AOS - Data IO Path

Nutanix AOS storage is the scale-out storage technology that appears to the hypervisor like any centralized storage array, however all of the I/Os are handled locally to provide the highest performance. More detail on how these nodes form a distributed system can be found in the next section.

Data Structure

Nutanix AOS storage is composed of the following high-level structure:

Storage Pool

- Key Role: Group of physical devices
- Description: A storage pool is a group of physical storage devices including PCIe SSD, SSD, and HDD devices for the cluster. The storage pool can span multiple Nutanix nodes and is expanded as the cluster scales. In most configurations, only a single storage pool is leveraged.

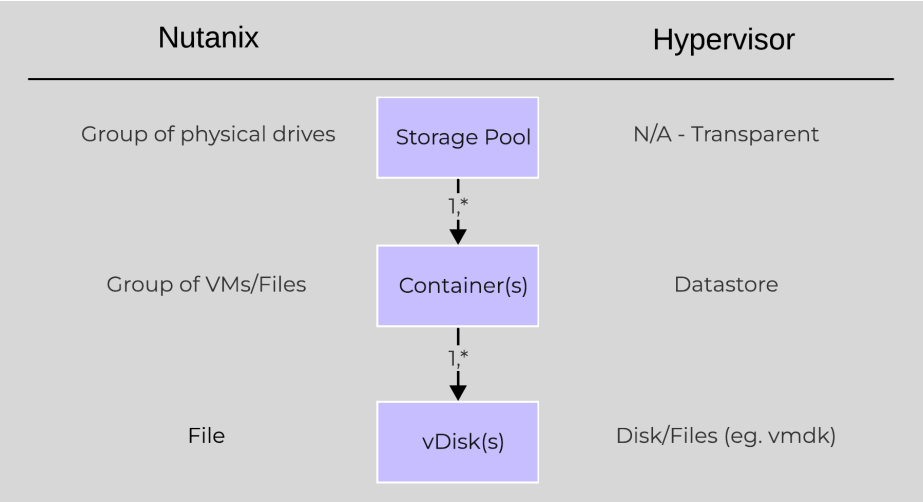
Container

- Key Role: Group of VMs/files
- Description: A container is a logical segmentation of the Storage Pool and contains a group of VM or files (vDisks). Some configuration options (e.g., RF) are configured at the container level, however are applied at the individual VM/file level. Containers typically have a 1 to 1 mapping with a datastore (in the case of NFS/SMB).

vDisk

- Key Role: vDisk
- Description: A vDisk is any file over 512KB on AOS including .vmdks and VM hard disks. vDisks are logically composed of vBlocks which make up the 'block map.'

The following figure shows how these map between AOS and the hypervisor:



High-level Filesystem Breakdown

vBlock

- Key Role: 1MB chunk of vDisk address space
- Description: A vBlock is a 1MB chunk of virtual address space composing a vDisk. For example, a vDisk of 100MB will have 100 x 1MB vBlocks, vBlock 0 would be for 0-1MB, vBlock 1 would be from 1-2MB, and so forth. These vBlocks map to extents which are stored as files on disk as extent groups.

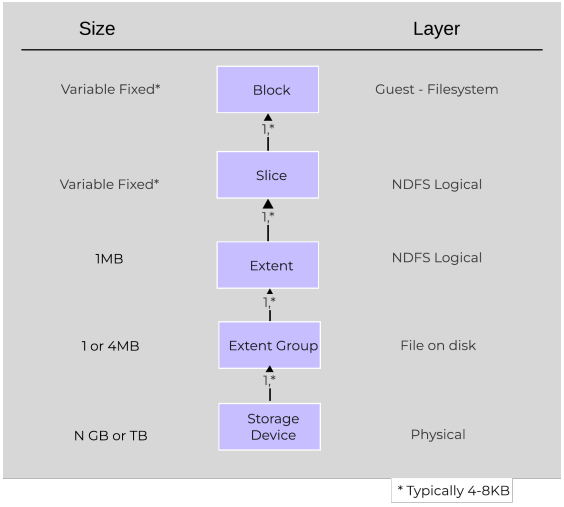
Extent

- Key Role: Logically contiguous data
- Description: An extent is a 1MB piece of logically contiguous data which consists of n number of contiguous blocks (varies depending on guest OS block size). Extents are written/read/modified on a sub-extent basis (aka slice) for granularity and efficiency. An extent's slice may be trimmed when moving into the cache depending on the amount of data being read/cached.

Extent Group

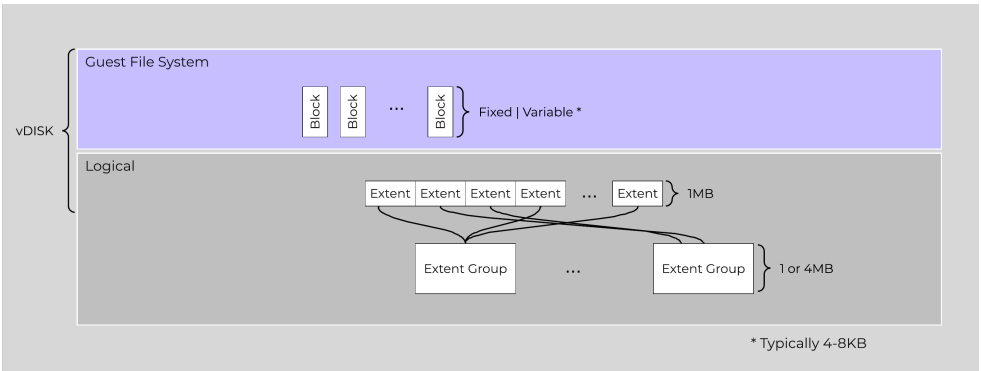
- Key Role: Physically contiguous stored data
- Description: An extent group is a 1MB or 4MB piece of physically contiguous stored data. This data is stored as a file on the storage device owned by the CVM. Extents are dynamically distributed among extent groups to provide data striping across nodes/disks to improve performance. NOTE: Extent groups can now be either 1MB or 4MB depending on deduplication.

The following figure shows how these structs relate between the various file systems:



Low-level Filesystem Breakdown

Here is another graphical representation of how these units are related:



Graphical Filesystem Breakdown

I/O Path and Cache

For a visual explanation, you can watch the following video: LINK

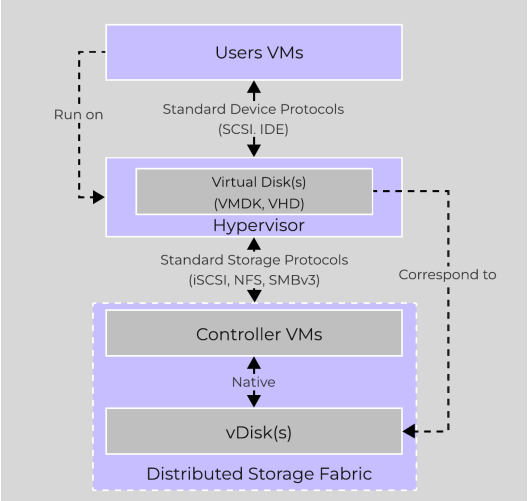
The typical hyperconverged storage I/O path can be characterized into the following core layers:

1. Guest OS (UVM) to virtual disk(s)
   - This remains unchanged with Nutanix. Depending on the hypervisor the guest OS will use a device driver to talk to a virtual disk device. Depending on the hypervisor this could be virtio-scsi (AHV), pv-scsi (ESXi), etc. The virtual disks will also vary based upon the hypervisor (e.g. vmdk, vhd, etc.)

2. Hypervisor to AOS (via CVM)
   - Communication between the hypervisor and Nutanix occurs via standard storage protocols (e.g. iSCSI, NFS, SMBv3) over the local interface of the CVM and hypervisor. At this point all communication has been local to the host (there are scenarios where I/O will be remote (e.g. local CVM down, etc.).

3. Nutanix I/O path
   - This is all transparent to the hypervisor and UVMs and it native to the Nutanix platform.

The following image shows a high-level overview of these layers:

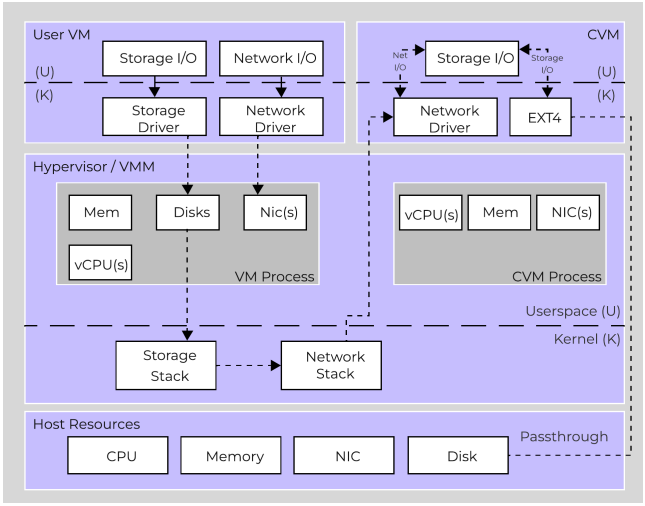


High-level I/O Path - Traditional

Communication and I/O

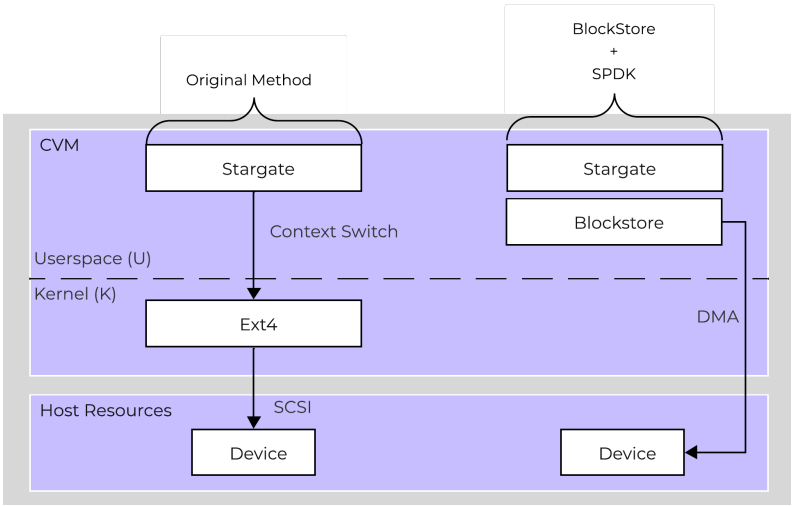
Within the CVM the Stargate process is responsible for handling all storage I/O requests and interaction with other CVMs / physical devices. Storage device controllers are passed through directly to the CVM so all storage I/O bypasses the hypervisor.

The following image shows a high-level overview of the traditional I/O path:



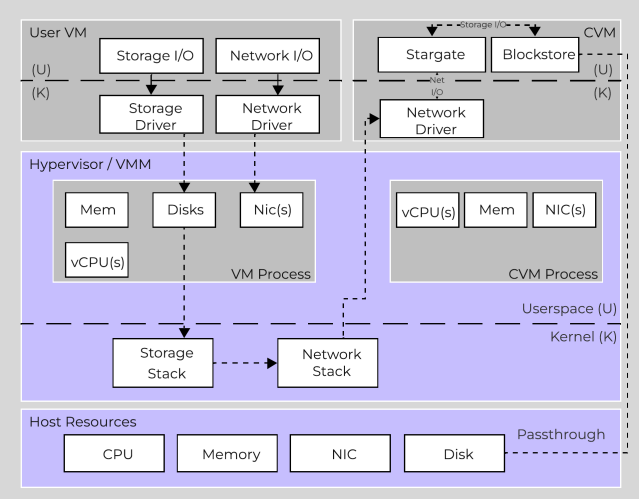
High-level I/O Path

Nutanix BlockStore is an AOS capability which creates an extensible filesystem and block management layer all handled in user space. This eliminates the filesystem from the devices and removes the invoking of any filesystem kernel driver. The introduction of newer storage media (e.g. NVMe), devices now come with user space libraries to handle device I/O directly (e.g. SPDK) eliminating the need to make any system calls (context switches). With the combination of BlockStore + SPDK all Stargate device interaction has moved into user space eliminating any context switching or kernel driver invocation.



Stargate - Device I/O Path

The following image shows a high-level overview of the updated I/O path with BlockStore + SPDK:

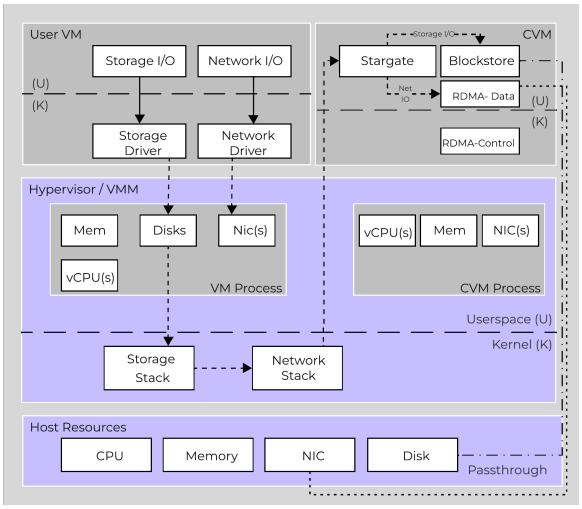


High-level I/O Path - BlockStore

To perform data replication the CVMs communicate over the network. With the default stack this will invoke kernel level drivers to do so.

However, with RDMA these NICs are passed through to the CVM, bypassing the hypervisor and reducing interrupts. Also, within the CVM all network traffic using RDMA only uses a kernel level driver for the control path, then all actual data I/O is done in user-space without any context switches.

The following image shows a high-level overview of the I/O path with RDMA:



High-level I/O Path - RDMA

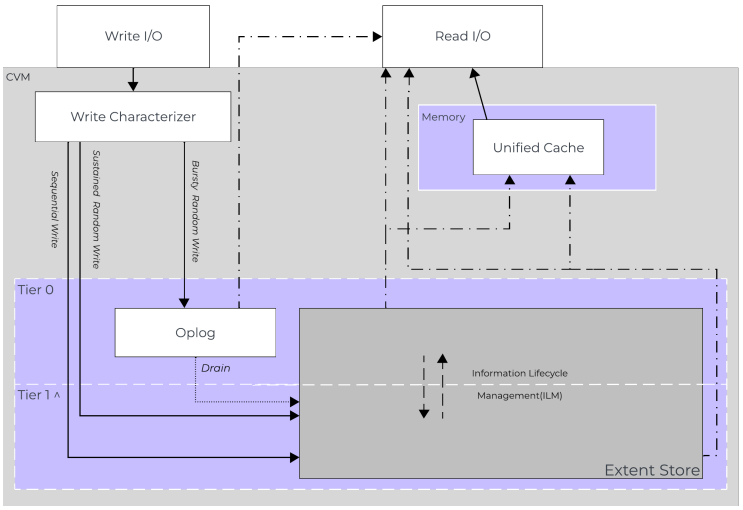
To summarize, the following enhancements optimize with the following:

1. PCI passthrough bypasses the hypervisor for device I/O
2. SPDK + Blockstore eliminates kernel storage driver interactions and moves them to user-space
3. RDMA bypasses the hypervisor and all data transfer is done in CVM user-space N

Stargate I/O Logic

Within the CVM the Stargate process is responsible for handling all I/O coming from user VMs (UVMs) and persistence (RF, etc.). When a write request comes to Stargate, there is a write characterizer which will determine if the write gets persisted to the OpLog for bursty random writes, or to Extent Store for sustained random and sequential writes. Read requests are satisfied from Oplog or Extent Store depending on where the data is residing when it is requested.

The Nutanix I/O path is composed of the following high-level components:



AOS I/O Path

^:

1. In all-flash node configurations (All NVMe SSD, All SATA/SAS SSD, NVMe+SATA/SAS SSD) the Extent Store will only consist of SSD devices and no tier ILM will occur as only a single flash tier exists.
2. In cases where Optane is used (e.g. Intel Optane + NVMe/SATA SSD) the highest performance media will be Tier 0 and the lower performance media will be Tier 1.
3. For hybrid scenarios with flash and HDD, the flash would be Tier 0 with HDD being Tier 1.
4. OpLog is always on SSDs in Hybrid Clusters and on both Optane and NVMe SSDs in Optane+NVMe clusters.
5. Data is moved between tiers by Intelligent Lifecycle Management (ILM) based on access patterns.

OpLog

- Key Role: Persistent write buffer
- Description: The OpLog is similar to a filesystem journal and is built as a staging area to handle bursts of random writes, coalesce them, and then sequentially drain the data to the extent store. Upon a write, the OpLog is synchronously replicated to another n number of CVM's OpLog before the write is acknowledged for data availability purposes. All CVM OpLogs partake in the replication and are dynamically chosen based upon load. The OpLog is stored on the SSD tier on the CVM to provide extremely fast write I/O performance, especially for random I/O workloads. All SSD devices participate and handle a portion of OpLog storage. For sequential workloads, the OpLog is bypassed and the writes go directly to the extent store. If data is currently sitting in the OpLog and has not been drained, all read requests will be directly fulfilled from the OpLog until they have been drained, where they would then be served by the extent store/unified cache. For containers where fingerprinting (aka Dedupe) has been enabled, all write I/Os will be fingerprinted using a hashing scheme allowing them to be deduplicated based upon fingerprint in the unified cache.

Extent Store

- Key Role: Persistent data storage
- Description: The Extent Store is the persistent bulk storage of AOS and spans all device tiers (NVME, PCIe SSD, SATA SSD, HDD) and is extensible to facilitate additional devices/tiers. Data entering the extent store is either being A) drained from the OpLog or B) is sequential/sustained in nature and has bypassed the OpLog directly. Nutanix ILM will determine tier placement dynamically based upon I/O patterns, number of accesses of data and weight given to individual tiers and will move data between tiers.

Sequential Write Characterization

Write IO is deemed as sequential when there is more than 1.5MB of outstanding write IO to a vDisk. IOs meeting this will bypass the OpLog and go directly to the Extent Store since they are already large chunks of aligned data and won't benefit from coalescing.

This is controlled by the following Gflag: vdisk_distributed_oplog_skip_min_outstanding_write_bytes.

All other IOs, including those which can be large (e.g. >64K) will still be handled by the OpLog.

Autonomous Extent Store (AES)

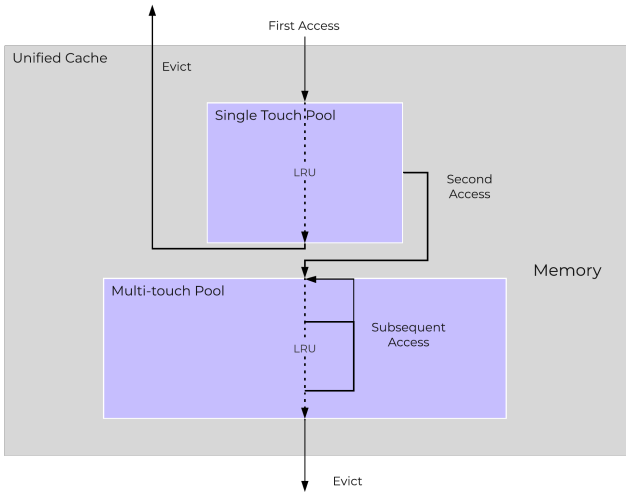
- Key Role: Persistent data storage
- Description: The Autonomous Extent Store (AES) is a method for writing / storing data in the Extent Store. It leverages a mix of primarily local + global metadata (more detail in the 'Scalable Metadata' section following) allowing for much more efficient sustained performance due to metadata locality. For sustained random write workloads, these will bypass the OpLog and be written directly to the Extent Store using AES. For bursty random workloads these will take the typical OpLog I/O path then drain to the Extent Store using AES where possible.

  In AOS 6.8, an enhancement to AES was introduced called AES Optimized Metadata to improve performance and maximize resource utlization for all-flash and NVMe clusters. It was built to reduce CPU usage and drive the performance for sustained random writes going to the Extent Store. With AES Optimized Metadata, writes are batched into fewer transaction updates. This helps in reducing the number of disk operations needed to persist sustained random writes to the extent store. This translates to more optimized CPU usage through fewer cycles consumed and drives higher sustained random write performance on all-flash and NVME clusters.

Unified Cache

- Key Role: Dynamic read cache
- Description: The Unified Cache is a read cache which is used for data, metadata and deduplication and stored in the CVM's memory. Upon a read request of data not in the cache (or based upon a particular fingerprint), the data will be read from the extent store and will also be placed into the single-touch pool of the Unified Cache which completely sits in memory, where it will use LRU (least recently used) until it is evicted from the cache. Any subsequent read request will “move” (no data is actually moved, just cache metadata) the data into the multi-touch pool. Any read request for data in the multi-touch pool will cause the data to go to the peak of the multi-touch pool where it will be given a new LRU counter. Cache size can be calculated using the following formula: ((CVM Memory - 12 GB) * 0.45). For example a 32GB CVM would have the following cache size: ((32 - 12)*0.45) == 9GB.

The following figure shows a high-level overview of the Unified Cache:



AOS Unified Cache

Cache Granularity and Logic

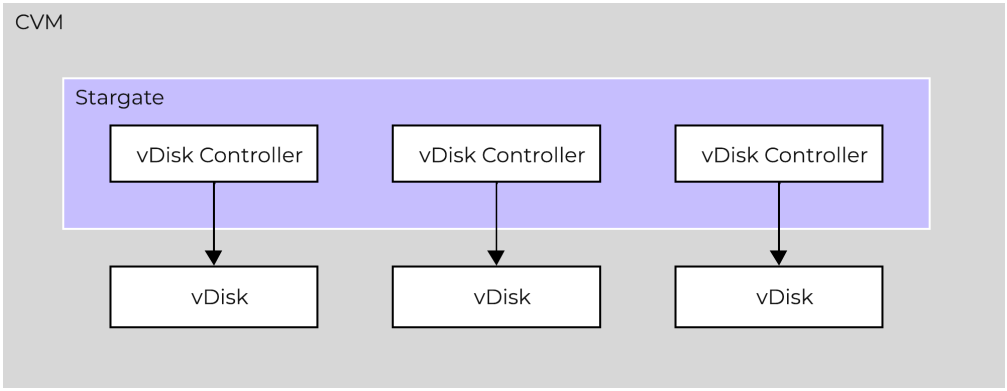
Data is brought into the cache at a 4K granularity and all caching is done real-time (e.g. no delay or batch process data to pull data into the cache).

Each CVM has its own local cache that it manages for the vDisk(s) it is hosting (e.g. VM(s) running on the same node). When a vDisk is cloned (e.g. new clones, snapshots, etc.) each new vDisk has its own block map and the original vDisk is marked as immutable. This allows us to ensure that each CVM can have it's own cached copy of the base vDisk with cache coherency.

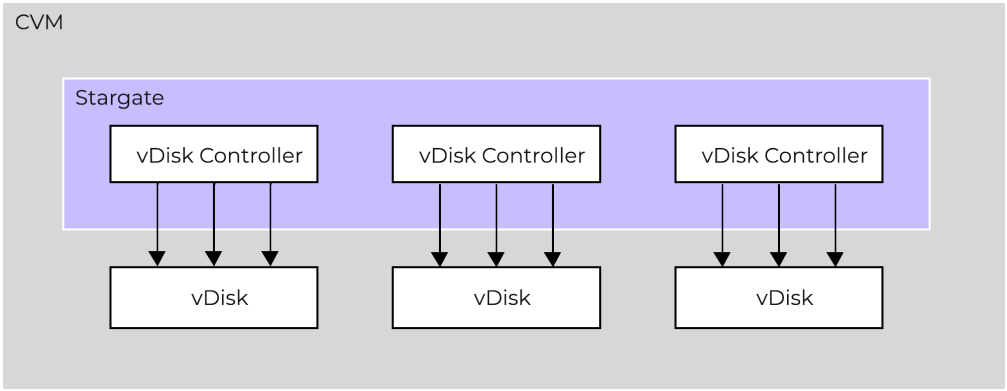
In the event of an overwrite, that will be re-directed to a new extent in the VM's own block map. This ensures that there will not be any cache corruption.

Single vDisk Sharding

AOS was designed and architected to deliver performance for applications at scale. Inside Stargate, I/O is processed by threads for every vdisk created by something called vdisk controller. Every vdisk gets its own vdisk controller inside Stargate responsible for I/O for that vdisk. The expectation was that workloads and applications would have multiple vdisks each having its own vdisk controller thread capable of driving high performance the system is capable of delivering.



This architecture works well except in cases of traditional applications and workloads that had VMs with single large vdisk. These VMs were not able to leverage the capabilities of AOS to its fullest. So we enhanced our architecture such that the vdisk controller requests to a single vdisk are now distributed to multiple vdisk controllers. This is accomplished by creating shards of the controller, each having its own thread. I/O distribution to multiple controllers is done by a primary controller so for external interaction this still looks like a single vdisk. This results in effectively sharding the single vdisk making it multi-threaded. This enhancement alongwith other technologies talked above like Blockstore, AES allows AOS to deliver consistent high performance at scale even for traditional applications that use a single vdisk.



Scalable Metadata

For a visual explanation, you can watch the following YouTube video: Tech TopX by Nutanix University: Scalable Metadata

Metadata is at the core of any intelligent system and is even more critical for any filesystem or storage array. For those unsure about the term 'metadata'; essentially metadata is 'data about data'. In terms of AOS, there are a few key principles that are critical for its success:

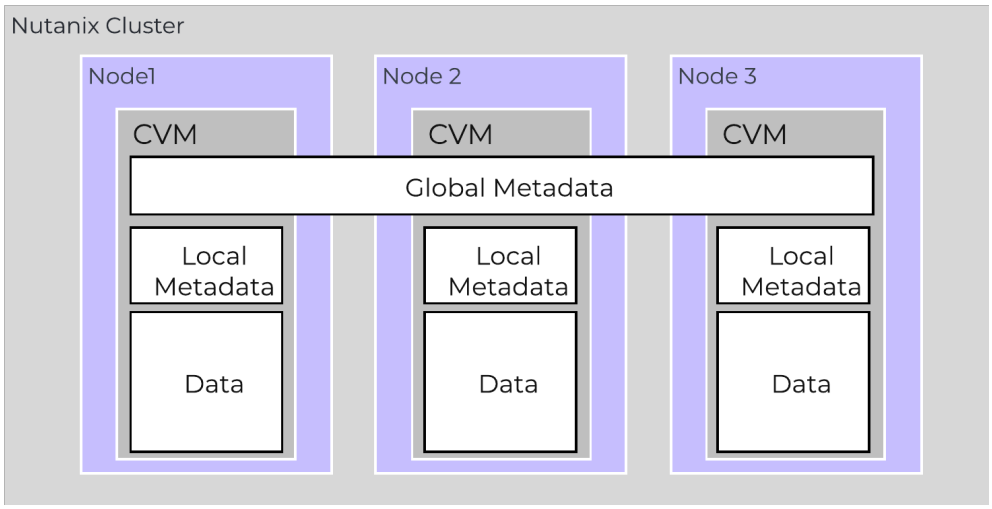
- Must be right 100% of the time (known as "strictly consistent")
- Must be ACID compliant
- Must have unlimited scalability
- Must not have any bottlenecks at any scale (must be linearly scalable)

As of AOS 5.10 metadata is broken into two areas: global vs. local metadata (prior all metadata was global). The motivation for this is to optimize for "metadata locality" and limit the network traffic on the system for metadata lookups.

The basis for this change is that not all data needs to be global. For example, every CVM doesn't need to know which physical disk a particular extent sits on, they just need to know which node holds that data, and only that node needs to know which disk has the data.

By doing this we can limit the amount of metadata stored by the system (eliminate metadata RF for local only data), and optimize for "metadata locality."

The following image shows the differentiation between global vs. local metadata:



Global vs. Local Metadata

Local Metadata

- Description:
  - Local metadata store per CVM containing information only needed by the local node. This is leveraged by the Autonomous Extent Store (AES) introduced in 5.10.

- Storage Mechanism:
  - AES DB (based on Rocksdb)
  - AES Optimized Metadata (based on B-Trees)

- Types of data stored:
  - Physical extent / extent group placement (e.g. egroup to disk mappings), etc.

Global Metadata

- Description:
  - Metadata that is globally available to any CVM and sharded across CVMs in the cluster. All metadata prior to 5.10.

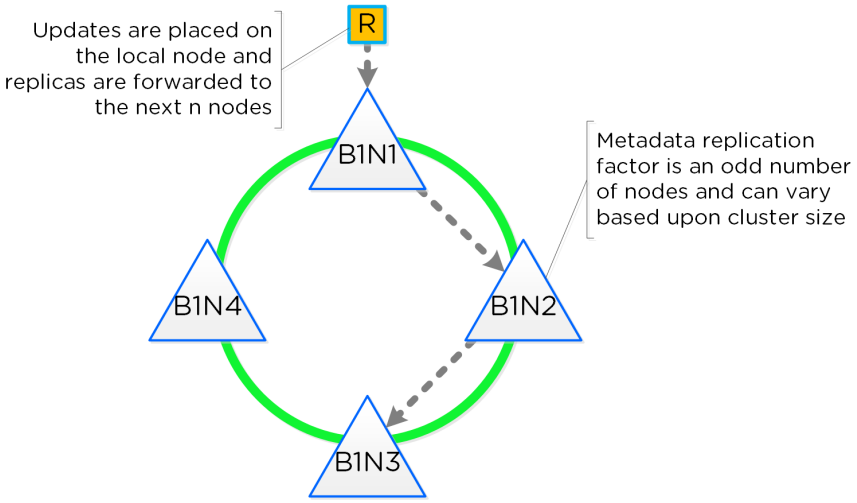
- Storage Mechanism:
  - Medusa Store (based on Cassandra)

- Types of data stored:
  - vDisk block maps, extent to node mappings, time series stats, configurations, etc.

The section below covers how global metadata is managed:

As mentioned in the architecture section above, AOS utilizes a “ring-like” structure as a key-value store which stores essential global metadata as well as other platform data (e.g., stats, etc.). In order to ensure global metadata availability and redundancy a replication factor (RF) is utilized among an odd amount of nodes (e.g., 3, 5, etc.). Upon a global metadata write or update, the row is written to a node in the ring that owns that key and then replicated to n number of peers (where n is dependent on cluster size). A majority of nodes must agree before anything is committed, which is enforced using the Paxos algorithm. This ensures strict consistency for all data and global metadata stored as part of the platform.

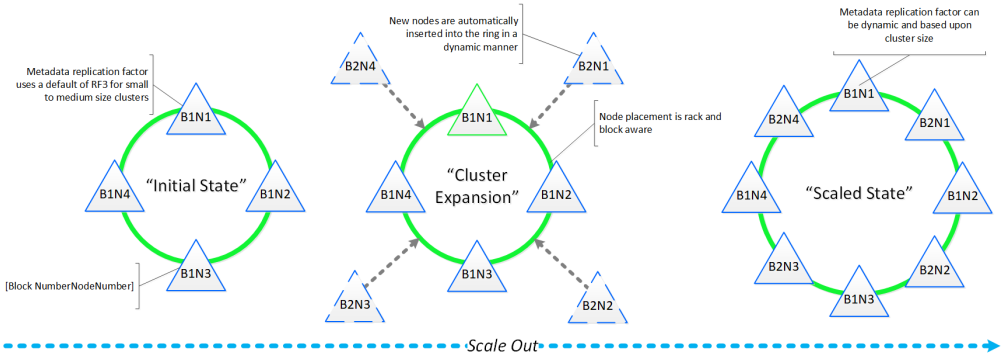
The following figure shows an example of a global metadata insert/update for a 4 node cluster:



Cassandra Ring Structure

Performance at scale is also another important struct for AOS global metadata. Contrary to traditional dual-controller or "leader/worker" models, each Nutanix node is responsible for a subset of the overall platform's metadata. This eliminates the traditional bottlenecks by allowing global metadata to be served and manipulated by all nodes in the cluster. A consistent hashing scheme is utilized for key partitioning to minimize the redistribution of keys during cluster size modifications (also known as "add/remove node"). When the cluster scales (e.g., from 4 to 8 nodes), the nodes are inserted throughout the ring between nodes for "block awareness" and reliability.

The following figure shows an example of the global metadata "ring" and how it scales:



Cassandra Scale Out