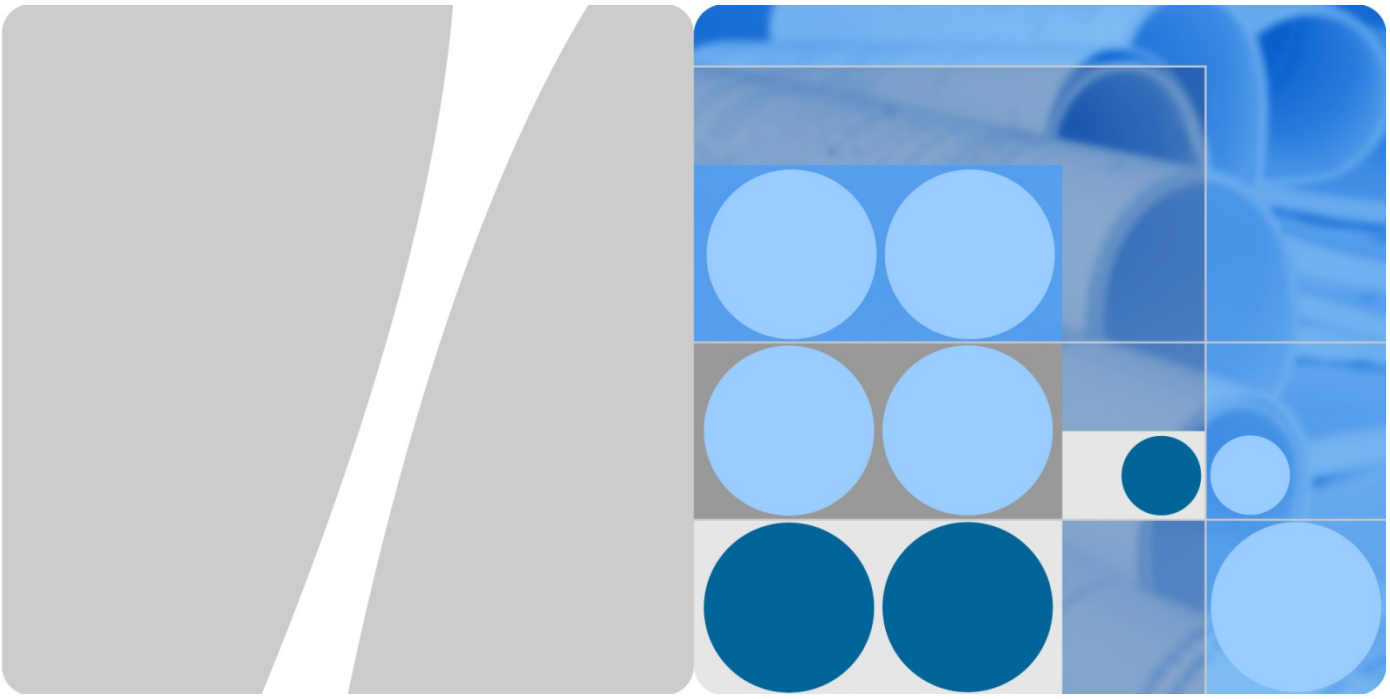


Huawei OceanStor Dorado V3 All-Flash Storage Systems

Performance Technical White Paper

Issue 1.1

Date 2018-10-30

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.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

Contents

1 Executive Summary

This document describes the performance design and optimization of OceanStor Dorado V3 all-flash storage systems in terms of software and hardware architectures that involve Huawei-developed SSDs (called HSSDs), high-performance cache, multi-core scheduling design, stable latency assurance mechanism, and FlashLink.

2 Product Overview

OceanStor Dorado V3 all-flash storage systems are optimized and adjusted in terms of system hardware, SSDs, I/O software stack, and space configuration. The storage systems provide stable and optimal performance with a low latency upon inline deduplication and compression, support Scale-up (capacity expansion) and Scale-out (performance expansion), and meet different customers' mission-critical service requirements.

3 System Architecture

OceanStor Dorado V3 all-flash storage systems use Huawei OceanStor OS as the operating system and inherit abundant features of the operating system. The SSDs of the storage systems have been optimized to deliver optimal performance at a cost-effective price. In addition, the storage systems are able to work with other products that use OceanStor OS, provide block storage services for external applications, and support various host ports such as 8 Gbit/s, 16 Gbit/s, and 32 Gbit/s Fibre Channel ports, 10 Gbit/s, 25 Gbit/s, 40 Gbit/s, and 100 Gbit/s Ethernet iSCSI ports, as well as high-speed low-latency 56 Gbit/s InfiniBand (IB) ports.

The Scale-out technology adopted by OceanStor Dorado V3 expands the number of controller enclosures to ensure linear growth of capacity and performance at a stable low latency whereas the Scale-up technology adds disk enclosures and disks to expand the capacity. When the capacity increases, the back-end data balancing mechanism automatically balances data among all disks.

3.1 Hardware Architecture

OceanStor Dorado V3 has two hardware forms: integration of disks and controllers and separation of disks and controllers. As for integration of disks and controllers, the disks and controllers are integrated into a 2 U controller enclosure. As for separation of disks and controllers, the controllers and disks are separately installed in a 3 U or 6 U controller enclosure and a 2 U disk enclosure respectively. Controller and disk enclosures are connected in Scale-up mode using the redundant SAS 3.0 technology. Controller enclosures are connected in Scale-out mode using the redundant PCIe 3.0 technology to achieve data and heartbeat backup.

3.1.1 Hardware Capabilities

As for hardware design, OceanStor Dorado V3 uses high-performance front-end ports, back-end ports, and internal nodes to ensure outstanding performance and sufficient hardware capabilities.

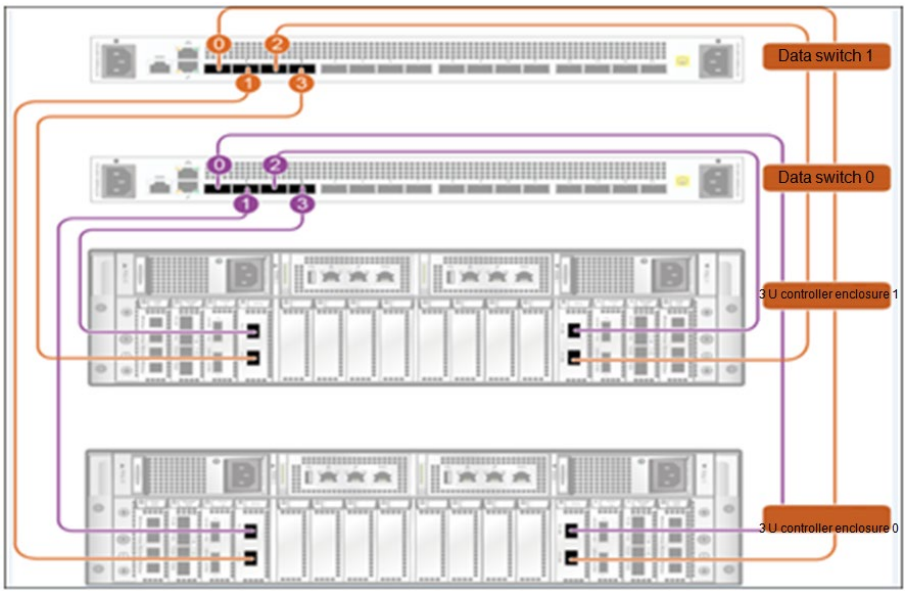
1. Processor: Each controller is equipped with Intel Xeon multi-core processors.
2. Front-end ports: The storage systems support 16 Gbit/s and 32 Gbit/s Fibre Channel ports, 10GE, 25GE, 40GE, and 100GE ports, as well as 56 Gbit/s IB ports. Each controller supports a maximum of 28 ports.

3. Back-end bandwidth: Each 25-slot disk enclosure can be directly connected to a controller enclosure through 4 x (4 x 12 Gbit/s SAS links). The bandwidth of each disk enclosure can be up to 192 Gbit/s.
4. Cluster: Controller enclosures are interconnected through PCIe 3.0 switches.

3.1.2 Scale-out Interconnection

OceanStor Dorado6000 V3 is used as an example here. The Scale-out network is as follows:

Figure 3-1 Dorado6000 V3 Scale-out network

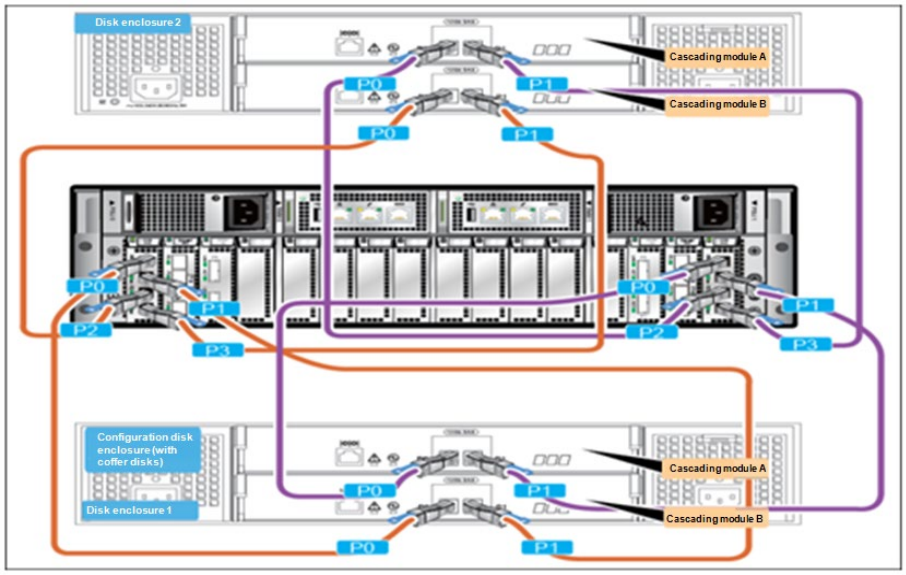


OceanStor Dorado V3 uses dual-channel PCIe 3.0 interconnection with low latency and high bandwidth to implement Scale-out, ensuring performance and reliability. In addition, the performance increases linearly as the controller quantity increases, achieving concurrent expansion of both capacity and performance.

3.1.3 Scale-up Interconnection

OceanStor Dorado6000 V3 is used as an example here. The Scale-up network is as follows:

Figure 3-2 Dorado6000 V3 Scale-up network



Upon capacity expansion by Scale-up, all disk enclosures of OceanStor Dorado V3 are directly connected to controller enclosures through 12 Gbit/s SAS links instead of cascading disk enclosures, ensuring stable low latency and linear bandwidth expansion.

The controller and disk enclosures are connected in redundant switchover mode and controllers A and B are working in load balancing mode. Each controller is connected to expansion modules through dual SAS 3.0 ports (dual-uplink network) in redundant backup mode for disks to fully exert their performance.

SAS 3.0 is used and the back-end network is optimized. Scale-up of capacity helps exert the advantages of more disks, delivering lower latency for I/O access.

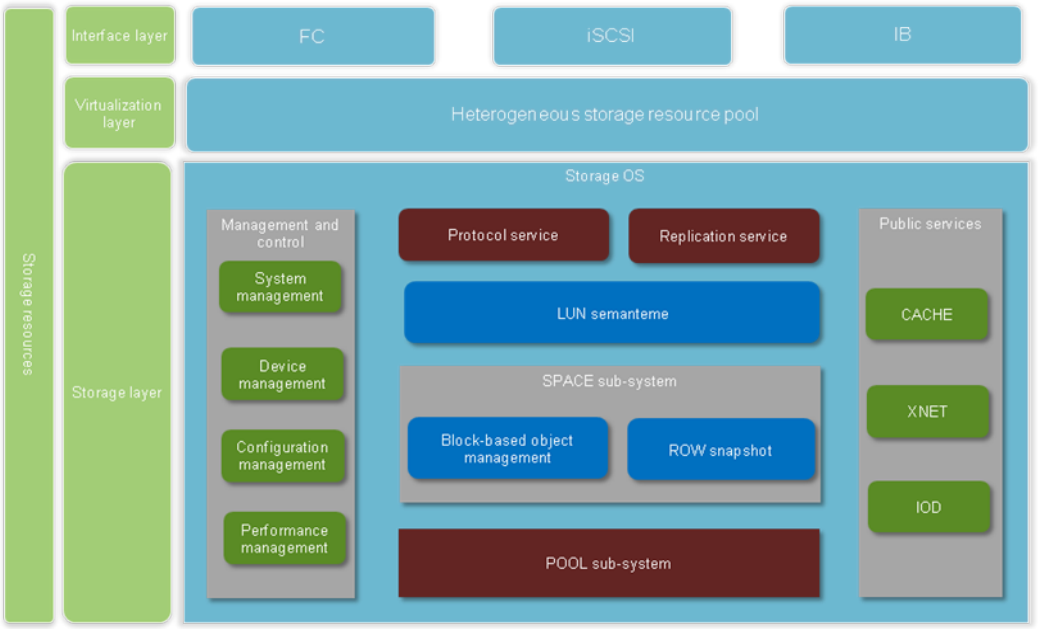
3.2 Software Architecture

The OceanStor OS software architecture consists of two planes: control plane and service plane. The control plane is designed for user configurations and device maintenance and management on a friendly humane-machine interface. The control plane analyzes user configuration policies and controls system events and service running status. The service plane is designed for disk space management as well as I/O functions and value-added functions. The two planes are supported by an infrastructure platform that provides an environment for software running and cluster management.

From the perspective of service types, the service plane consists of two layers: service layer and basic function layer. The service layer is configured for differentiated services, including protocol and replication services. The basic function layer tends to storage efficiency and reliability, including cache, Xnet (communication infrastructure that supports cross-controller and cross-engine request forwarding), and IOD (multi-core unified scheduling framework oriented to storage services).

OceanStor Dorado V3 adopts the OceanStor OS V3 architecture targeting at SSDs and stable-latency and high-performance products. The core technologies include Huawei FlashLink, high-performance hardware selection, high-performance cache, and system scheduling.

Figure 3-3 OceanStor OS software architecture

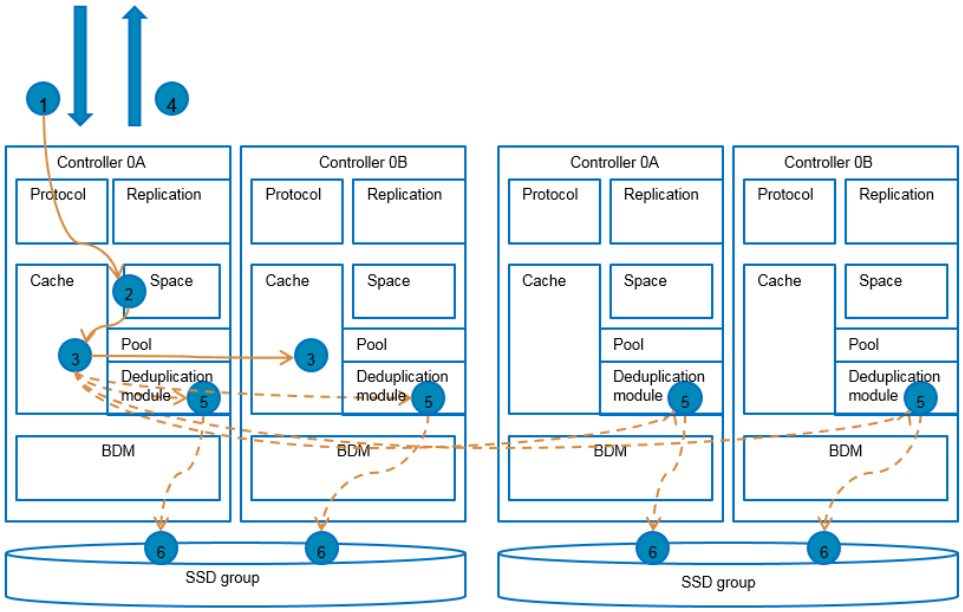


OceanStor Dorado V3 is optimized based on the characteristics of reads/writes and erase of flash granules and all-flash arrays. Key improvements are as follows:

1. Full-stripe write and system garbage collection reduce system write amplification and improve IOPS capability of the system.
2. Core grouping, I/O priority, and efficient metadata cache ensure stable system latency shorter than 1 ms.
3. Intelligent multi-core scheduling mechanism and lock-free design for I/Os reduce scheduling overheads and improve system IOPS.
4. Hardware FTL acceleration, disk and controller cooperation, and the read first and write cache policy of HSSDs reduce the latency of HSSDs and ensure stable system latency shorter than 1 ms.

3.2.1 Write Process

Figure 3-4 Data write process of OceanStor Dorado V3



1. A write I/O enters the LUN space after being parsed at the protocol layer.
2. The I/O is distributed to the corresponding controller for parsing.
3. I/O data is written to the local cache and synchronized to the cache of the mirrored controller.
4. The I/O is returned to the host.

NOTE

The I/O process is completed. The following illustrates how the I/O is handled in the background in the system.

5. Cache starts data flush to deduplicate and compress data in the pool. The I/O is processed in the pool as follows:
6. The pool divides the received data into data blocks at a fixed length (grain size).
7. The data blocks are distributed to controllers according to the LBA.

NOTE

If deduplication is enabled, steps 3 and 4 will be executed. If deduplication is disabled, the system directly executes step 5.

8. The fingerprint of each data block is calculated. Then, data blocks are distributed to controllers according to the fingerprints.
9. Query the fingerprint table to obtain the location of data of a certain fingerprint.
   - If the same fingerprints exist in the fingerprint table:

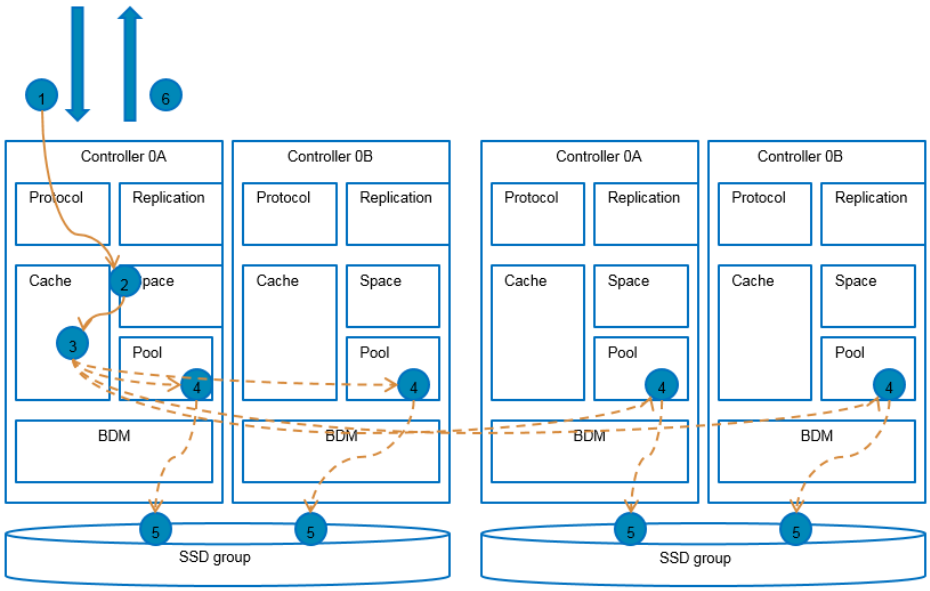
     Obtain the locations where related data is stored, and compare the data with data blocks byte by byte. If they are the same, the system increases the reference count of the fingerprints and does not write the data blocks to disks.
   - If all fingerprints in the fingerprint table are different or the comparison of data and data block shows they do not match:

     The system compresses the data block and writes it to disks later.

10. Write full-strip data into SSDs.
    - If the write strip is fully written (if the write strip is not fully written in a specified period, zeros are added), the parity check is executed, and the data and parity data are written into disks together.
11. The BDM writes the data to disks.
12. The pool updates the fingerprint table.
13. The cache refreshes the data cache status and deletes data in the mirrored cache. The write process is complete.

3.2.2 Read Process

Figure 3-5 Data read process of OceanStor Dorado V3



When a read I/O is delivered to the storage array, the data read process is as follows:

1. A read I/O enters the LUN space after being parsed at the protocol layer.
2. The I/O is distributed to the corresponding controller for parsing.
3. The cache determines whether the cache is hit.
    - Cache hit: The system obtains data from the cache and returns a read success to the host.
    - Cache miss: The I/O is sent to the pool for further processing.
4. The pool segments the I/O into data blocks of a fixed size (grain size) and sends these blocks to different controllers for processing based on the LBA.
    - Query the LBA-to-fingerprint mapping table to acquire the corresponding fingerprint, and send these fingerprints to controllers.
    - Query the fingerprint-to-address mapping table to acquire the storage address. Read data from the address.
5. Read data and decompress the data to obtain the original data.
6. The read I/O success message is returned to the host.

4 Superb Performance

With the purpose of providing superb performance to customers, Huawei makes innovative changes in hardware, disks, I/O software stack, and space allocation management of OceanStor Dorado V3.

4.1 High-Performance HSSDs

HSSDs use the industry-leading hardware architecture, Hi1812 chips designed for OceanStor Dorado, and DDR4 with lower power consumption and stronger performance. Up to 18 NAND flash channels are supported. In addition, HSSDs use the unique FTL architecture. Hardware acceleration is implemented on I/O paths. Multiple accelerators work together to reduce the I/O latency and improve the throughput of disks. The stream identification technology is used to reduce the write amplification of HSSDs and improve the write performance of HSSDs.

4.1.1 FTL Hardware Acceleration

The flash translation layer (FTL) in the HSSDs is a core data structure that saves the mapping relationship between LBA addresses and the physical pages in the HSSDs. An LBA address goes with an I/O read/write request. When HSSDs receive the I/O request, they query the physical page of this LBA address from the FTL table, therefore accessing the data. In the case of data read, traditional SSDs find the physical address for the LBA address, read data from the flash, and then return the data to the host. In the case of data write, the software writes data and then updates the FTL mapping table. HSSDs use the hardware acceleration FTL table. All reads/writes are completed by hardware. This reduces software interactions, improves IOPS, and shortens latency, as shown in the following figures.

Figure 4-1 Software FTL management

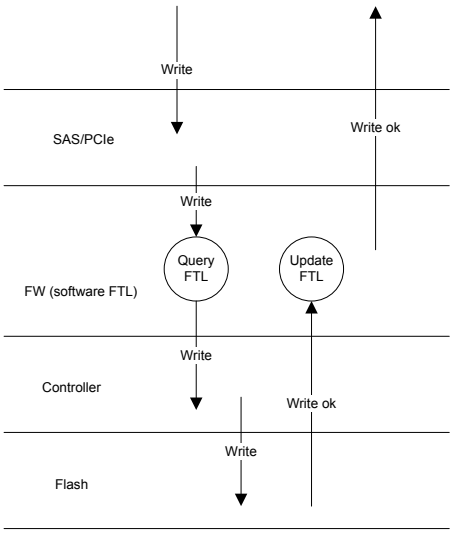
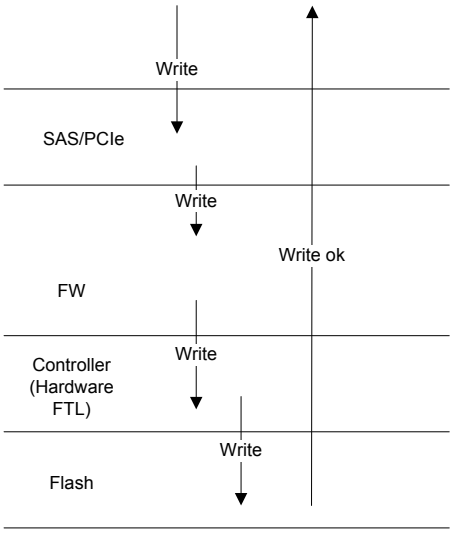


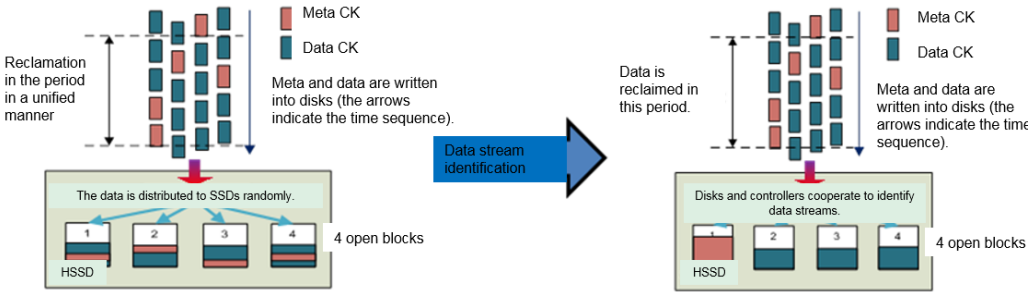
Figure 4-2 Hardware FTL management



4.1.2 Data Stream Identification

To improve garbage collection efficiency of HSSDs, reduce write amplification, and accelerate data writes, the system divides the data written to HSSDs into multiple data streams based on the update frequency. HSSDs identify these data streams and write the data of the same characteristics together to consecutive physical spaces on disks.

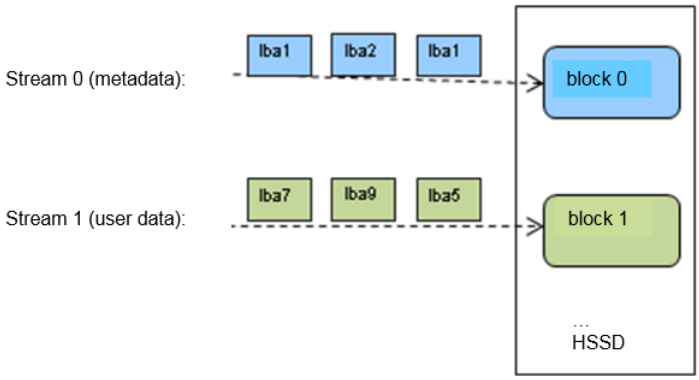
Figure 4-3 Data stream identification' impact on reclamation



Write data stream identification

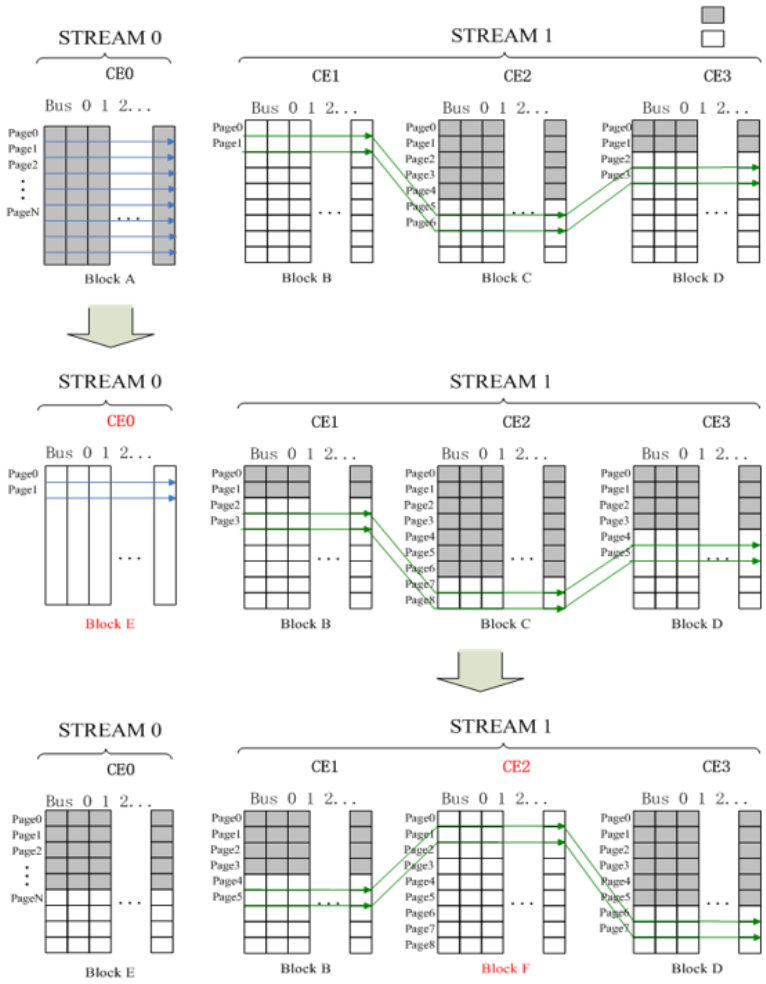
Huawei storage arrays identify cold and hot data using FlashLink technology and put the data of the same access frequency into a logical zone for management. Different data is distinguished based on LBAs.

Figure 4-4 Data stream identification process



After receiving data of different LBAs, HSSDs write the data in the same zone to the same block based on the zones preconfigured in the system. HSSDs support multi-channel concurrency to ensure disk performance.

Figure 4-5 Data distribution in the disks after data stream identification



CE0 to CE3: stripe 0 to stripe 3 on disks

STREAM0: metadata stream

STREAM1: user data stream

4.2 High-Performance Cache

For host write I/Os, after a write I/O enters the cache and is mirrored, it is returned to the host. Efficient cache processing is an important means to ensure high IOPS and low latency. OceanStor Dorado V3 cache is built based on flash features.

In traditional mechanical disk arrays, cache is an important component for system performance improvement. Cache reduces the disk seek time by sorting and reduces the interactions with back-end disks by aggregating continuous data. An I/O's response time of an HDD is about 6 ms (rotation time + seek time). Cache spends dozens of us in executing complex sorting and search operations. The time used by cache is less than 2% of the overall response time. Therefore, reducing the moving time of the head can remarkably improve performance. However, the read/write latency of SSDs' I/Os is stably low. The response time of one I/O ranges from 200 us to 500 us, and the proportion of cache sorting and searching

increases. As a result, the overhead is too large, adversely affecting system performance. The cache of Dorado V3 does not have the sorting and search algorithms. The cache delivers data to the pool in high concurrency mode so that the pool can aggregate full-stripe data and write the data into disks. This improves the back-end processing bandwidth and ensures the optimal write performance of hosts.

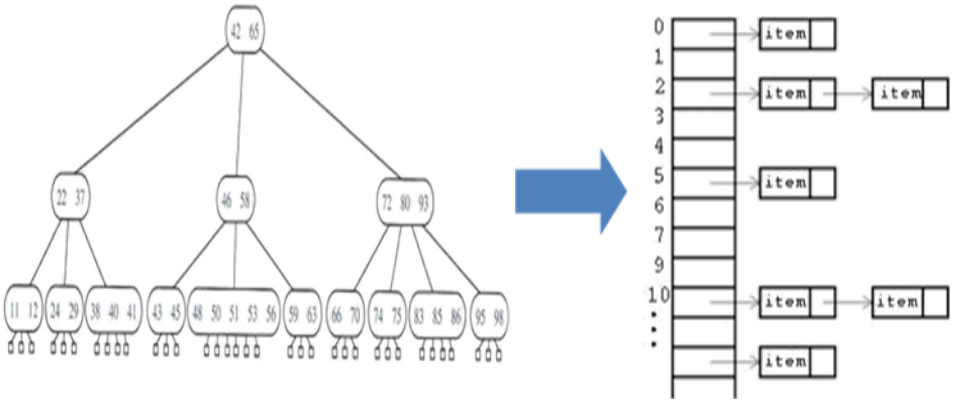
4.2.1 Efficient Data Search Algorithm

To improve search efficiency, the all-flash cache query algorithm uses the hash algorithm instead of the binary tree algorithm of the traditional storage arrays. Therefore, the search efficiency is improved from two aspects: reducing both search time and lock conflicts, thereby greatly shortening the data query time.

Cache constructs the number of hash table buckets based on the number of cached data blocks. Efficient hash algorithm distributes data blocks and limits the length of hash conflict chain under 10. Then the complexity of average query time can be set to O (10), and the complexity of binary tree algorithm is O (logn).

Accesses to the binary tree require the root lock. In high I/O concurrency scenarios, many lock conflicts are inevitable, resulting in low search efficiency. Accesses to the hash table only require the hash bucket lock. Lock conflicts only occur when there is hash conflict. Since hash buckets are sufficient and data blocks are distributed, the probability of concurrent I/O conflicts decreases remarkably, thereby reducing lock conflicts.

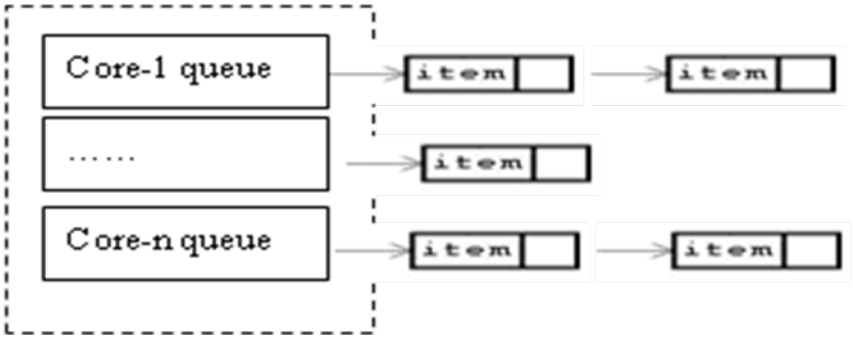
Figure 4-6 Principle of the cache search algorithm



4.2.2 Best Use of Cache Resources

The cache flushing algorithm is a multi-core FIFO queue algorithm. After data flushing is complete, clean data is directly deleted and not eliminated. The memory occupied by the read cache is saved, reduce memory overhead and improving cache space usage. The latency of direct disk read of Dorado V3 is low. The latency meets requirements in most scenarios. In addition, the Dorado memory is mostly used for metadata cache. Therefore, the Dorado disables the read cache function by default. The saved memory is used for metadata cache to improve the metadata hit ratio, reduce the read overhead of metadata, and improve the overall performance.

Figure 4-7 Cache FIFO organization



4.2.3 Cache Concurrency Processing Acceleration

Cache adopts the non-exclusive mechanism for data writing and flushing. The write I/Os of the front-end hosts are isolated from the back-end data flushing I/Os. This ensures that the front-end write I/Os have a stably low latency. The cache data flushing task balances loads among all CPU cores to prevent a single CPU core from overheating and fully utilize all CPU cores, achieving the optimal performance.

Based on the performance characteristics of the flash memory, the cache discards the red and black tree algorithms for I/O aggregation. The hash algorithm is selected to reduce the lock granularity, improve the front-end concurrency, and improve the overall performance.

The metadata cache efficiently caches hotspot metadata based on the hot and cold data, preventing performance from being affected in the case of large capacity and ensuring low latency of application I/Os.

4.3 Multi-Core Scheduling Optimization

In terms of pure computational overheads involving deduplication and compression as well as metadata organizations derived from ROW-based AFA, ROW represents not only a memory access intensive system, but also a compute-intensive system. Therefore, powerful Intel Xeon CPUs are used for AFA. More logical CPU cores, larger CPU cache, and two-socket CPUs with cross-CPU performance consumption bring a challenge to software design.

4.3.1 NUMA

For a common dual-channel NUMA system, the tested data indicates that the memory access bandwidth measured when NUMA is enabled is twice that measured after NUMA is disabled. For this reason, you must fully consider the affinity scheduling of CPUs and local memory in request distribution and execution, preventing intensive memory access requests from being sent to remote memory nodes. Therefore, Dorado uses a multi-core mechanism to improve performance.

4.3.2 Run-to-Complete Scheduling Principles

After host I/O requests are delivered to a storage array and distributed to different CPUs based on certain rules, the I/O requests are continuously executed on the CPUs by using the Run-To-Complete principles, avoiding invalid CPU cache caused by unnecessary switchovers.

4.3.3 Lock-Free Design

A traditional lock mechanism can be used to handle issues on exclusiveness and conflicts among concurrent operations and operation atomicity of high-concurrency and high-performance systems. However, using many locks may cause wide conflict range and high latency, imposing great impact on the high IOPS and low latency scenarios.

Lock-free design can ensure the atomicity of operations on a CPU core without thread switchover operations.

In addition, it can regularly disorder logics that need to be mutually exclusive are distributed to different CPUs by a specific algorithm so that the processing positions are mutually exclusive. To ensure the atomicity of operations, the lock-free design helps atomic operations not be switched on the core. In this way, logics that are mutually exclusive and provide atomicity protection are achieved without the need to apply a lock mechanism.

4.4 Stable Latency Guarantee

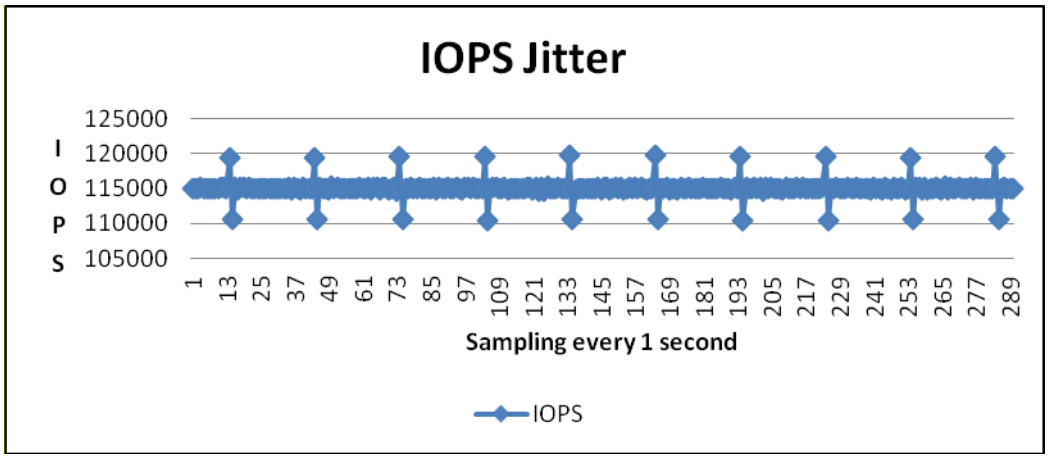
One of the basic issues for AFA is how to provide customers with excellent and stable low latency under heavy I/O load conditions.

Except for traditional and concurrent resources–based reservation and stream control policies, the system also requires mitigating the latency jitter caused by factors, such as interruptions, interferences from I/O priorities, intermixing of large and small I/Os, and metadata caching efficiency in the design, improving the latency stability. Additionally, in the design and implementation phases, reasonable division and utilization of system resources must be considered to avoid I/O latency ripples caused by fierce resource competition. The system also includes a large number of internal I/O requests except host I/O requests. To balance the effect of processing various requests and ensure that host I/Os meet 1 ms latency requirements, OceanStor Dorado V3 uses the following system-level optimization solutions:

4.4.1 Isolation of Interference Sources

The I/O processing process is similar to CPU pipelining. Once a pipeline is interrupted, it is costly to recover the pipeline. In addition, uncertain interruption duration causes a large I/O latency ripple, further resulting in host IOPS jitters. Figure 4-8 shows latency jitters of an IOPS sequence that has been interrupted by a system periodical timer.

Figure 4-8 Latency jitters along the timeline



Various interference sources that interrupt the system I/O processing process are classified as follows:

1. Asynchronous interference sources

   Generally, they may be asynchronous system events, such as hardware interruptions caused by peripherals, all types of timer tasks, and background tasks with high priorities. Once these interference sources are triggered, the I/O request service flows that are being executed on CPUs will be interrupted. In addition, because the running time of the interference sources is uncertain, the duration (from interruption to recovery) of the I/O request flows is uncontrollable. Therefore, asynchronous interference sources represent one type of source to cause I/O fluctuation.

2. Service interference sources

   If a multi-core scheduler cannot respond to I/O requests in a timely manner or a single I/O request is executed for a long period, host I/O fluctuation may occur.

   Under heavy load conditions, host I/O fluctuation may be intensified if data cache fails to be flushed in time and smoothly, and resources fails to be updated in a timely manner.

   If multi-core concurrency is not designed for metadata transaction handling, metadata transaction handling forms a single-point bottleneck source is easy to occur, aggravating I/O fluctuation with a high probability under heavy load conditions.

   For AFA with GC functions, the resource competition between host I/Os and GC-related operations is intensified because GC-related operations are not implemented on the host I/O path and data volume involving GC migration may be large and occupies resources required by host I/Os. On hosts, the competition is reflected by I/O fluctuation intensification and overall performance deterioration.

3. Computing interference sources

   Enterprise-class storage provides data with complete RAID protection. The calculation of typical RAID-5/6/TP consumes huge computing capabilities. For example, RAID 6 calculation consumes the computing capability of a CPU core if the single-core computing bandwidth of RAID 6 can be up to 7 GB/s and the host write bandwidth reaches 7 GB/s.

   In addition to basic RAID groups, AFS is provided with inline deduplication and compression functions. The hash algorithm that is well developed is used for deduplication. The compression algorithm not only maintains a high compression ratio, but also provides quick computing speed. Therefore, deduplication and compression calculations also consume a lot of computing resources.

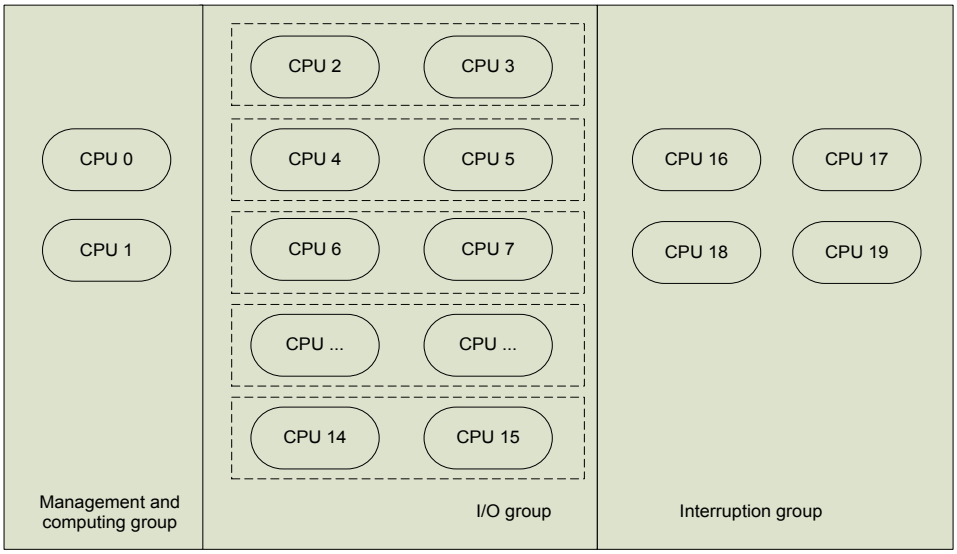
   The complete enterprise-class AFA also must be provided with a strict data protection mechanism, for example, industrial standard end-to-end DIF, to ensure user data security.

   If the impact of the preceding computing operations on host I/Os is not fully considered in the design and implementation phases, host I/O fluctuation may be intensified. For example, if a single compression takes a long period, host read requests fail to be responded in a timely manner.

   If the impacts on host I/Os are not fully considered in design and implementation phases, host I/O fluctuation will be greatly intensified. For example, if a single decompression task takes a long time, host read requests cannot be responded in a timely manner.

Interferences to host I/Os include but not limited to the preceding factors. To ensure the stable latency of host I/O responses, we need to isolate the preceding interference sources. For example, we can deploy these interference sources and primary host I/O paths in different CPU groups using the CPU grouping mode. Figure 4-9 shows a CPU grouping design.

Figure 4-9 CPU grouping



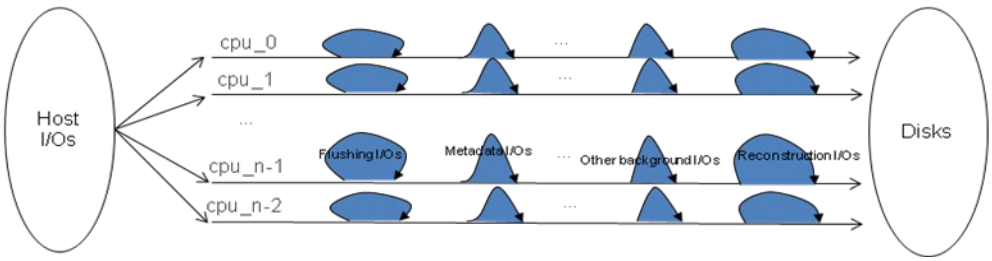
4.4.2 I/O Priority Guarantee Mechanism

System resources are classified into two types: hardware and software. Hardware resources cover CPU computing capabilities, CPU capacity, and front- and back-end ports. Software resources cover internal concurrency and quota resources.

Resource shortage occurs because the growth rate of system resources lags behind that of external performance demands. The system load increases. When a type of resources meets the bottleneck, resource competitions become fierce, resulting in host I/O fluctuation intensification and service capability degradation. In terms of system design and implementation, an end-to-end solution must be provided for I/Os to enjoy priority in obtaining resources. For AFA, CPUs are critical to systems.

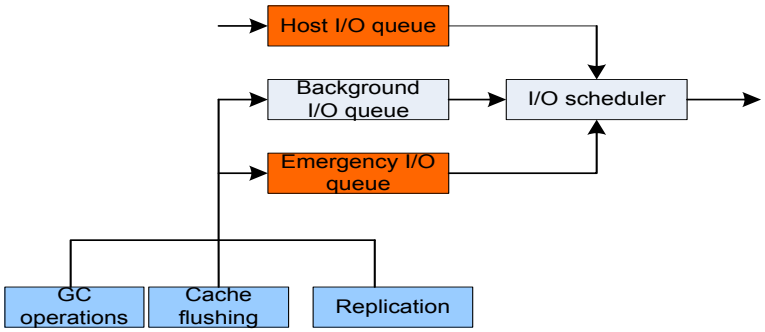
CPUs can be properly used by scheduling. When the computing capability bottleneck occurs, host read I/Os and host write I/Os on caches must be first scheduled, as shown in the following figure.

Figure 4-10 I/O working flows on CPUs



To ensure the latency of host I/O responses, a scheduler must be used to properly schedule the CPU usage (such as execution priority and duration for a single execution) of different types of I/Os. Based on the read/write process, only highest host read I/O priority can ensure that host read I/O latency is stable under mixed read/write service conditions.

Figure 4-11 I/O priority management of OceanStor Dorado V3

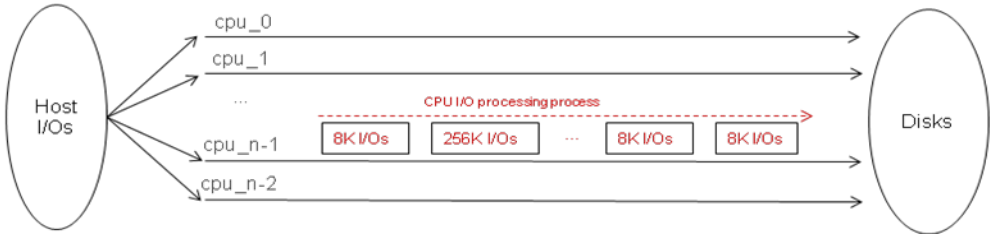


Except for CPUs, memory, ports, and channels also use the principle that high priorities are obtained to respond and execute requests, conforming to the SLA grade finally provided by users.

4.4.3 Latency Guarantee Mechanism for Intermixed Large and Small I/Os

Generally, large and small I/Os are intermixed. For example, if a system is equipped with OLTP and OLAP, arrays can receive typical I/Os with 8 KB size and large I/Os with over 128 KB size. An example is shown in the following figure.

Figure 4-12 I/O working flows on CPUs in a scenario where small and large I/Os are intermixed



The duration for processing large I/Os once is longer than that for processing small I/Os. For example, if we assume that the single-core compression bandwidth is 500 MB/s, 8 KB I/O compression takes approximately 16 μs and 256 KB I/O compression takes 1 ms. AFA requires that the average response latency of host I/Os is smaller than 1 ms. If a single I/O process takes a long period, responses to small host I/Os are interrupted and customers' key service processing capabilities will be adversely affected.

In addition, the page write latency of SSD MLC media ranges from 1.5 ms to 2 ms. If a large I/O is written once for a long period, SSD responses to read requests of other small I/Os are interrupted. For this reason, a host I/O latency ripple occurs.

To minimize the impact of intermixed large and small I/Os and ensure the processing efficiency of large I/Os, storage arrays divide large host I/O requests received into multiple I/Os of small granularity and send them to SSDs for processing.

4.4.4 Metadata Cache

All metadata needs to be cached on I/O paths to achieve a stable latency. The metadata in a pool consists of multiple Flash Translation Layer (FTL) tables, each of which represents a linear space. You can perform read caching for these FTL tables to achieve metadata caching. Classify each type of FTL metadata caches, and manage it using a global hash table to ensure that the percentage of scenarios where the number of conflicted nodes is larger than 5 is lower than 5% and improve search efficiency.

As the write data volume increases, an FTL cache with insufficient memory needs to be eliminated. Blocks that are stored at a layer with the maximum data volume eliminated and are not accessed for a longest period must be eliminated based on multi-level priority policies, ensuring that metadata is read at the lowest frequency without hits and eliminated at a lowest cost. In most scenarios where memory is insufficient, there is only once metadata read overhead, ensuring that metadata I/O latency is stable.

To reduce metadata volume, metadata can be compressed based on its characteristics. For example, in the scenario where snapshot is not provided, 25% of metadata memory overheads can be reduced using the prefix compression technology for metadata obtained from the space layer, expanding full cache capacity for users.

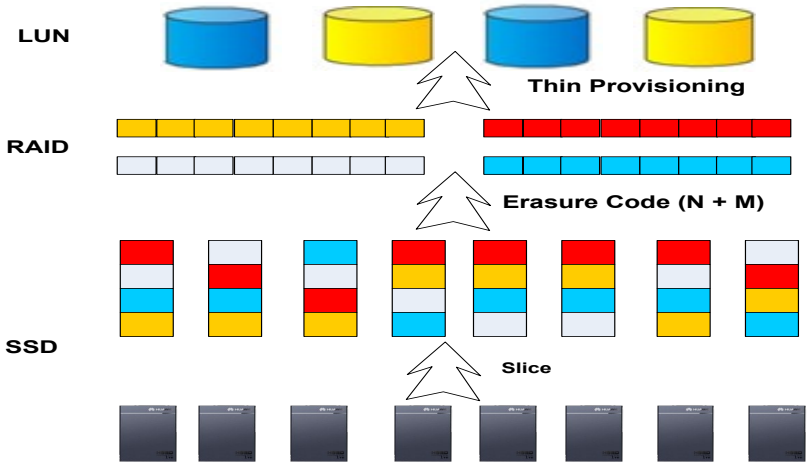
4.5 FlashLink

SSDs use flash chips and do not involve seeking overhead of mechanical disks. In addition, the wear leveling algorithm is used to implement all new write logic in the internal space. Therefore, the write-in-place solution is not applicable to SSDs.

FlashLink uses full-stripe write and background garbage collection solutions to manage data on disks. Full-stripe write: All data is written in new or redirected mode. This avoids read/write amplification caused by RAID, reduces access pressure on hard disks, and reduces read and write overheads. If the inventory data involved in the service data is changed, the data is set to invalid upon writing. The changed inventory data is collected and utilized at the background using the global garbage collection mode.

Based on FlashLink and SSD access features, resource allocation is optimized, reducing the write amplification factor of SSDs, improving disk performance, and prolonging service life of SSDs.

Figure 4-13 RAID 2.0+ For Flash



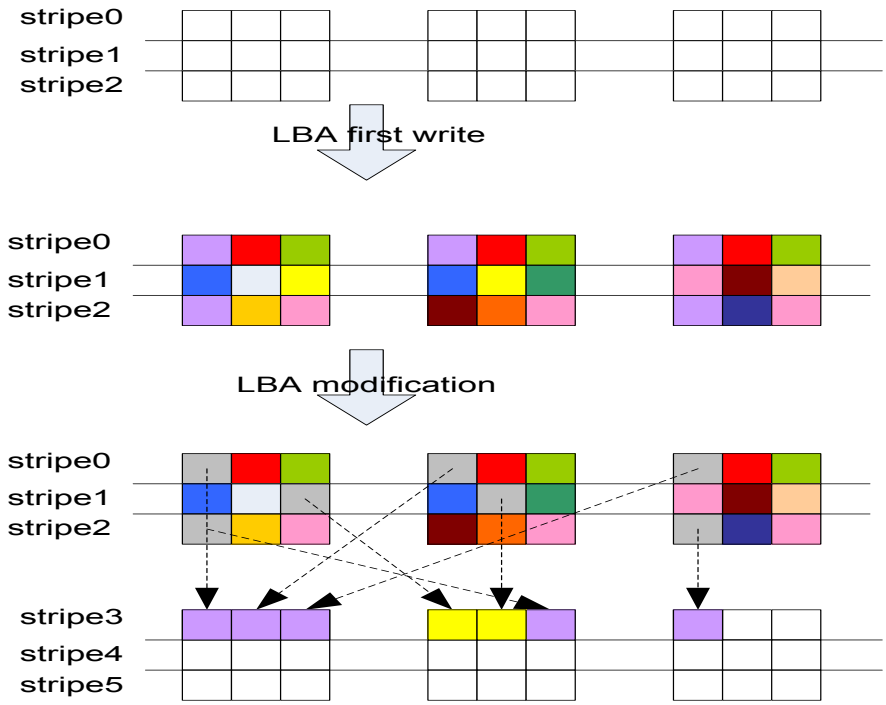
Multiple SSDs form a disk domain. In each disk domain, every SSD is divided into chunks (CKs) of a fixed size, generally 1 MB, for logical space management. An erasure code (EC) is selected based on customers' redundancy requirements. CKs from different SSDs form a chunk group (CKG) through the Huawei patented algorithm to provide highly reliable storage space. CKGs are then divided into grains with finer granularity as the space units of LUNs.

FlashLink has the following advantages:

- Uses private commands that work between SSDs and disk enclosures, full-stripe write, and global garbage collection to better support I/O read and write within SSDs, reducing write amplification, improving performance, and enhancing reliability.
- Supports dynamic RAID. Customers only need to specify RAID redundancy M and the system automatically selects a value for data N based on the usage of CKs, ensuring data reliability and optimal performance and space usage.

4.5.1 Full-Stripe Write

Figure 4-14 Full-stripe write



Full-stripe write: Data is written to new locations of back-end stripes.

First write to LBA: The system allocates the data to free space, namely idle stripes provided by the CKG. When allocating space, the CKG selects idle space according to the stripes.

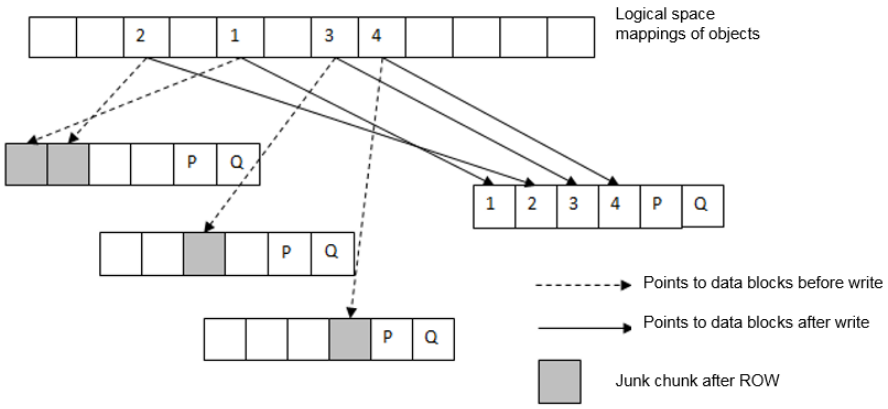
Modification to LBA: The system applies for new space for modified data and sets the legacy data as invalid/garbage data that waits to be reclaimed by the background. LUNs identify the modified data using the metadata table and CKGs check space usage by data validity status.

Full-stripe write effectively reduces write I/Os within an SSD, easing read and write conflicts and reducing I/O response latency.

Dorado uses the row-based full-stripe write design. Being compared with traditional RAID, the design has the following advantages:

- ROW-based full-stripe write

Figure 4-15 ROW-based full-stripe write



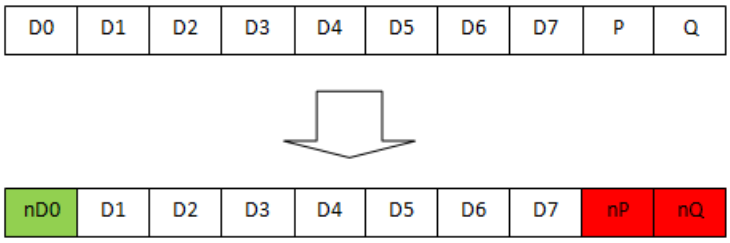
When RAID 6 is used to write data 1, 2, 3, and 4, data 1, 2, 3, and 4 are used to calculate P and Q. Then the data is written into disks. No extra prefetch is required.

For RAID-TP, only the P, Q, and R need to be calculated and written into disks.

In typical scenarios, the data is in 21D+2P and 20D+3P modes. The difference of data writes is about 5%. The performance is almost the same.

- Overwrite of traditional RAID

Figure 4-16 Overwrite of traditional RAID



When D0->nD0 is modified, D0, P, and Q, need to be pre-read. D0, nD0, P, and Q are used to calculate nP and nQ. Then, nD0, nP, and nQ are written into disks.

When some data of a RAID stripe changes (green in the preceding figure), you must update the corresponding parity bits (red blocks in the preceding figure). For an xD+yP RAID group:

When random small I/Os are written, the write amplification is as follows: y + 1 (2 for RAID 5, 3 for RAID 6, and 4 for RAID-TP).

Upon full-stripe write, the write amplification is as follows: (x + y)/x.

For a xD + yP RAID group, when the random small I/Os are written, the pre-read amplification is y + 1 (2 for RAID 5, 3 for RAID 6, and 4 for RAID-TP).

In the case of full-stripe write, because the pre-read does not exist, the pre-read amplification is 1.

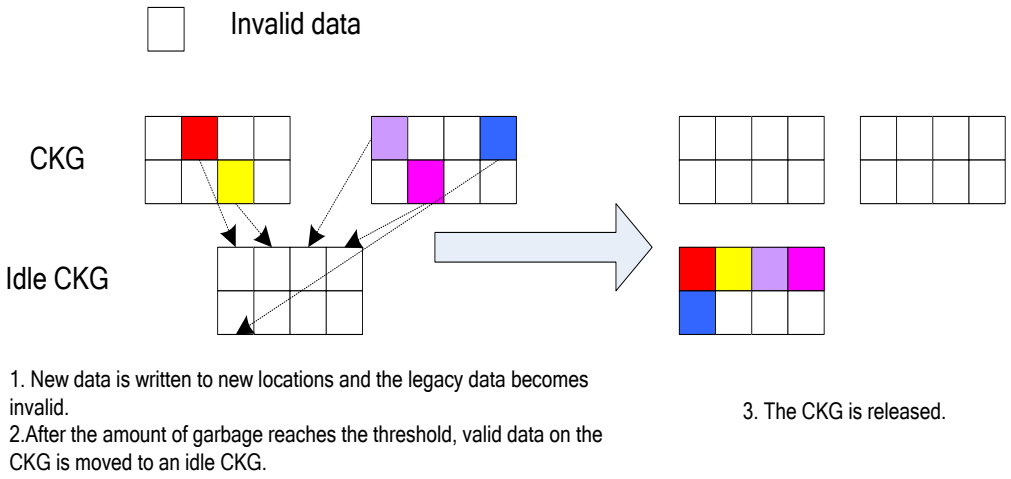
Table 4-1 Comparison of amplification values of different RAID groups

RAID	Write Amplification Generated by Random Small I/Os	Pre-Read Amplification Generated by Random Small I/Os	Write Amplification of Sequential I/Os
RAID 1	2	1	2
RAID 5 8D+1P	2	2	1.125
RAID 6 16D+2P	3	3	1.125
Dorado RAID 5 22D+1P	1.045	1	1.045
Dorado RAID 6 21D+2P	1.095	1	1.095
Dorado RAID-TP 20D+3P	1.15	1	1.15

In ROW mode, RAID write amplification is equivalent to sequential I/O write amplification and there is no pre-read amplification. The performance difference between different RAID levels is less than 5%. Therefore, for Dorado V3, you only need to select a RAID level based on the reliability requirements of application data without the need to consider the performance deterioration and write life problems caused by different RAID levels.

4.5.2 Garbage Collection

Figure 4-17 Garbage collection



Garbage collection is performed in the background. The system checks the proportion of garbage in a CKG and automatically starts garbage collection if the proportion meets the reclaiming conditions set in the system. The system migrates valid data in stripes to be recycled to idle stripes and marks the space of the migrated data invalid. When all space on a stripe becomes invalid, this stripe is released and can be used as new space.

Dorado V3 adopts the following technologies to minimize impact on host I/O performance exerted by GC, ensuring stable latency.

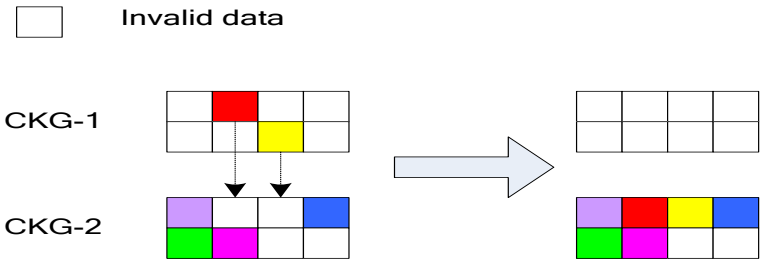
End-to-end I/O priority: ensures that I/Os generated by GC do not affect host I/Os.

On-demand GC start/stop: ensures that GC bandwidth matches with front-end write I/O bandwidth and migrates data as required, reducing write amplification and impact on host performance.

Optimized Cost-Benefit policy for selecting GC target data:

1. Stores cold and hot data separately. Metadata and data, and data blocks frequently and seldom used are stored separately.
2. Batch reclaiming based on when data is written. Data that was written to the system at the same point in time is reclaimed in the same batch, reducing migration of invalid data.
3. Based on the cost of releasing space, valid data in one CKG is moved to idle blocks of another CKG.

Figure 4-18 Garbage collection (moving data from one CKG to another)



GC ensures that the system can allocate new and large consecutive space in any conditions. I/Os are written to full stripes instead of different locations in SSDs, reducing processing complexity.

5 Conclusion

Design of traditional storage systems focuses on solving jitter and is suitable for sequential read and write characteristics of disks. SSDs have the following features:

1. Excellent sequential write and average random write performance: SDDs use a write method based on logs. All data is written to new flash units and a mapping relationship is established between LBAs and PBAs. SSDs also have an internal garbage collection mechanism that reclaims invalid LBA mapping units. However, garbage collection severely affects system performance by 40%. To solve this problem, Dorado V3 uses a disk and controller integration mechanism for full-stripe write and system garbage collection, optimizing SSD performance.
2. High random read performance: Unlike disks that feature random read, SSDs use flash as storage medium and allows direct read by hosts.
3. Unexpected latency: Dorado V3 keeps latency at 1 ms by grouping host I/Os and internal I/Os and optimizing system scheduling and hardware and software interruption.

Dorado V3 provides high-performance and low latency storage products and meets customers' requirements on performance, function, efficiency through software and hardware design and optimization, maximizing customer benefits.

6 Acronyms and Abbreviations

Table 6-1 Acronyms and abbreviations

Acronym and Abbreviation	Full Spelling
AFA	All-Flash Array
BBU	Backup Battery Unit
BTU	British Thermal Unit
CK	Chunk
DIF	Data Integrity Field
FC	Fibre Channel
GC	Garbage Collector
LUN	Logical Unit Number
NAS	Network Attached Storage
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OP	Over-Provisioning
RAID	Redundant Array of Independent Disks
ROW	Redirect on Write
SAN	Storage Area Network
SAS	Serial Attached SCSI
SCSI	Small Computer System Interface
S.M.A.R.T	Self-Monitoring, Analysis, and Reporting Technology
SSD	Solid-State Drive
SPOF	Single Point of Failure

Acronym and Abbreviation	Full Spelling
T10 PI	T10 Protection Information
VDI	Virtual Desktop Infrastructure
VSI	Virtual Server Infrastructure
WA	Write Amplification