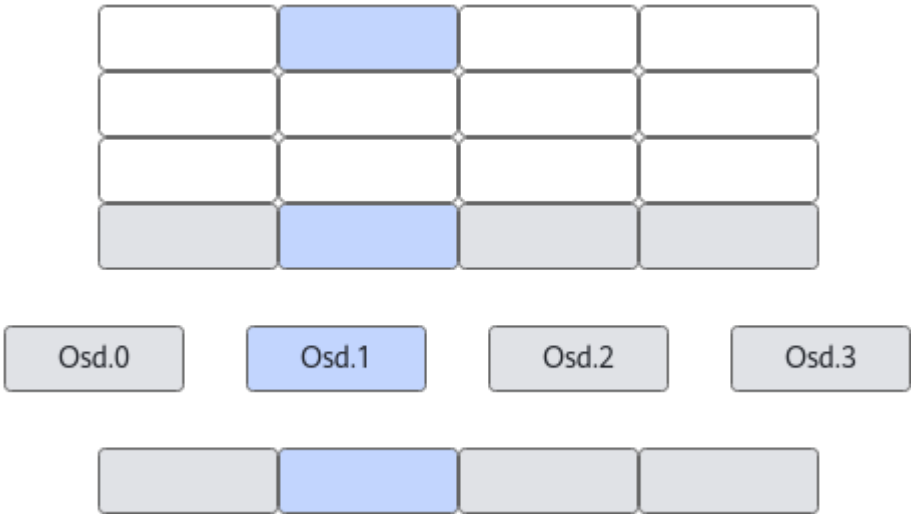
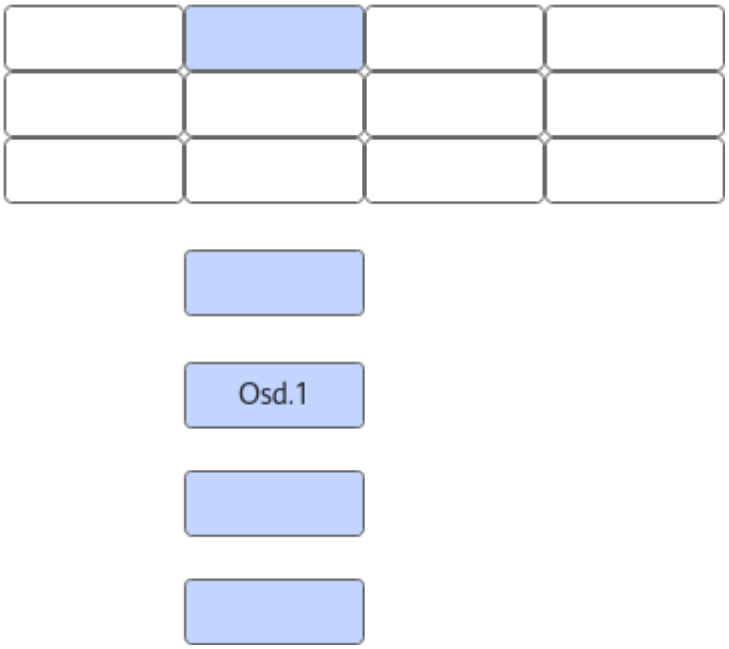


Figure 6-5 Partial read/write with EC Turbo enabled

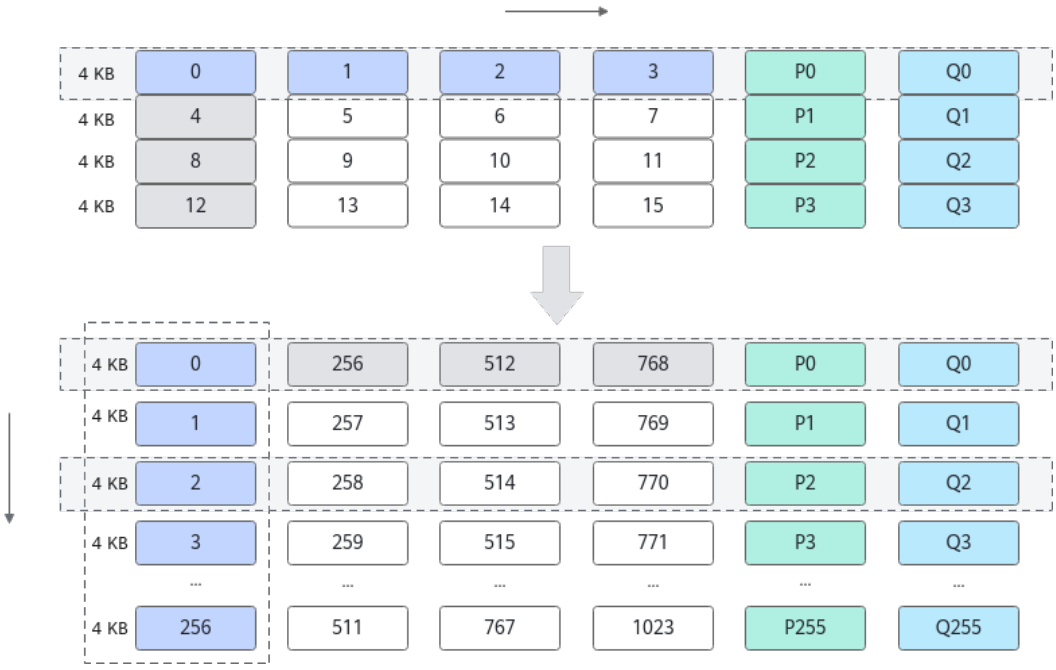


Technical Principles

In Ceph, the minimum storage unit size for an EC stripe is 4 KB. When a user reads or writes only 4 KB data, open source Ceph does not support partial modification. As a result, when a small data block is read or written, read/write amplification is severe. The EC Turbo feature enables partial modification of an EC stripe. Only required data blocks are read, reducing read/write amplification. When data to be read exceeds the 4 KB block size, cross-stripe read is required. In this case, an 8 KB operation is amplified into two 4 KB drive read operations. The performance is only half that of one local 8 KB read operation in the replication solution. EC Turbo enlarges the local storage unit size for an EC stripe (for example, to 1 MB) and supports modification of small stripes. For small I/Os such as 8 KB or 32 KB read, only one local read operation is required, similar to the replication solution. This improves read performance. In addition, operator pushdown is used to reduce

network transmission overhead, and refined striping strategies are used to eliminate unnecessary padding zeros in EC stripes. These methods reduce read/write amplification.

Figure 6-6 EC Turbo row-to-column store



Expected Results

The EC performance of open source Ceph is only about 40% of the replication solution. By reducing read/write amplification during the I/O process, EC Turbo improves the read/write performance of EC to over 80% of the replication solution.

6.7 Kunpeng Storage Acceleration Library

Overview

The Kunpeng Storage Acceleration Library (KSAL) is developed by Huawei. It contains the EC, CRC16, and CRC32 algorithms.

The EC algorithm replaces the high-order finite field GF(2^w) multiplication required in the EC process with binary matrix multiplication through isomorphism mapping and uses exclusive or (XOR) instead of complex finite field multiplication that is implemented via lookup tables. In addition, it uses an orchestration algorithm to reuse intermediate results in the parity block calculation process, which reduces XOR operations and accelerates coding by working with Kunpeng vectorized instructions.

The CRC16 and CRC32 algorithms optimized based on the principles of a large-number modulo algorithm are used to replace the standard CRC16 and CRC32 algorithms, respectively. They have better Kunpeng affinity, improving system performance.

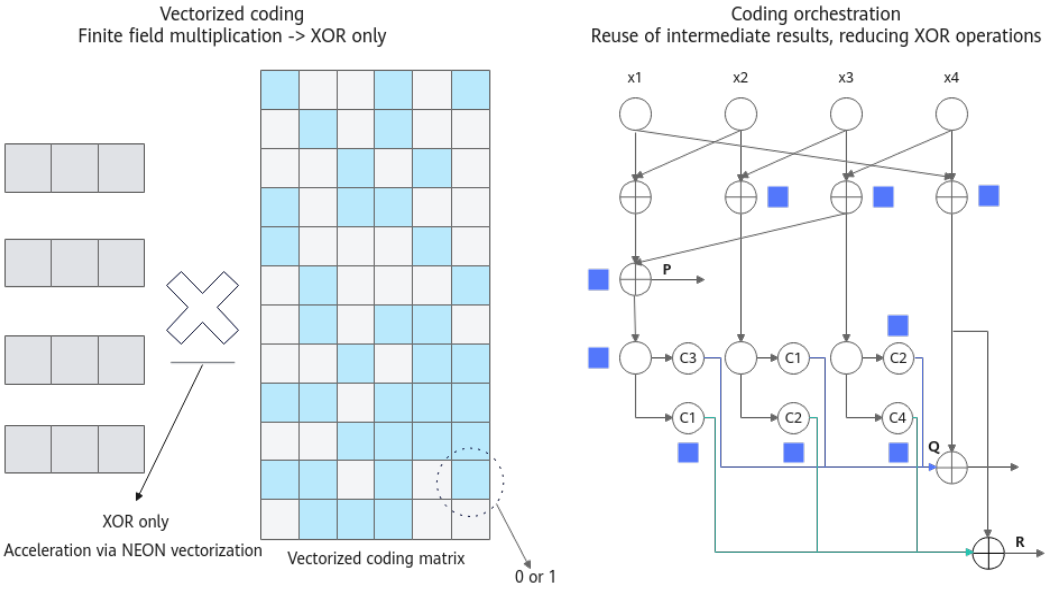
Technical Principles

EC

Algorithm principles:

Vectorized coding is used to replace the high-order finite field multiplication of traditional scalar coding (Jerasure, ISA-L) with low-order binary XOR operations (acceleration via vectorization). LUT and 10+ instruction operations are required. With coding orchestration, intermediate results are reused to reduce the number of operations, and thus the EC coding performance is greatly improved.

Figure 6-7 Multi-level acceleration via vectorized coding



Running context:

Plog client (index side)

Interfaces:

EC coding, decoding, reconstruction, and degradation read interfaces

CRC16 verification

Assuming that the data information sequence polynomial is M(x) and the primitive polynomial is P(x), the CRC is defined as follows:

$$CRC\left(M(x)\right) = \left(x^{\deg(P(x))}\, M(x)\right) \left(\mathrm{mod}\left(P(x)\right)\right)$$

$$\text{If } \overline{\mathrm{M}}(x) = x^{\deg(P(x))}\, M(x), \text{ then:}$$

$$CRC\left(M(x)\right) = \overline{M}(x) \left(\mathrm{mod}\left(P(x)\right)\right)$$

$$\text{If } Q(x) = P(x)\, T(x), \text{ then:}$$

$$CRC\left(M(x)\right) = R(x) \left(\mathrm{mod}\left(P(x)\right)\right), \text{ where } R(x) = \overline{M}(x) \left(\mathrm{mod}\left(Q(x)\right)\right)$$

Assume that the dividend is 10 and the divisor is 3. You can divide 10 by 6 first, and then divide the remainder by 3. That is, the remainder of 10 divided by 6 is 4, and the remainder of 4 divided by 3 is 1.

Because there are multiple calculation methods, the selection scope can be narrowed down based on the following criteria:

- The number of Q(x) non-zero terms corresponds to the algorithm complexity.
- The order of Q(x) corresponds to the number of registers required by the algorithm.
- The difference between the order of the highest-order term and the order of the second highest-order term of Q(x) corresponds to the degree of parallelism required by the algorithm.

Based on the application scenario, you can select a proper method according to the preceding criteria.

CRC32 verification

Algorithm principles:

```
#ifdef __aarch64__
#define CRC32D(crc, value) __asm__("crc32x %w[c], %w[c], %x[v]":[c]"+r"(crc):[v]"r"(value))
#define CRC32W(crc, value) __asm__("crc32w %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(value))
#define CRC32H(crc, value) __asm__("crc32h %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(value))
#define CRC32B(crc, value) __asm__("crc32b %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(value))
```

Running context:

Message verification and data consistency verification

Expected Results

EC

Compared with mainstream open source EC algorithms, the average coding throughput is doubled.

CRC16 verification

Compared with mainstream open source CRC16 algorithms, the 4 KB verification performance of this algorithm is doubled.

CRC32 verification

The CPU computing power consumed by a single I/O operation is reduced by more than 50%, and the overall gain is estimated to be 3%. When the block size is 4 KB, 8 KB, 64 KB, 256 KB, or 1 MB, the performance is twice that of ceph_crc32c_sctp and 1.2 times that of ceph_crc32_sctp.

6.8 Smart Write Cache

Overview

In balanced storage, to improve the read/write efficiency of HDDs, users generally use the built-in Bcache module of the Linux kernel to accelerate drive throughput. Although the acceleration effect is obvious, in scenarios where a large number of

small files are written, the IOPS fluctuates greatly or even drops to zero (as shown in **Figure 6-8**), affecting user experience. As shown in **Figure 6-9**, the smart write cache feature stabilizes data streams and improves throughput.

Figure 6-8 IOPS of an x86 model without smart write cache

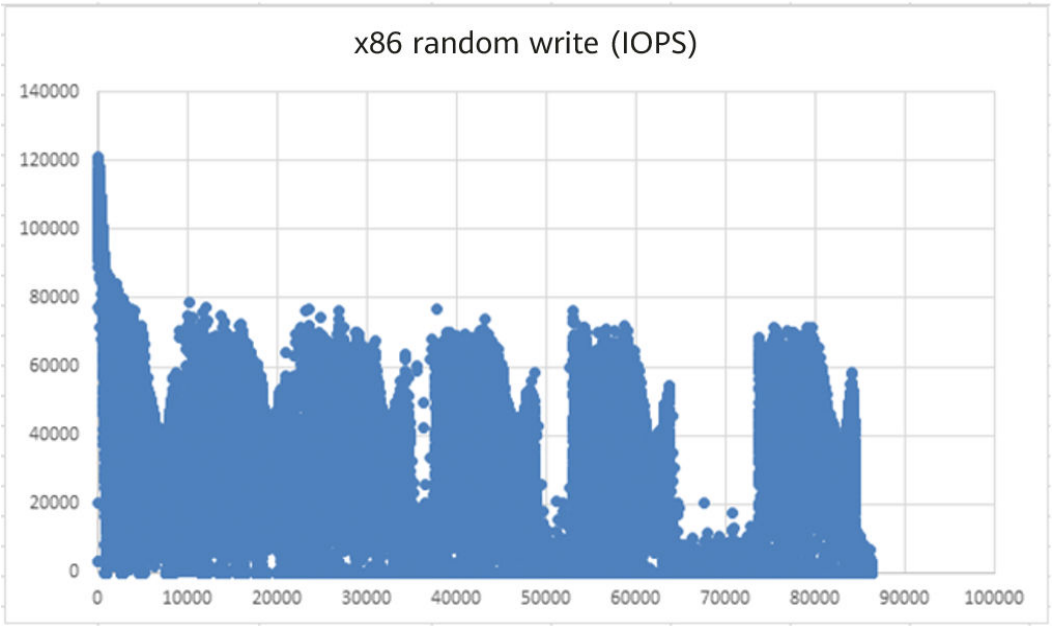
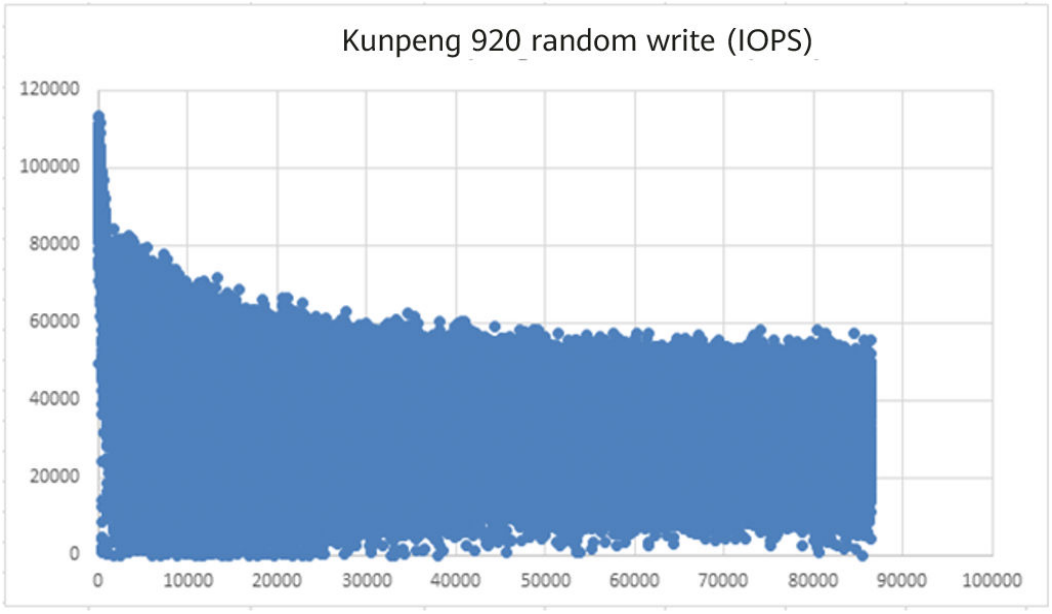


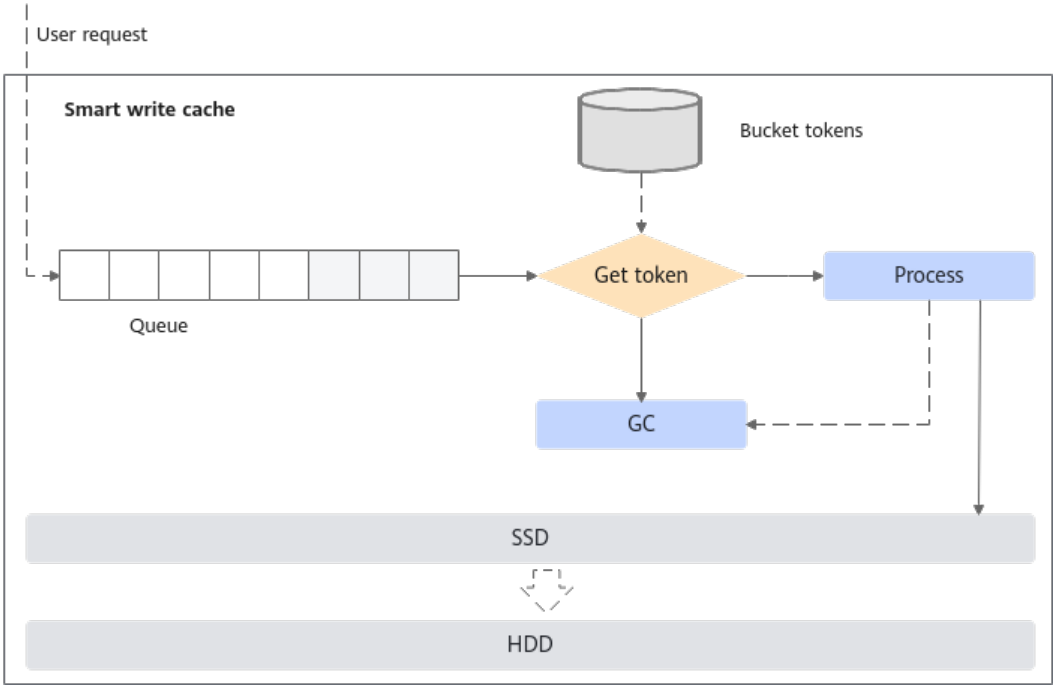
Figure 6-9 IOPS of a Kunpeng model with smart write cache enabled



Technical Principles

The Bcache performance is unstable because it does not have the QoS function and the cache efficiency is low. The smart write cache feature introduces bandwidth and IOPS traffic limiting measures and optimizes cache management to stabilize data streams and improve throughput.

Figure 6-10 Reduced IOPS fluctuation through the dual-token solution of smart write cache



Expected Results

The integration test with the Ceph cluster shows that the write performance is 1.2 times and the mixed read/write performance is 1.1 times the corresponding performance before optimization.

6.9 Compression Algorithms

Overview

LZ4 is a widely used compression algorithm with good compression performance. However, its compression ratio is low. In this feature, Huawei-developed algorithms and instruction acceleration are used to improve the compression ratio and write performance.

Technical Principles

The compression algorithm feature uses Huawei-developed compression algorithms, including the dynamic global history information table, hierarchical algorithm model, dynamic compression header, table header structure, NEON instruction optimization, and elastic matching window, to improve the data block compression ratio without decreasing the compression efficiency.

Figure 6-11 Dynamic global history information table

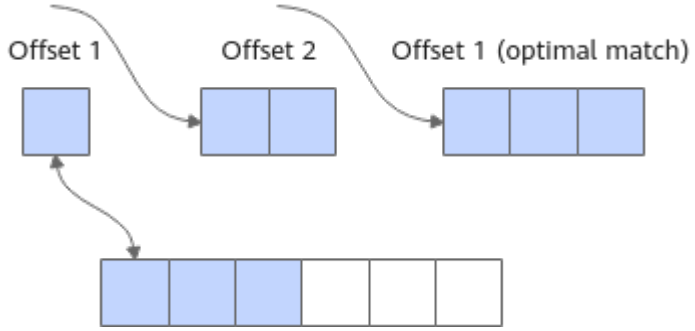
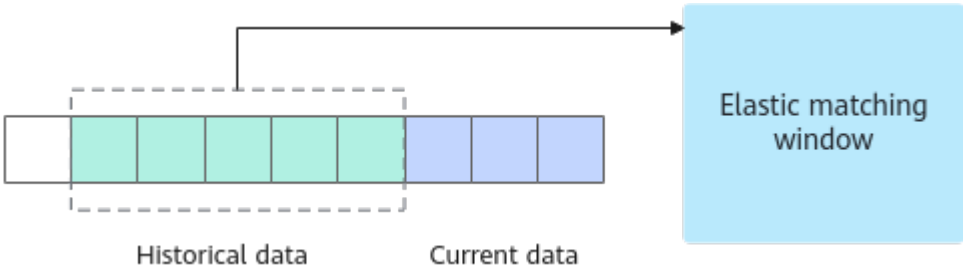


Figure 6-12 Elastic matching window



Expected Results

The self-developed compression algorithms improve the compression ratio by 25% compared with LZ4 and improve the write performance by 10% in balanced scenarios.

6.10 Data Compaction

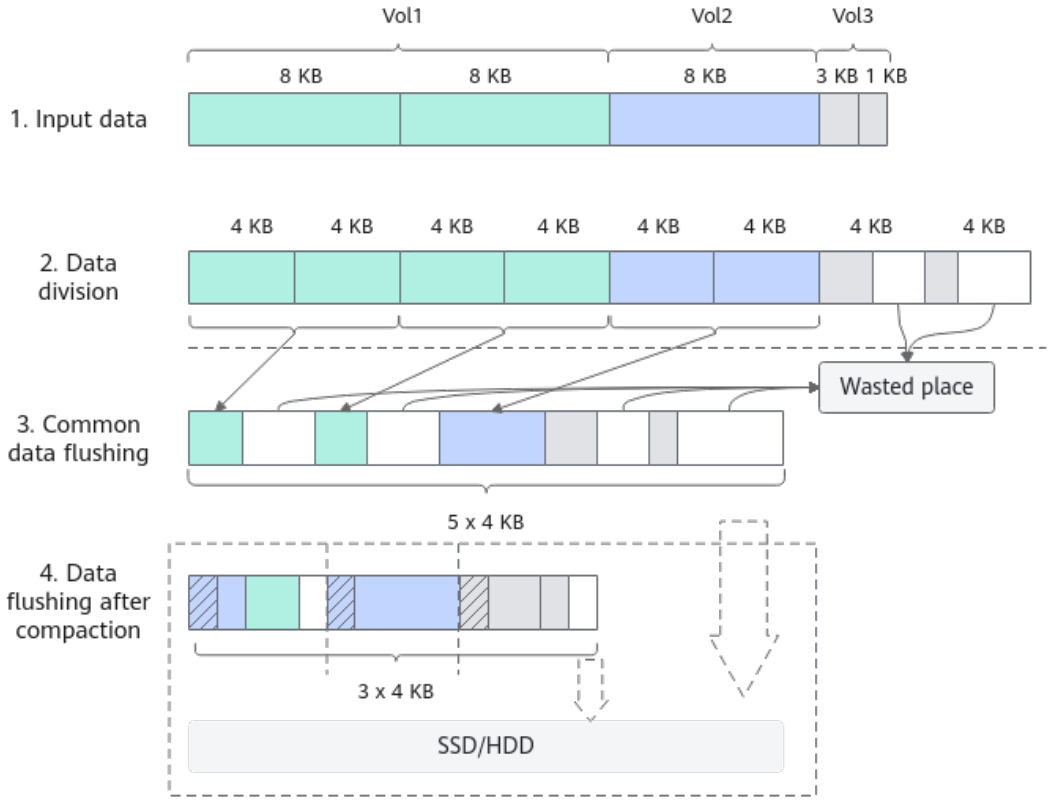
Overview

In the Ceph system, data blocks are flushed to drives based on 4 KB alignment. If an input data block does not conform to 4 KB alignment, it is padded with zeros. This wastes storage space, especially in HDD/SSD hybrid storage where compression is aligned to 64 KB. In this case, the space wasted by unaligned parts is even greater after data compression. The data compaction feature compacts different data blocks by byte and flushes them to drives by 4 KB, reducing space waste.

Technical Principles

The data compaction feature compacts the unaligned parts of data blocks to achieve higher storage density. In compression scenarios, compressed data blocks are not aligned. Through data compaction, storage space can be saved, and the compression ratio can be improved.

Figure 6-13 Data compaction process



Expected Results

The compression ratio is improved by more than 20%, and the performance of flushing data to drives does not deteriorate.

6.11 Smart Prefetch

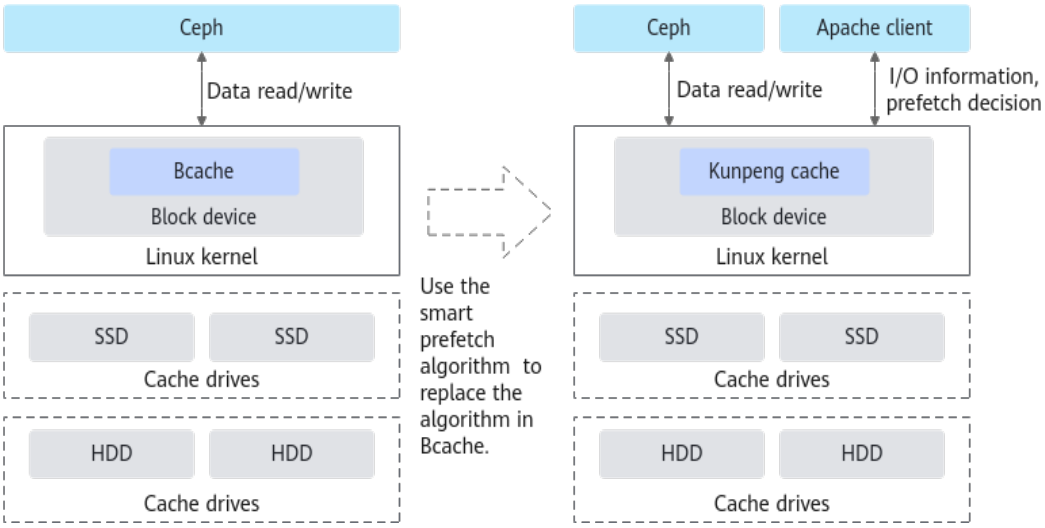
Overview

The built-in block device acceleration component Bcache of the Linux kernel caches hotspot data and sequentializes write addresses to improve read/write efficiency. However, in scenarios where data streams are sequential streams instead of hotspot data streams, the performance is poor. The smart prefetch feature identifies sequential read streams and uses the prefetch algorithm to prefetch valid data to the cache in advance, improving read performance.

Technical Principles

By identifying sequential data streams and prefetching data as large blocks to the cache in advance, this feature improves the read hit rate and drive throughput.

Figure 6-14 Technical principles of smart prefetch



Expected Results

In sequential read scenarios of block storage:

- If the I/O size is less than 512 KB, the performance is improved by more than 20%.
- If the I/O size is greater than 512 KB, the performance improvement is not obvious.

6.12 Ucache Read Cache

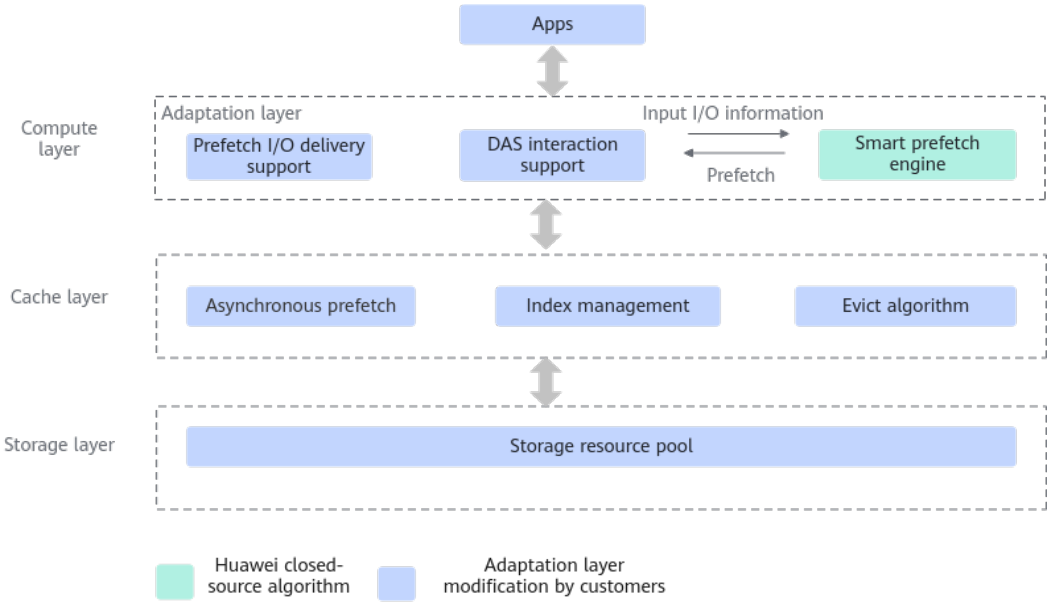
Overview

In a balanced storage system, data from different clients is aggregated to the backend. As a result, data block storage continuity is disordered, affecting read performance. The Ucache read cache system identifies the I/O patterns (sequential, reverse, interleaved, or hotspot) of user applications on a client and uses the prefetch policy to increase the data hit rate.

Technical Principles

The smart prefetch analysis engine and execution engine are deployed separately. The analysis engine uses multiple algorithms to identify sequential, reverse, interleaved, and hotspot streams, and prefetches target data to the read cache. Such data is directly hit in the cache, reducing read latency and drive read pressure.

Figure 6-15 Smart prefetch framework



Expected Results

The performance is doubled in various scenarios, including block storage services, hotspot, sequential, and interleaved read in a 4 KB cluster or single volume, workload proxy, and 4 KB read.

6.13 Metadata Acceleration

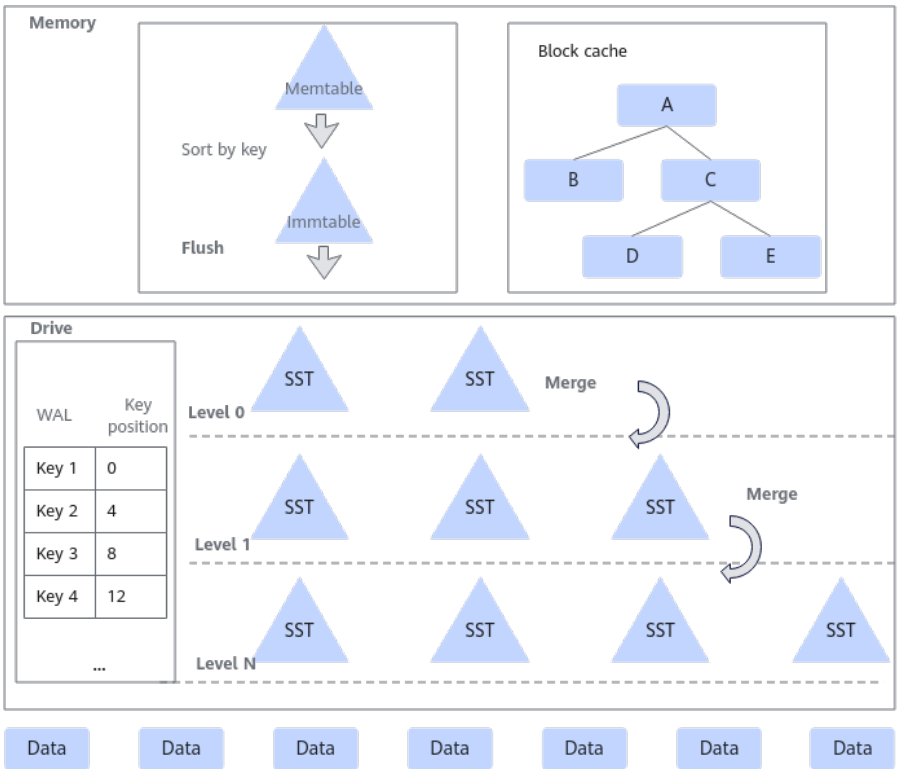
Overview

As the data volume increases, the read/write amplification of the RocksDB database backend increases exponentially. This acceleration feature optimizes the RocksDB compaction process to mitigate write amplification and improves the execution efficiency of hotspot functions based on Kunpeng instruction optimization.

Technical Principles

1. Metadata compression is used to improve the cache hit rate, and hardware-software collaboration, function tuning, and computing overhead reduction improve the comprehensive capabilities of RocksDB.
2. KAEZstd hardware offload is used to accelerate background compression.

Figure 6-16 Metadata acceleration framework



Expected Results

1. Compared with the open source RocksDB database, the mixed read/write performance of 64B/128B, 64B/512B, and 64B/1024B key-values is improved by 30% after this feature is enabled.
2. Compared with CPU software compression, the key-value write performance is improved by 10%.

6.14 Kunpeng Storage Maintenance Library

Overview

To solve the problem that user services are affected by slow drive I/O response, a traditional method is to identify faulty drives based on the time threshold of slow I/O response provided by experts. This method overlooks the following scenarios: (1) A drive is rejected when I/Os on the drive are slow but upper-layer services are not affected. (2) The I/O response of a drive does not reach the threshold, but services are affected. The Kunpeng Storage Maintenance Library (KSML) uses the SMART slow drive detection algorithm to identify faulty drives together with machine learning algorithms based on drive I/O data and upper-layer service latency statistics.

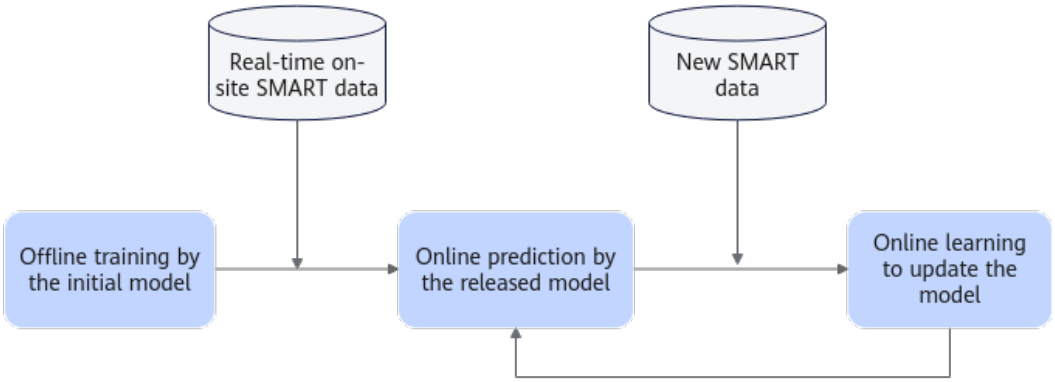
Technical Principles

A common drive fault detection method is to identify faulty drives based on the time threshold of slow I/O response provided by experts. However, this threshold-based method is not suitable in the following scenarios:

- A drive is rejected when I/Os on the drive are slow but upper-layer services are not affected.
- The drive I/O does not reach the threshold, but services are affected.

To handle such pain points, this feature provides an intelligent fault prediction algorithm based on machine learning on SMART data to predict faulty drives that affect user services, which requires no experience in determining the threshold. Alarms can be generated before services are affected, and customers can handle the faults in a timely manner to prevent faulty drives from affecting service functions.

Figure 6-17 KSML working principle



A distributed storage system typically consists of multiple nodes, and each node has multiple drives. The drives jointly store data through logical data division. Due to sector status and external environment differences, the I/O request processing duration of different drives may vary. As a result, I/O response is slow, and services may be interrupted, affecting cluster performance. If slow drives can be detected in advance when services are running, service isolation can be performed to reduce long-tail latency in the cluster and improve cluster stability. The slow drive detection feature can be used to collect w_await information of system drives, identify and process abnormal drive data, and confirm the drive status.

Figure 6-18 Slow HDD/SSD detection process

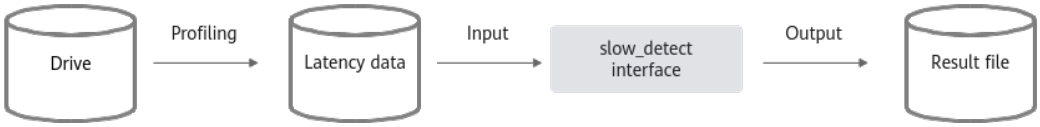
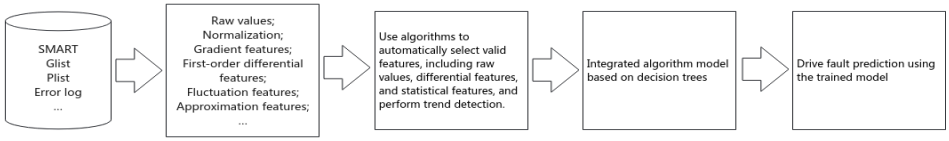


Figure 6-19 Drive fault prediction process



Expected Results

- Slow HDD/SSD detection: SATA HDD/SSD FDR > 80%, precision > 70%
- HDD fault prediction: SATA HDD FDR > 60%, FAR < 0.5%

- SSD fault prediction: SATA SSD FDR > 80%, FAR < 0.3%

6.15 KAE-enabled SPDK

Overview

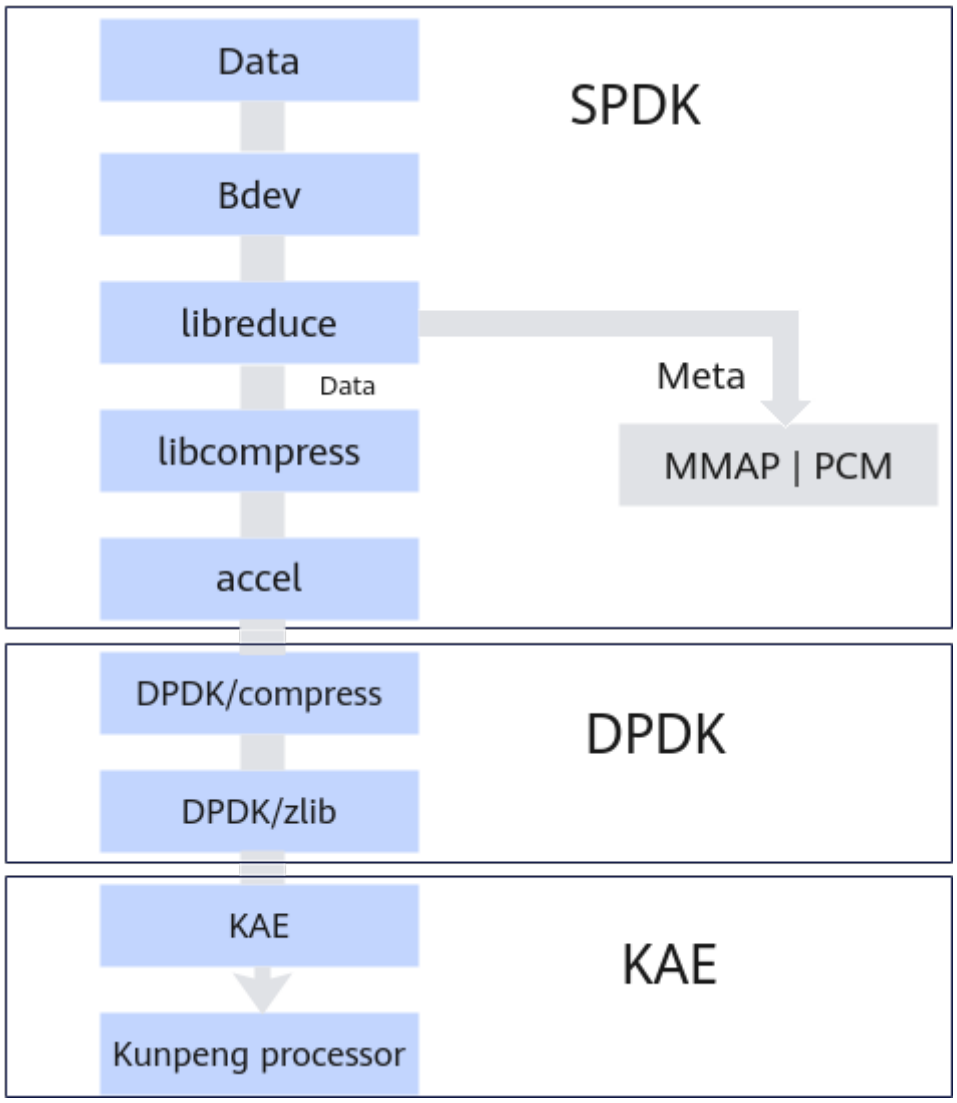
As the virtual device layer, the Storage Performance Development Kit block device (SPDK bdev) interconnects with underlying virtual and physical devices. By enabling compression, encryption, and decryption in the bdev, all SPDK devices can be supported. The Kunpeng Accelerator Engine (KAE) performs compression, encryption, and decryption using zlib and OpenSSL. Specifically, KAE is enabled for the bdev to implement hardware offload of these capabilities.

Technical Principles

KAE is enabled for the SPDK bdev to accelerate data processing.

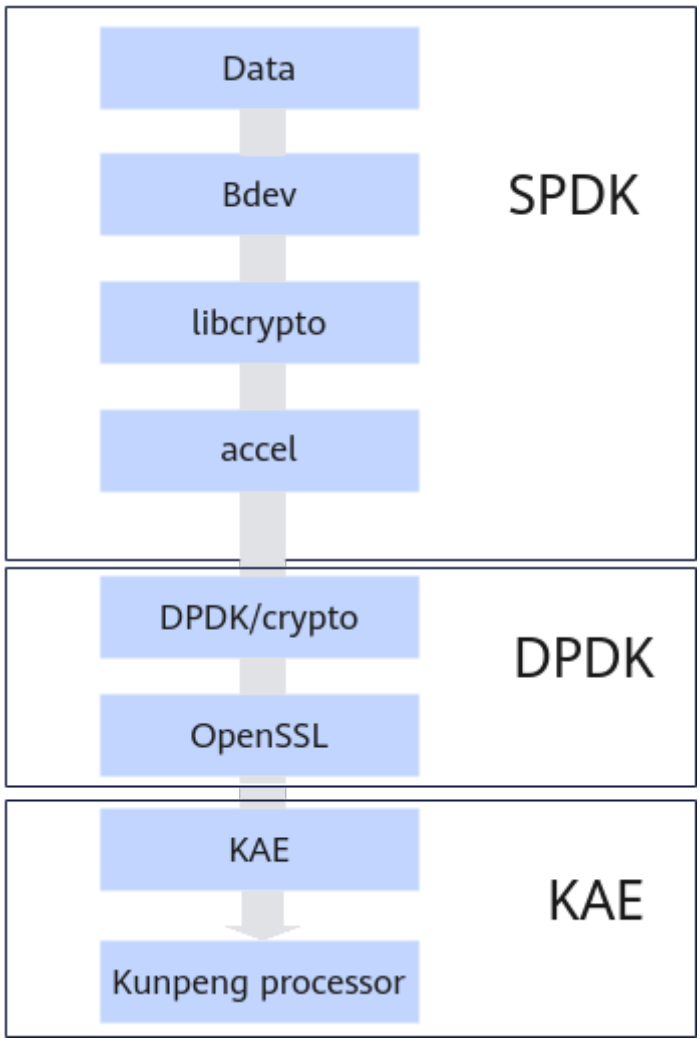
1. Use KAE to offload the compression overhead of the SPDK bdev, improving SPDK compression performance.

Figure 6-20 KAE compression enabled



2. Use KAE to offload the crypto overhead of the SPDK bdev, improving SPDK crypto performance.

Figure 6-21 KAE encryption and decryption enabled



Expected Results

The compression performance is doubled when KAE zlib and gzip are enabled, and the crypto performance is improved by 20% when KAE SM4 is enabled.

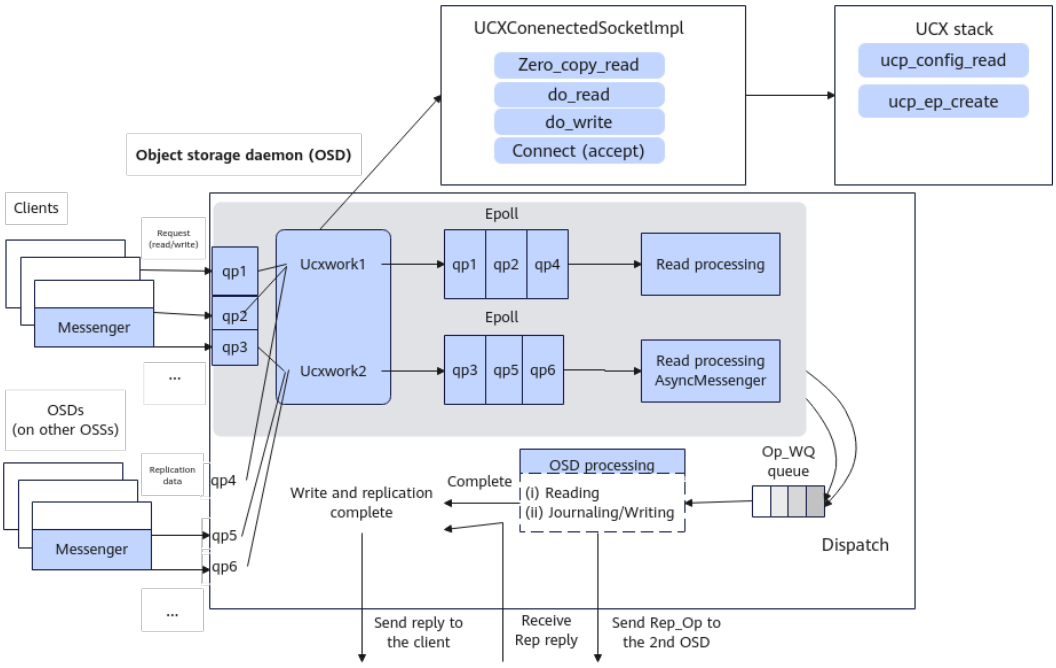
6.16 UCX RDMA Network

Overview

A plugin is applied to the Ceph network framework AsyncMessager to support the UCX network framework, which enables full RDMA in Ceph all-flash storage. The UCX communication processing layer adapts Ceph and UCX interfaces and implements zero copy based on rendezvous (RNDV) protocol characteristics to improve large-block write performance.

Technical Principles

Figure 6-22 Ceph RDMA network communication framework



Expected Results

RDMA accelerates network processing and reduces the read/write latency. The cluster performance within 1 ms can be improved by 10%.

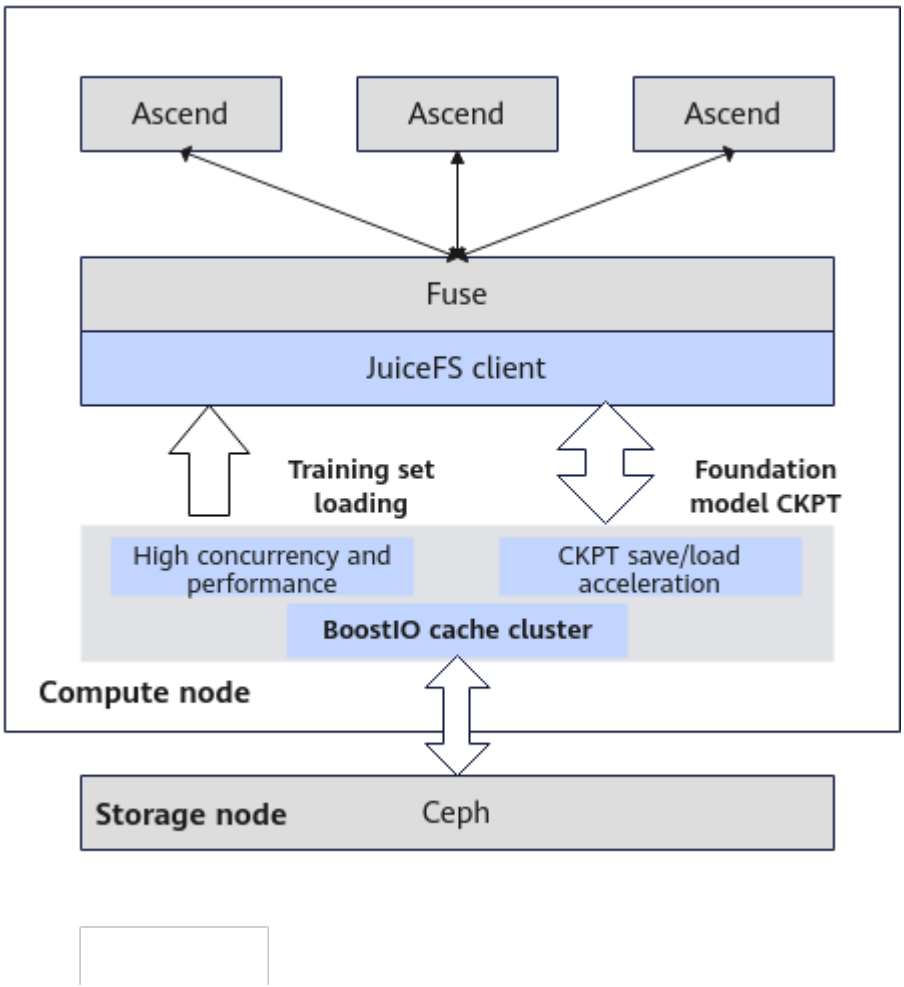
6.17 BoostIO

Overview

In the decoupled storage and compute architecture, BoostIO uses memory and drive resources on the compute side to build a distributed multi-tier cache. The write cache uses RDMA high-speed communication, cache affinity, data replication, and linear layout characteristics to improve service write performance and data reliability. The read cache pre-loads hotspot data to cache drives through data prefetch and leverages the LRU and cold/hot data identification algorithms to improve the read cache hit rate, thereby improving the read performance.

Technical Principles

Figure 6-23 BoostIO framework



Expected Results

This feature can be used in AI foundation model applications to increase the large block read-write performance by more than 300% in checkpoint (CKPT) and dataset loading scenarios.

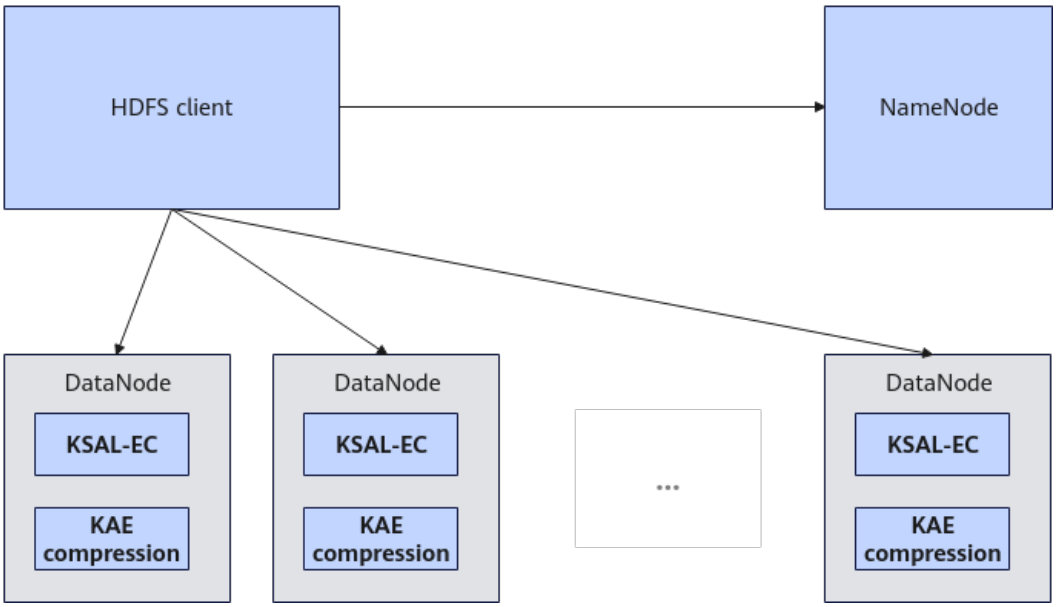
6.18 KAE and KSAL for HDFS

Overview

The EC algorithm of open source HDFS is replaced with the KSAL EC algorithm to improve the computing efficiency. KAE hardware-based compression is enabled to enhance data flushing.

Technical Principles

Figure 6-24 KAE and KSAL acceleration for HDFS



Expected Results

The cluster throughput is improved by 20% after the optimization.

6.19 SPDK I/O Acceleration

Overview

The SPDK is used to accelerate I/O flushing in Ceph all-flash storage and reduce I/O latency.

Technical Principles

Key technologies:

User space only: This feature moves all necessary drivers to the user space to slash system call overheads and implement zero copy.

Polling: For high-speed physical storage devices, the use of polling instead of interrupt notifications to determine request completion reduces latency and minimizes performance fluctuations.

Lock-free: No lock mechanism is used for synchronization on I/O paths to reduce latency and improve throughput.

Figure 6-25 Ceph I/O write process before the optimization

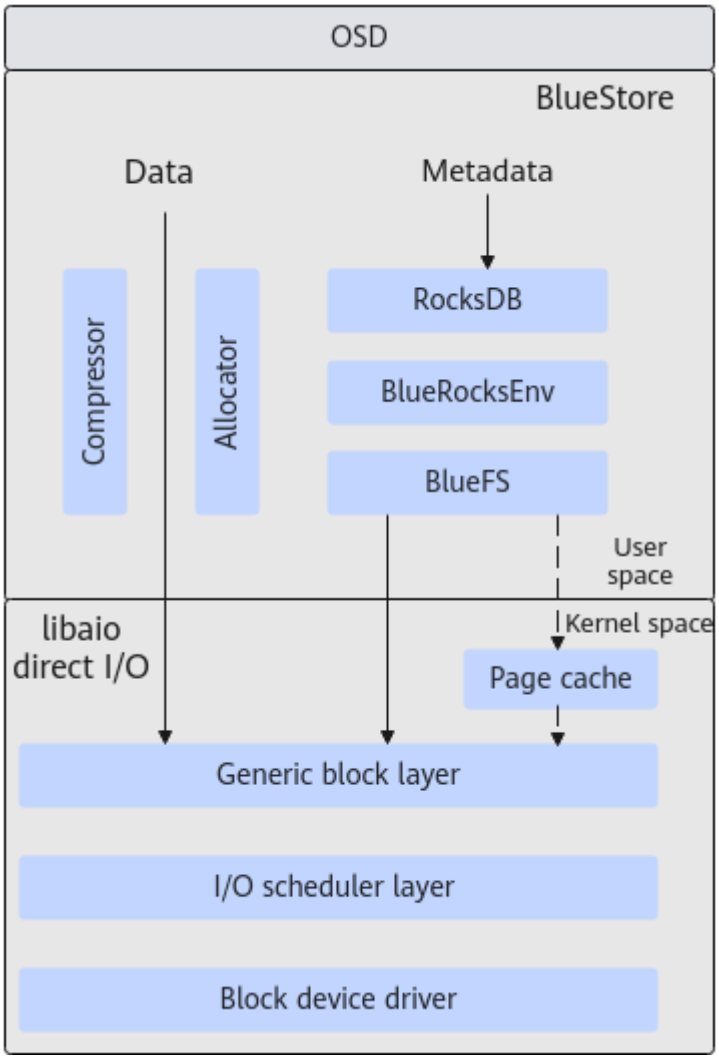
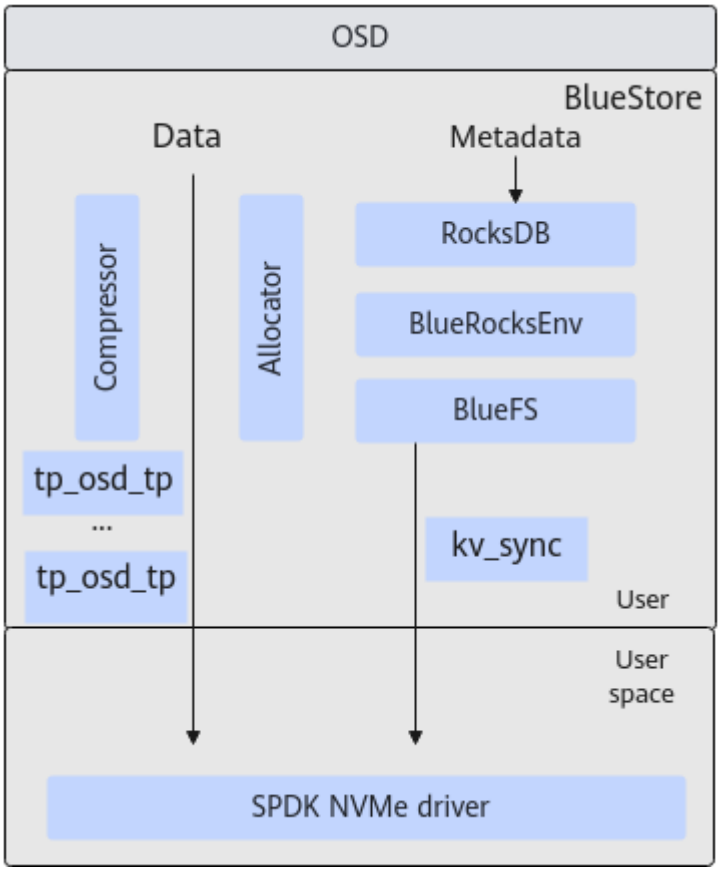


Figure 6-26 Ceph I/O write process after the optimization



Expected Results

The latency of a single Ceph 4 KB I/O is reduced from 600 μs to 400 μs.

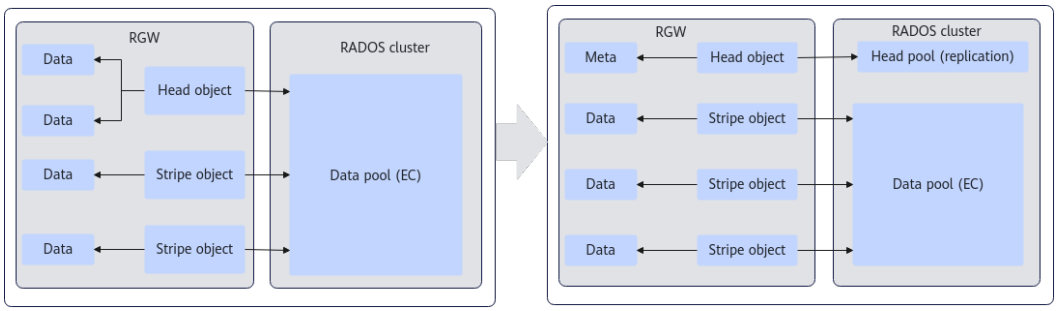
6.20 Ceph Object Storage Metadata Reduction

Overview

In Ceph object storage, to reduce data amplification at the backend, the data pool adopts the EC mode instead of replication. In EC mode, the metadata amplification factor is the number of EC copies, and metadata amplification is severe. This feature extracts xattr metadata of RGW objects from the data pool into a new pool adopting the replication mode to reduce metadata amplification.

Technical Principles

Figure 6-27 Ceph metadata reduction solution



Metadata stored in EC mode is changed to the replication mode to reduce metadata write amplification. Ceph RGW object metadata can be stored in NVMe SSDs and will not overflow to HDDs, preventing performance deterioration caused by metadata overflow.

Expected Results

Compared with the solution before metadata separation, metadata (excluding object maps) is reduced by 15% under the same data volume. In extreme conditions with 10 billion objects, the read/write performance decreases by no more than 50% compared with that of 100 million objects (100 KB block size).

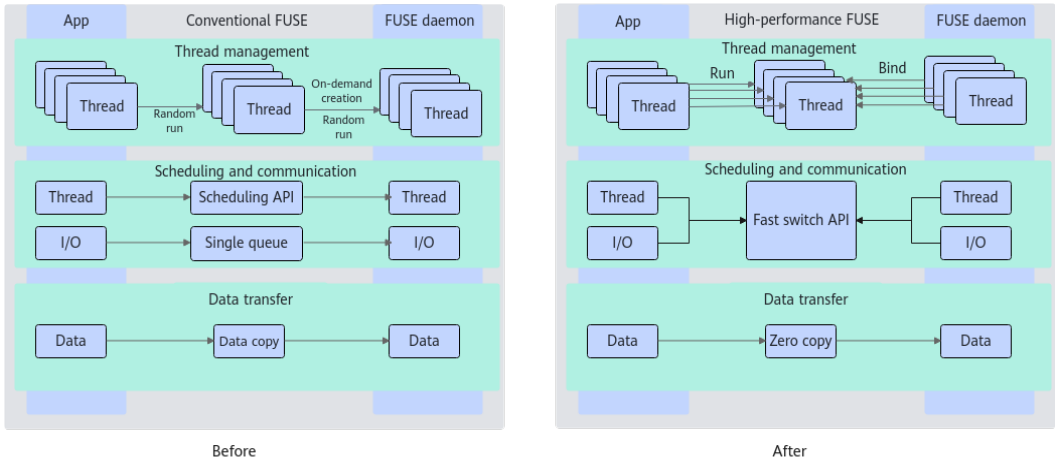
6.21 FUSE Kernel Space Tuning

Overview

In AI scenarios, the Filesystem in Userspace (FUSE) is widely used, which imposes higher requirements on I/O latency. The open source Linux FUSE has high latency due to overheads of frequent thread creation/destruction and data copy between the user space and kernel space.

Technical Principles

Figure 6-28 FUSE kernel space tuning principles



1. **Thread-to-core binding modification**

   A fixed number of threads are created for I/O transmission between the FUSE kernel and libfuse and are bound to CPU cores. In the FUSE kernel, the read/write request structures are bound to threads one by one, which reduces the overhead of thread switching between CPU cores, thereby improving overall I/O performance.

2. **Metadata copy-free through MMAP**

   Metadata of each read/write is transmitted through MMAP, reducing the number of memory copies and improving I/O performance.

Expected Results

Both the multi-concurrency 4 KB random read/write performance and multi-concurrency 1 MB sequential read/write performance are improved by 20%.

6.22 Kunpeng Instruction-based ISA-L Optimization

Overview

The Intelligent Storage Acceleration Library (Intel® ISA-L) is widely used in distributed storage system software. This feature accelerates the CRC32 and EC algorithm APIs based on Kunpeng vectorized instructions.

Technical Principles

- CRC32 algorithm optimization

  Optimization 1: The 3-way CRC32 instruction parallelism of open source ISA-L can be expanded to 6-way parallelism on the new Kunpeng 920 processor model. This optimization improves the computing performance under cache full-hit conditions. Furthermore, prefetch optimization tailored for small data blocks are integrated to improve the performance during cache miss occurrences.

- EC algorithm optimization

  Optimization 1: The first parity block of EC calculation is calculated by using the XOR operation instead of table lookup.

  Optimization 2: Prefetch offset optimization is added to improve the cache hit ratio.

Expected Results

The optimized CRC32 and EC (EC 10+1, 4+2, and 8+3) algorithms deliver 10% and 20% performance gains, respectively, over open source ISA-L 2.31.

7 Feature List

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
Hybrid deployment	x86-Kunpeng hybrid deployment	In block/object/file storage services, x86 and Kunpeng servers are deployed in the same storage pool. In block/object/file storage services, Kunpeng servers are used to expand the capacity of an x86 server cluster.	• Supported OSs: CentOS 7.6 and RHEL 7.5. • Constraint: The same Ceph version and OS version must be used.	-	No
Bcache	Bcache	In block/object/file storage services, NVMe SSDs are used as the cache of HDDs to improve system performance.	• Supported OSs: CentOS 7.6 and openEuler 20.03 LTS. • Constraint: The OS pagesize needs to be changed from 64 KB to 4 KB.	Patch package: Kernel patch	No

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
I/O passthrough	I/O passthrough	I/O pass-through improves the storage performance in balanced configuration and 7:3 read/write hybrid scenarios.	• Supported OSs: CentOS 7.6 and openEuler 20.03 LTS. • Constraint: When Bcache is configured, I/O passthrough does not improve performance.	RPM package: Contact Huawei technical support.	No
Certification by commercial-edition Ceph for Ubuntu	Kunpeng server (model 2280)	The Kunpeng server (model 2280) has passed the certification by commercial-edition Ceph for Ubuntu.	-	-	-
	Kunpeng server (model 5280)	The Kunpeng server (model 5280) has passed the certification by commercial-edition Ceph for Ubuntu.			

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
Compression algorithms	Data compression for block storage service	Compared with the open source LZ4 compression, the Kunpeng BoostKit for SDS compression algorithm delivers a 25% higher compression ratio and a 25% lower effective capacity cost per TB.	• Supported OSs: CentOS 7.6 and openEuler 20.03 LTS SP1. This feature can be used on other OSs by compiling the source code. • Constraint: This feature is invalid for incompressible data sources such as images and videos.	Binary package: Contact Huawei technical support.	No
		In the balanced configuration, the Kunpeng BoostKit for SDS compression algorithm delivers a 10% higher bandwidth performance than the open source LZ4 compression.			
	Data compression for object storage service	Compared with the open source LZ4 compression, the Kunpeng BoostKit for SDS compression algorithm delivers a 25% higher compression ratio and a 25% lower effective capacity cost per TB.			

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
		In the balanced configuration, the Kunpeng BoostKit for SDS compression algorithm delivers a 10% higher bandwidth performance than the open source LZ4 compression.			
Smart write cache	Performance acceleration for write operations in block storage and object storage services	In the balanced configuration, it prevents the random write performance from dropping to zero and increases the random write performance using three-copy data pools by 20%. It also improves the mixed read/write (7:3) performance by 10%.	• Supported OSs: CentOS 7.6 (kernel 4.14) and openEuler 20.03 LTS SP1 (kernel 4.19). • Constraint: On CentOS, the smart write cache and smart I/O prefetch features cannot be used together, while on openEuler, the two features can be used together. • Storage engine: BlueStore.	RPM package: Contact Huawei technical support.	No

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
EC Turbo	Block storage service	In the balanced configuration, the EC Turbo (4+2) performance reaches over 80% of the x86 three-copy performance, and the storage cost is reduced by 50%.	• Supported OSs: CentOS 7.6 and openEuler 20.03 LTS SP1. • Supported software version: Ceph 14.2.8. • Constraints: – Use the block or object storage service in mixed read/write (7:3). – The Bcache feature is not supported.	RPM package: Contact Huawei technical support.	No
	Object storage service	In the balanced configuration, the EC Turbo (4+2) performance reaches over 80% of the x86 three-copy performance. The storage cost of large I/Os is reduced by 50%, and the storage cost of small I/Os is approximately the same.			
Ucache read cache	Block storage service	In balanced configuration, the 4 KB hotspot read performance is improved by 100%.	Supported OSs: openEuler 20.03 LTS SP1 openEuler 22.03 LTS SP2	Patch package: Contact Huawei technical support.	No

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
Metadata acceleration	Database service	In all-flash scenarios, the mixed read/write performance of 64B/128B, 64B/512B, and 64B/1024B key-values is improved by 30% after this feature is enabled compared with the open source RocksDB database.	Supported OSs: openEuler 20.03 LTS SP1 openEuler 22.03 LTS SP2	Patch package: Contact Huawei technical support.	No
KSML	Reliability service	Slow HDD detection: SATA HDD FDR > 80%, precision > 70% HDD fault prediction: SATA HDD FDR > 60%, FAR < 0.5%	Supported OSs: openEuler 20.03 LTS SP1 openEuler 22.03 LTS SP2	RPM package: Contact Huawei technical support.	No
Data compaction	Block storage service	In a data compression scenario with the balanced configuration, the compression ratio of "Kunpeng + data compaction + data compression" is 1.3 times that of "x86 + open-source data compression."	• Supported OSs: CentOS 7.6 and openEuler 20.03 LTS SP1. • Supported software version: Ceph 14.2.8. • Constraint: Use the block or object storage service.	Patch package: Contact Huawei technical support.	No
	Object storage service	In a data compression scenario with the balanced configuration, the compression ratio of "Kunpeng + data compaction + data compression" is 1.3 times that of "x86 + open-source data compression."			

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
KSAL	EC coding and decoding	The vectorized EC coding and decoding scheme replaces the high-order finite field multiplication of traditional scalar encoding with low-order binary XOR operations and reuses intermediate calculation results through coding orchestration to reduce the number of operands.	• Supported OS: openEuler 20.03 LTS SP1. • Supported software version: Ceph 14.2.8. • Constraint: The EC algorithm offers 2+2, 4+2, 6+2, 8+2, 8+3, 12+3, 20+3 and 25+4 configuration acceleration and allows configurations 25+4 and below. Other configurations are not supported.	RPM package: Contact Huawei technical support.	No
	CRC16 verification	The CRC16 algorithm optimized based on the principles of a large-number modulo algorithm is used to replace the standard CRC16 algorithm. It has better Kunpeng affinity, improving system performance.			
	CRC32 verification	The CRC32 algorithm optimized based on the Kunpeng platform is used to replace the standard CRC32 algorithm, improving system performance.			

Feature	Sub-feature	Feature Description	Constraint	Required Software Package	Supported on VMs
Object storage metadata reduction	Object storage service	Metadata is separated from application object data by setting different storage classes for buckets and objects in the buckets.	-	**Metadata reduction patch**	No
SPDK I/O acceleration	Block storage service	A Ceph cluster is deployed in containers on Kunpeng servers running openEuler 20.03. The integration of the SPDK, UCX, and KSAL maximizes storage and network performance, answering the need for high throughput and low latency in modern distributed storage.	The SPDK I/O acceleration feature adopts a container-based deployment of SPDK and Ceph 17.2.7. SPDK + other distributed storage modes are not supported.	**SPDK I/O acceleration package**	No

8 Typical Configurations

All-flash configuration:

Item	Configuration
Server	Kunpeng server (model 2280)
CPU	2 x Huawei Kunpeng 920 5250 processor
Memory	16 x 16 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Data drive	12 x 3.2 TB NVMe SSD

Typical configuration 1 of the balanced model:

Item	Configuration
Server	Kunpeng server
CPU	2 x Huawei Kunpeng 920 3210 processor
Memory	12 x 16 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Acceleration SSD	1 x 3.2 TB NVMe SSD

Item	Configuration
Data drive	12 x 8 TB HDD

Typical configuration 2 of the balanced model:

Item	Configuration
Server	Kunpeng server
CPU	2 x Huawei Kunpeng 920 5220 processor
Memory	12 x 16 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Acceleration SSD	2 x 3.2 TB NVMe SSD
Data drive	36 x 8 TB HDD

Typical configuration 3 of the balanced model:

Item	Configuration
Server	Kunpeng server (model 5290)
CPU	2 x Huawei Kunpeng 920 5220 processor
Memory	16 x 6 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), 2 x Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Acceleration SSD	2 x 3.2 TB NVMe SSD
Data drive	72 x 8 TB HDD

Typical configuration 1 of cold storage

Item	Configuration
Server	Kunpeng server (model 5280)
CPU	2 x Huawei Kunpeng 920 5220 processor
Memory	8 x 16 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Data drive	36 x 8 TB HDD

Typical configuration 2 of cold storage

Item	Configuration
Server	Kunpeng server (model 5290)
CPU	2 x Huawei Kunpeng 920 5220 processor
Memory	12 x 16 GB DDR4-2933 MHz
Network	Onboard NIC (4 x GE), 2 x Huawei IN200 NIC (4 x 25GE)
RAID controller card	Avago SAS 3508
OS drive	2 x 480 GB SATA SSD
Data drive	72 x 8 TB HDD

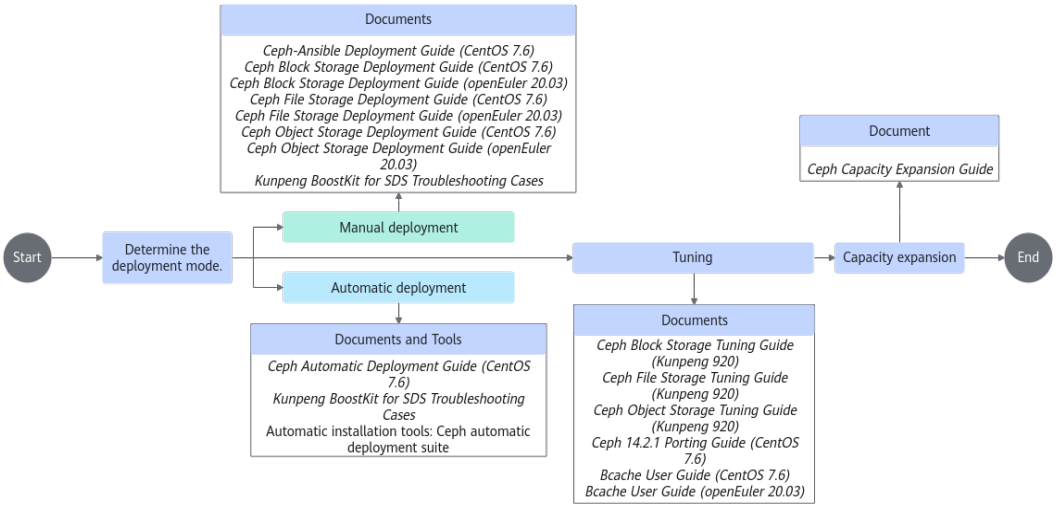
9 Software Compatibility

Use the **Compatibility Checker** to obtain information about software supported by Kunpeng BoostKit for SDS.

10 Process

Figure 10-1 shows the process.

Figure 10-1 Process



11 References

Some documents on the Ceph official website are referenced in **3 SDS Cluster Architecture**.

Ceph official website: **https://ceph.readthedocs.io/en/latest/**

A Change History

Date	Description
2025-12-30	This issue is the seventh official release. Added **6.22 Kunpeng Instruction-based ISA-L Optimization**.
2025-06-30	This issue is the sixth official release. Added **6.21 FUSE Kernel Space Tuning**.
2025-03-30	This issue is the fifth official release. Added the following: • **6.19 SPDK I/O Acceleration** • **6.20 Ceph Object Storage Metadata Reduction**
2024-12-30	This issue is the fourth official release. Added **6.18 KAE and KSAL for HDFS**.
2020-05-20	This issue is the third official release. Adjusted the document structure and optimized the figures.
2019-12-31	This issue is the second official release. Updated based on the solution version.
2019-09-24	This issue is the first official release.

