



UNIVERSAL STORAGE EXPLAINED



Last Updated:
December 14, 2022

TABLE OF CONTENTS

INTRODUCTION	3
Why Universal Storage	4
Defining Universal Storage	5
New Technologies Lay A New Storage Foundation	5
Scale-out Beyond Shared Nothing	5
Hyperscale flash	5
Stateless Containers	5
Storage Class Memory	5
NVMe over Fabrics	5
HOW IT WORKS	17
The Element Store: the VAST Universal Storage Namespace	17
Inherently Persisten	18
Transactionally Consisten	19
A Thin-Provisioned, Byte-Granular Data Store	22
Inherently Scalable Namespace	22
A Breakthrough Approach to Commodity flash Management	23
Challenges With Commodity flash	23
Hyperscale flash Page Size and Erase Block Challenges	24
Universal Storage Data Structures: Optimized for Hyperscale flash	26
Endurance is Amortized: Wear-Leveling Across Many-Petabyte Storage	30
VAST Foresight: Retention-Aware Data Protection Stripes	30
Hyperscale flash and 10-Year Endurance	33
A Breakthrough Approach to Data Protection	34
Breaking the Resilience vs. Cost Tradeoff	34
VAST Locally-Decodable Error Correction Codes	34
VAST Checksums	38
Low-Overhead Snapshots	38
Snap to Object	39
Native Replication	40
A Breakthrough Approach to Data Reduction	42
Beyond Deduplication and Compression	42
Similarity Reduction to the Rescue	43
VASTOS: the VAST Operating System	46
A Realtime Operating System in Userspace	46
VASTOS: Multi-Protocol Support	46
VAST NFS: Parallel File System Speed, NAS Simplicity	47
VAST SMB: File services for Windows and Macintosh	52
VAST S3: Object Storage for Modern Applications	55
Batch Delete: The .Trash folder	55
Platform Integration Drivers	56
Encryption at Rest	57
VASTOS Cluster Management	58
Longevity	65
CONCLUSIONS	66
DASE: The Universal Storage Architecture	66
Hardware	67
DF-5615 NVMe Enclosure	67
VAST Quad Server Chassis	68

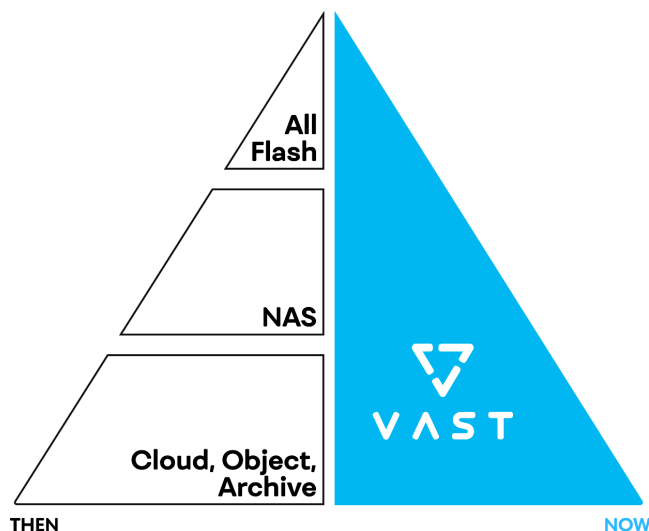
INTRODUCTION

VAST Data's Universal Storage redefines the economics of flash storage, making flash affordable for all applications, from the highest performance databases to the largest data archives, for the first time. The Universal Storage concept blends game-changing storage innovations to lower the acquisition cost of flash with an exabyte-scale file and object storage architecture breaking decades of storage tradeoffs.

With the advantage of new, enabling technologies that weren't available before 2018, this new Universal Storage concept can achieve a previously-impossible architecture design point. The system combines low-cost hyperscale flash drives and Storage Class Memory with stateless, containerized storage services all connected over new low-latency NVMe over Fabrics networks to create VAST's Disaggregated Shared Everything (DASE) scale-out architecture. Next-generation global algorithms are applied to this DASE architecture to deliver new levels of storage efficiency, resilience, and scale.

While the architecture concepts are sophisticated, the intent and vision of Universal Storage are simple: to bring an end to the data center HDD era and end the complexity of storage tiering that is a byproduct of the decades of compromises caused by mechanical media. This White Paper will introduce you to VAST Data's Universal Storage, the DASE architecture, and explain how this new architecture defies all conventional definitions of storage. In breaking the classic price/performance tradeoff, this system features all-flash performance at archive economics to simplify the data center and accelerate all modern applications.

Why Universal Storage?



VAST's Universal Storage challenges the fundamental premises that underlie the storage tiering paradigm

storage technologies.

While the savings are clear when applying this model with legacy storage architectures, the idea that data should exist on a specific storage tier according to its current value creates multiple challenges:

The Tyranny of Tiers

Over 30 years ago, Gartner introduced the storage tiering model as a means to optimize data center costs by advising customers to deprecate older and less-valuable data to lower-cost (and slower) tiers of storage. Fast forward 30 years and the sprawl of storage technologies within organizations has grown to unmanageable proportions – where many of the world's largest companies can be found managing dozens of different types of storage. This problem is exhibited when defining both storage class (for example: all-flash, hybrid, all-HDD, tape) as well as by classes of protocols (block, file, object, big data, etc.).... all of it creates a complex pyramid of

- Having multiple tiers of storage forces storage users to create workflows to move and/or copy data back and forth between primary and secondary storage. Users can spend extraordinary amounts of time balancing their applications across a myriad of storage silos while application wait time can be significant as data travels between archival and production storage.
- Even with automation frameworks and data management systems that can simplify the challenge of “where’s my data?”, all this data movement creates integration complication and reduces overall infrastructure efficiency by creating data copies, inefficiently using silos of variably-utilized infrastructure and wasting infrastructure resources that could otherwise be used for user I/O on data movement.

The Demands of Artificial Intelligence Render Storage Tiering Obsolete

Arguably the greater problem with storage tiering is that this concept assumes that the applications accessing data enjoy a narrow and predefined view of their data access requirements. While that’s true for some applications, such as traditional database engines, new game-changing AI and analytics tools, such as machine learning and deep learning, see value in all data and want the fastest access to the largest amounts of data. For example, when a deep learning system trains it’s neural network model for facial recognition, the model becomes more accurate only once it’s run against all the photos in the dataset, not just the 15–30% that may fit in some expensive flash tier. The value these applications bring is proportionate to the corpus of data they get exposed to, where they thrive with large data sets.

Defining Universal Storage

Universal Storage is a next-generation, scale-out file and object storage concept that breaks decades of storage tradeoffs, and in so doing defies classical storage definitions. Universal Storage is:

- Fast enough to meet the needs of the most demanding legacy and modern applications
- Scales to exabytes and scales to service the needs of many applications simultaneously
- Affordable enough to eliminate the economic benefit of storage tiering or archiving
- Accessible over standard, file, and object protocols that provide easy data access for all applications

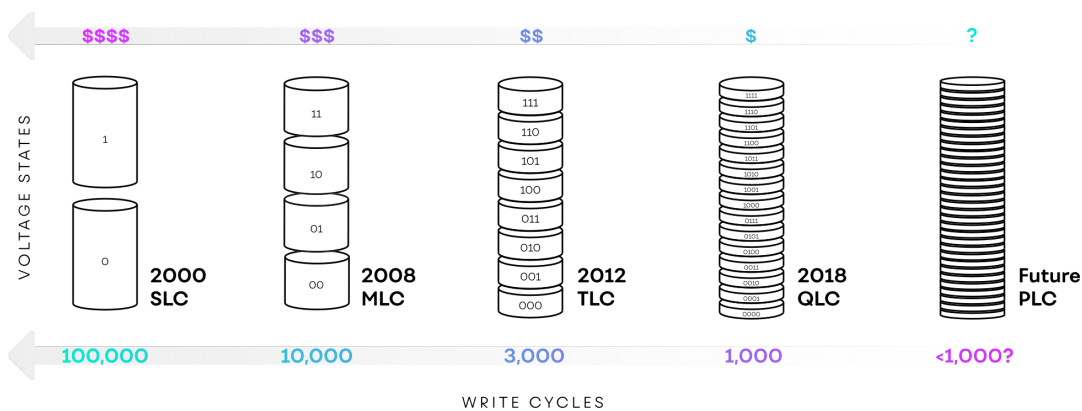
New Technologies Lay A New Storage Foundation

There are points in time where the introduction of new technologies make it possible to rethink fundamental approaches to system architecture. In order to realize the Universal Storage architecture vision, VAST made a bet on a trio of underlying technologies that were not available to previous storage architecture efforts, and in fact, only all became commercially viable in 2018. These are:

Hyperscale flash

Hyperscale flash SSDs make flash affordable by constantly increasing flash density, and delivering that capacity directly to storage applications without the overprovisioning, dual-port controllers, DRAM caches and power protection circuitry of enterprise SSDs

Squeezing another bit into each flash cell boosts capacity, and since it doesn't significantly increase manufacturing costs brings the cost per GB of flash down to unprecedentedly low levels. Unfortunately, squeezing more bits in each cell comes with a cost. As each successive generation of flash chips reduced cost by fitting more bits in a cell, each generation also had lower endurance, wearing out after fewer write/erase cycles. The differences in endurance across flash generations are huge – while the first generation of NAND (SLC) could be overwritten 100,000 times, QLC endurance is 100x lower. As flash vendors promote their upcoming PLC (Penta Level Cell) flash that holds 5 bits per cell endurance is projected to be even lower.



Erasing flash memory requires high voltage that causes damage to the flash cell's insulating layer at a physical level. After multiple cycles, enough damage has accumulated to allow some electron leakage through the silicon's insulating layer. This insulating layer wear is the cause of high bit density flash's lower endurance. For a QLC cell, for example, to hold a four-bit value; it must hold one of 16 discrete charge/voltage levels all between 0 and 3 volts or so. Holding that many bits as slightly different voltage levels makes QLC more sensitive to electron leakage through the insulating layers. As the number of bits that have to be represented by a voltage between 0 and 3 volts increases the difference between one value and another shrinks making each generation of flash more sensitive to the escape of a few electrons from a cell. This allows low bit density flash to absorb more damaging erase cycles before leakage changes a 1 to a 0 or vice versa.

VAST's Universal Storage systems were designed to minimize flash wear by both using innovative new data structures that align with the internal geometry of low-cost hyperscale SSDs in ways never before attempted and a large Storage Class Memory write buffer to absorb writes, providing the time, and space, to minimize flash wear. The combination allows VAST Data to support our flash systems for 10 years, which has its own impact on system ownership economics.

Stateless Containers

The logic of VAST's Universal Storage cluster runs in stateless containers. Thanks to NVMe-oF and NVMe flash

and Storage Class Memory, each container enjoys direct-attached levels of storage performance without having any direct-attached stateful storage. Containers make it simple to deploy and scale VAST as a software-defined microservice while also laying the foundation for a much more resilient architecture where container failures are non-disruptive to system operation.

Storage Class Memory

For the first time in 30 years, a new type of media has been introduced into the classic media hierarchy. Storage Class Memory encompasses several new persistent memory technologies that are persistent and deliver significantly lower latency than the NAND flash memory used in most SSDs.

Universal Storage systems use Storage Class Memory both as a high-performance write buffer to enable the deployment of low-cost hyperscale flash for the system's data store, as well as a global metadata store. Storage Class Memory was selected for its low write latency and long endurance. A Universal Storage cluster includes tens to hundreds of terabytes of Storage Class Memory capacity, which provides the VAST DASE architecture with several architectural benefits:

- **Optimizes Write Latency:** VAST servers acknowledge writes to their clients after mirroring data to a buffer of ultra-low latency NVMe Storage Class Memory SSDs. This buffer isolates application write latency from the time required to perform data services like global flash translation and data reduction while protecting applications from the high write latency of hyperscale SSDs.
- **Protects Low-Endurance flash from Transient Writes:** Data can live in the SCM write buffer indefinitely, and because the buffer is so comparatively large it serves the purpose of relieving the hyperscale SSDs from the wear of any intermediate updates.
- **Data Protection Efficiency:** the Storage Class Memory write buffer provides the capacity to assemble many wide, deep stripes concurrently and write them in a near-perfect form to hyperscale SSDs in order to get 20x more longevity from these low-cost SSDs than classic enterprise storage write patterns can achieve.
- **Protects Low-Endurance flash from Aggressive Data Reduction:** Storage Class Memory enables the VAST Cluster to perform data reduction calculation after the write has been acknowledged to the application, but before the data is migrated to hyperscale flash – thus avoiding the write-amplification created by post-process approaches to data reduction.
- **Data Reduction Dictionary Efficiency:** Storage Class Memory also plays the role of acting as a shared pool of metadata that stores, among other types of metadata, a global compression dictionary that is available to all of the VAST Servers. This allows the system to enrich data with much more data reduction metadata than classic storage can (to further reduce infrastructure costs) while avoiding the need to replicate a data reduction index into the RAM space of every VAST Server (a classic problem with deduplication appliances).

NVMe over Fabrics

NVMe (Non-Volatile Memory express) is the software interface that replaced the SCSI command set for accessing PCIe SSDs. Greater parallelism and lower command queue overhead make NVMe SSDs significantly faster than their SAS or SATA equivalents.

NVMe over Fabrics (NVMe-oF) extends the NVMe API over commodity Ethernet and Infiniband networks to provide PCI levels of performance for remote storage access at data center scale. VAST's DASE architecture disaggregates CPUs and connects them to a globally accessible pool of Storage Class Memory and hyperscale flash SSDs to enable a system architecture that independently scales controllers from storage and provides the foundation to execute a new class of global storage algorithms with the intent of driving the effective cost of the system below the sum of the cost of goods. With NVMe-oF, VAST Containers enjoy the advantage of statelessness and shared-everything access to a global pool of Storage Class Memory and flash, with direct-attached levels of storage access performance.

Scale-out Beyond Shared Nothing

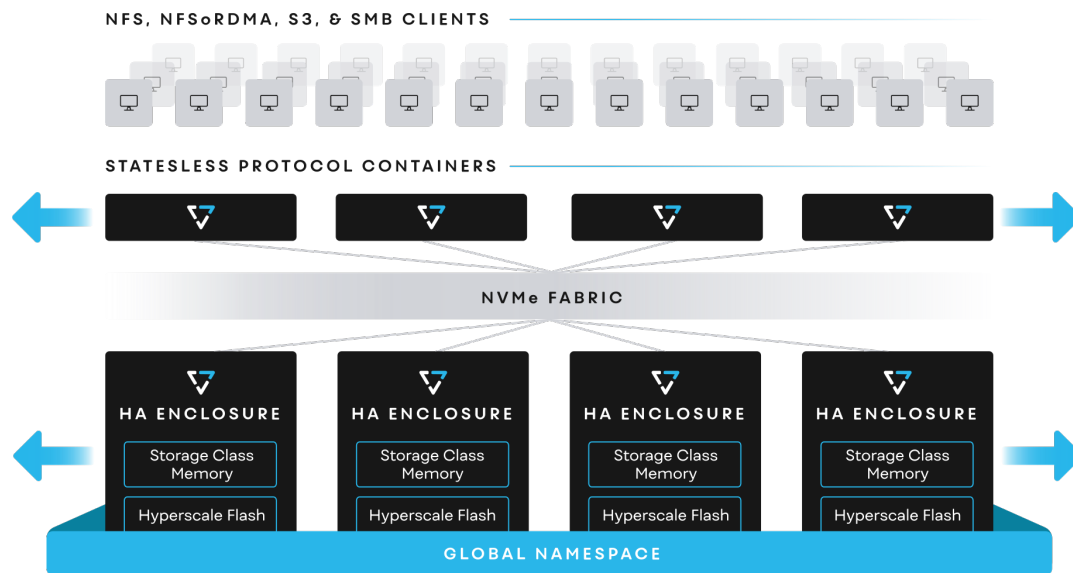
For the past 10 years, the storage industry has convinced itself that a shared-nothing storage architecture is the best approach to achieving storage scale and cost savings. Following the release of the Google File System architecture whitepaper in 2003, it became table stakes for storage architectures of almost any variety to be built from a shared-nothing model, ranging from hyper-converged storage to scale-out file storage to object storage to data warehouse systems and beyond. 10 years later, the basic principles that shared-nothing systems were founded on are much less valid for the following reasons:

- Shared-nothing systems were designed to co-locate disks with CPUs, under the assumption that networks were slower than DAS storage. However, with the advent of NVMe-oF, it's now possible to disaggregate CPUs from storage devices without compromising performance while accessing SSDs and SCM devices remotely.
- Shared-nothing systems force customers to scale compute power and capacity together, creating an inflexible infrastructure model vs. being able to scale up CPUs as the data set needs faster access performance
- Shared-nothing systems limit storage efficiency. Shared-nothing clusters must stripe data across nodes, limiting stripe width and shard, or replicate data reduction metadata, limiting data reduction efficiencies. Shared-media systems can build wider, more efficient RAID stripes when no one machine exclusively owns any SSDs; build more efficient global data reduction.
- As containers become an increasingly popular choice for deploying applications, this microservices approach to deploying applications benefits from the stateless approach containers bring to the table, making it possible to easily provision and scale data services on composable infrastructure when data locality is no

longer a concern.

The DASE Architecture

VAST Universal Storage is based on a new scale-out architecture concept consisting of two building blocks that are scaled across a common NVMe Fabric. First, the state (and storage capacity) of the system is built from resilient, high-density NVMe-oF storage enclosures. Second, the logic of the system is implemented by stateless docker containers that each has the ability to connect to and manage all of the media in the enclosures. Since the compute elements are disaggregated from the media across a data center scale Fabric, each can scale independently – thereby decoupling capacity and performance.



In this Disaggregated Shared Everything (DASE) architecture, every VAST Server in the cluster has direct access to all the cluster's storage media with PCI levels of low latency.

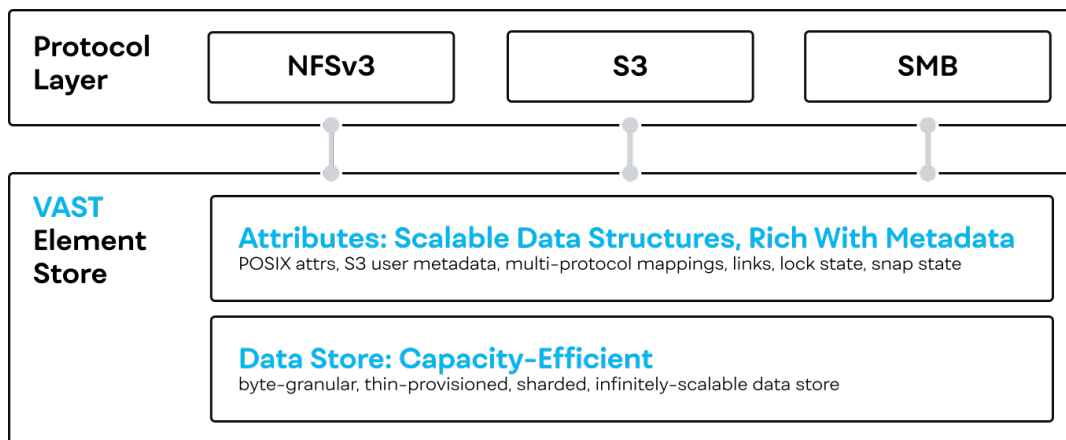
- 'Disaggregated' describes how a VAST Cluster separates the VAST Servers that manage data from the resilient VAST Enclosures that hold the storage media, both flash, and Storage Class Memory. This enables the cluster compute resources to be scaled independently from storage capacity across a commodity, data center scale network.
- VAST's 'Shared-Everything' concept combines NVMe-oF with a set of consistent data structures that makes it possible for all of the data and metadata, across all the enclosures, to be globally accessible by all VAST Servers in the cluster. This global view enables the system to implement global algorithms that define how the cluster builds an atomically consistent namespace, performs global data reduction and data protection.

VAST Servers

VAST Servers, also known as CNodes, provide the intelligence to transform enclosures full of Storage Class

Memory and hyperscale SSDs into an enterprise storage cluster. VAST Servers serve file and object protocol requests from NFS, S3, and SMB clients and manage the global namespace, called the VAST Element Store.

The VAST Server Operating System (VASTOS) provides multi-protocol access to the VAST Element store by treating file and object protocols as interchangeable peers. Clients can write a file to an NFS mount, or an SMB share and read the same data as an object from an S3 bucket (and vice-versa). VASTOS supports NFS including NFSoRDMA (NFS over RDMA) and SMB (Server Message Block, the Microsoft protocol previously known as CIFS) file protocols along with the de facto cloud standard S3 object storage protocol. Each server manages a collection of virtual IP addresses (VIPs) that clients mount via round-robin DNS services to balance load across the cluster.



All the VAST Servers in a cluster mount all the storage devices in the cluster via NVMe-oF, providing global and direct access to all the data and metadata in the system. With this global view, VASTOS distributes data management services (erasure encoding, data reduction, etc) across the cluster's CPUs so that cluster performance scales linearly as more CPUs are added.

VASTOS is deployed in stateless containers that simplify software management across VAST server appliances. Legacy systems must reboot nodes to instantiate a new software version, which can take a minute or more as the node's BIOS performs a power on self-test (POST) on the node's 768GB of DRAM. The upgrade process for VASTOS instantiates a new VASTOS container without restarting the underlying OS reducing the time a VAST server is offline to a few seconds.

Under VAST's Gemini model customers purchase capacity-based subscription licenses from VAST that allow them to run VASTOS containers on qualified hardware they own. VAST has arranged for customers to buy fully integrated VAST Server appliances and VAST Enclosures from our manufacturing partners at VAST's negotiated cost. Hardware maintenance, including SSD replacement is included in the Gemini subscription and VAST will support server appliances and/or enclosures for 10 years from the appliance's initial install date. VAST has also qualified select servers from major vendors for customers with strong preferences for servers from vendor D, vendor L or vendor H.

The Advantage of a Stateless Design

When a VAST Server receives a read request, the Server accesses persistent metadata that is housed in

Storage Class Memory across the fabric in order to locate a file or object's data, and then reads data from hyperscale flash (or SCM if the data has not yet been migrated from the buffer) before forwarding the data to the client. For write requests, the VAST Server writes both data and metadata directly to multiple SCM SSDs before acknowledging writes. This direct access to shared devices over an ultra-low latency fabric eliminates the need for VAST servers to talk with each other in order to service an IO request – no machine talks to any other machine in the synchronous write or read path. Shared-Everything makes it easy to linearly scale performance just by adding CPUs and thereby overcome the law of diminishing returns that is often found when shared-nothing architectures are scaled up. Clusters can be built from 1,000s of VAST servers to provide extreme levels of aggregate performance... the only scalability limiter is the size of the Fabric that customers configure.

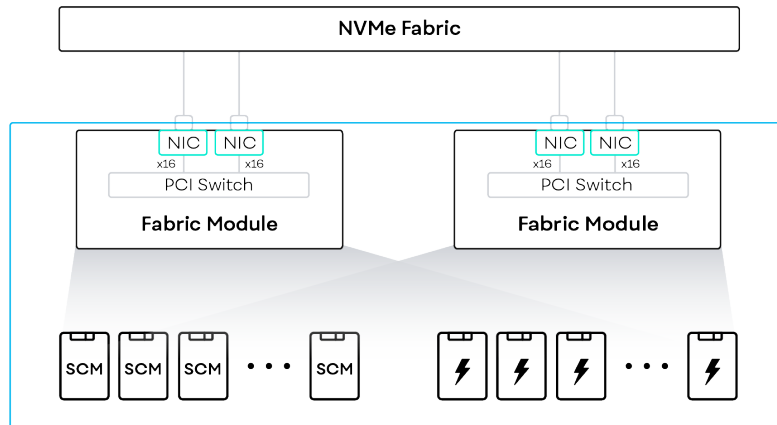
Storing all the system's metadata in shared, persistent SCM SSDs eliminates the need to maintain any coherency between Servers and eliminates the need for power failure protection hardware that would be otherwise required by volatile and expensive DRAM write-back caches. VAST's DASE architecture pairs 100% nonvolatile media with transactional storage semantics to ensure that updates to the Element Store are always consistent and persistent.

VAST Servers do not, themselves, maintain any local state – thereby making it easy to scale services and fail around any Server outage. When a VAST Server joins a cluster, it executes a consistent hashing function to locate the root of various metadata trees. As Server resources are added, the cluster leader rebalances responsibility for shared functions. Should a Server go offline, other Servers easily adopt its VIPs and the clients will connect to the new servers within standard timeout ranges upon retry.

This Shared Everything cluster concept breaks the rigid association that storage systems have historically built around specific storage devices within a cluster. A VAST Cluster will continue to operate and provide all data services even with just one VAST Server running, as all state is stored in a set of globally-accessible and resilient storage enclosures. If, for example, a cluster consisted of 100 Servers, said cluster could lose as many as 99 machines and still be 100% online.

VAST Enclosures

All VAST Enclosures, also known as DBoxes, are NVMe-oF storage shelves that connect SCM and hyperscale flash SSDs to an ultra-low latency NVMe fabric using Ethernet or InfiniBand. All VAST Enclosures are highly redundant with no single point of failure – Fabric modules, NICs, fans, and power supplies are all fully redundant, making VAST Clusters highly available regardless of whether they have one Enclosure or 1,000 Enclosures.



VAST Enclosure Block Diagram

As you can see in the figure above, each VAST Enclosure houses two Fabric Modules that are responsible for routing NVMe-oF requests from their fabric ports to the enclosure's SSDs through a complex of PCIe switch chips on each Fabric Module.

With no single point of failure from network port to SSD, VAST Enclosures combine enterprise-grade resiliency with high-throughput connectivity. While at face value, the architecture of a VAST Enclosure appears similar to a dual-controller storage array, there are in reality several fundamental differences:

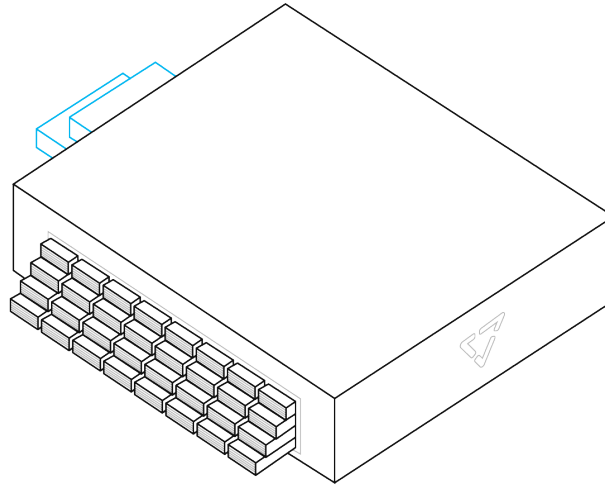
- Fabric Modules do not execute any of the storage logic of the cluster; thus, their CPUs can never become a bottleneck as new capability is added to VASTOS.
- The Fabric Modules don't aggregate SSDs to LUNs or provide data services, unlike legacy controllers. Instead, these servers need little more than the most basic logic to present each SSD to the Fabric and route requests to/from SSDs within microseconds.
- The two Fabric Modules within an enclosure run active-active. Under normal conditions, each Fabric Module presents half the enclosure's SSDs to the NVMe fabric. When a Fabric Module goes offline, the surviving fabric module's PCI switch remaps the PCIe lanes from the failed I/O module to the surviving Module while retaining atomic write consistency.

Fabric Failover With Single Ported SSDs

VAST Enclosures use the redundant PCIe switching fabric provided by the Fabric Modules to provide a redundant path between all of the SSDs and the NVMe Fabric using low-cost single-ported SSDs. Enterprise storage systems have historically used the much more expensive dual-port drives designed specifically for enterprise storage. VAST's unique technology reprograms the PCIe switch fabric to allow VAST systems to provide high availability with single port drives designed for hyperscale data centers, where they're plugged into servers that can only use one port.

VAST Lightspeed Enclosures

The second generation Lightspeed appliance uses a pair of dual Xeon servers with PCIe 3 switching complex as its Fabric Modules with 2 100 Gbps NICs in each fabric module. The 2U Lightspeed enclosure holds 12 SCM and 44 hyperscale/QLC SSDs.



Ceres Enclosures

VAST's third-generation Ceres enclosure once again takes advantage of the latest technologies available. Where those state-of-the-art technologies for LightSpeed were PCIe3, NVMe-oF, and SCM, Ceres uses new form-factor ESDFF SSDs, PCIe4, and Data Processing Units to pack even more performance in a 1U package.

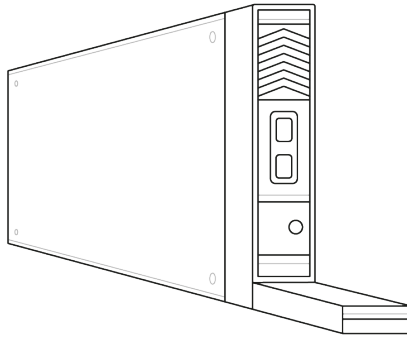
A Ceres enclosure holds 22 QLC SSDs in the ESDFF E1.L form factor along with 12 SCM SSDs.



Ceres Enclosure

New Form Factor SSDs

The U.2 SSD form factor was designed to be the same size as the 2.5" spinning disks of the day to make replacing hard drives with SSDs easier, not because it was an optimal size for an SSD. VAST's Ceres enclosures use SSDs in the EDSFF (Enterprise and Data Center Small Form Factor) E1.L form factor, colloquially called ruler.



The long thin form of ruler SSDs allows Ceres to pack 22 front-loaded SSDs in a 1U space by extending deeper into the available volume than U.2 SSDs and providing significantly better thermal performance than 2.5" SSDs. The entire case of an E1.L SSD is a heat sink, and the SSDs thinness means designers can put every chip in contact with that heatsink.

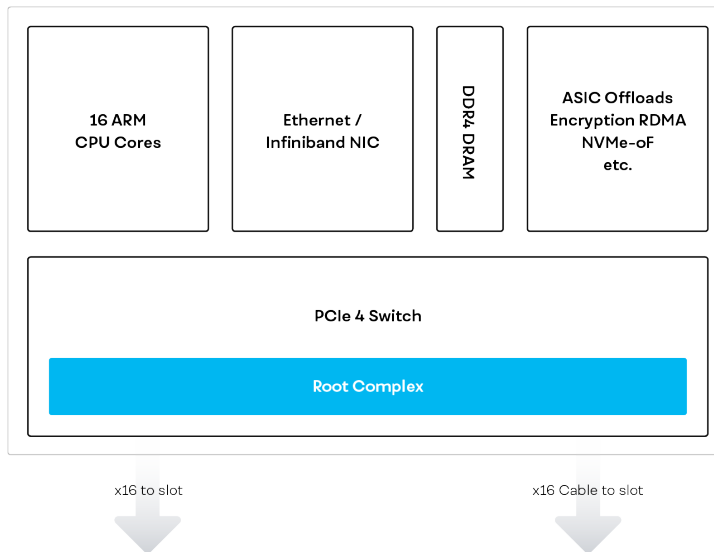
PCIe 4

The upgrade from PCIe 3 in Lightspeed to PCIe 4 doubles the bandwidth of all the connections between the NICs and SSDs. The most significant result is that a 16-lane (x16) PCIe 3 slot has only about 125 Gbps of bandwidth. This means the PCIe 3 Lightspeed enclosure has to use two 100Gbps Ethernet cards because a dual port card would have almost twice as much Ethernet bandwidth as the PCIe slot it plugs into, making the PCIe slot the bottleneck

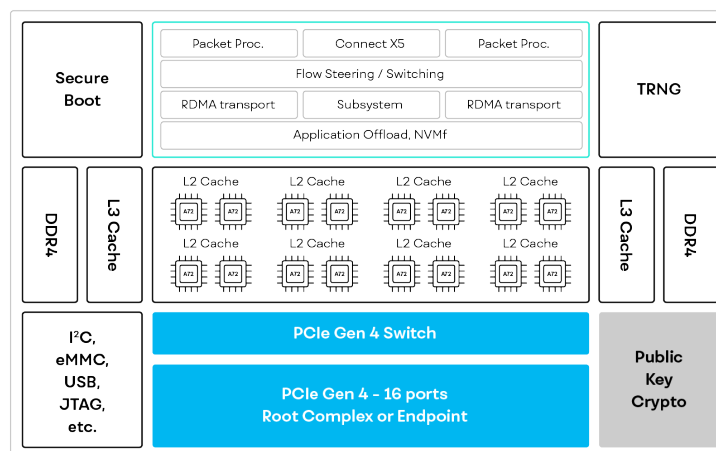
Each PCIe 4 slot in the Ceres DBox provides about 250Gbps of bandwidth, more than enough to support the two 100Gbps Ethernet ports on the Bluefield DPU card that plugs into it.

Data Processing Units

Data Processing Units, sometimes called smart NICs, use an ASIC (Application Specific Integrated Circuit) that combines the usual functions of a network card with intelligence in two forms. DPU ASICs implement functions like RDMA, encryption, and even NVMe-oF in optimized hardware to accelerate those functions and offload the primary CPU from those tasks. DPU ASICs also include a set of computing cores, almost always based on the ARM architecture that customers can use to run their own software. Cloud providers use DPUs like Amazon's Nitro series to offload networking and security tasks allowing their hosts to run more user instances.



DPU Block Diagram



Nvidia's Official Bluefield Block Diagram

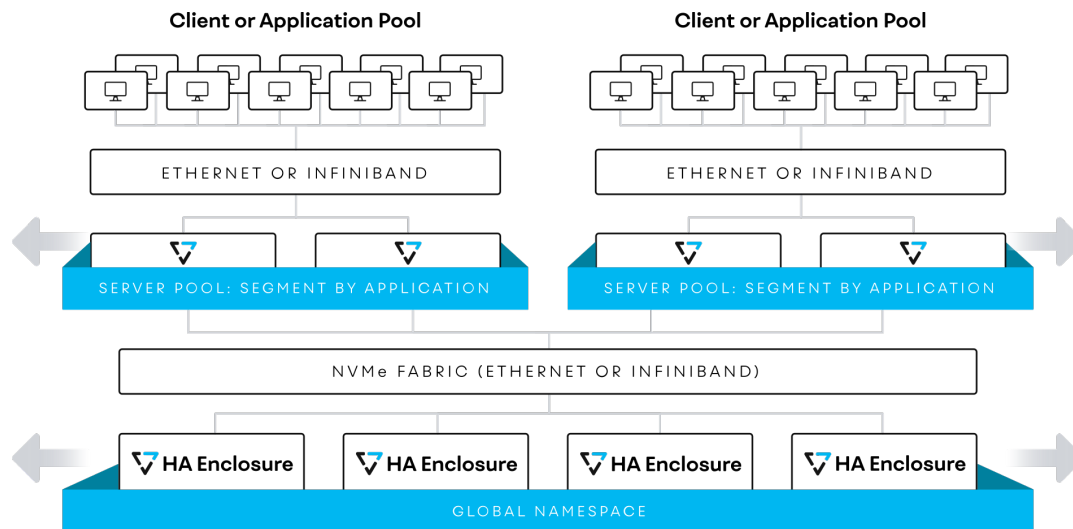
Ceres enclosures use Nvidia's Bluefield DPUs to connect the enclosure to the NVMe fabric and to route NVMe-oF to the enclosure's SSDs. These DPUs consume significantly less space and power than conventional CPUs and NICs allowing a Ceres enclosure to provide up to 675 TB of capacity in one rack unit while consuming only 500 watts.

Each of the two canisters in a Ceres enclosure holds one or two Bluefield cards, each with two 100Gbps network ports providing 200 or 400 Gbps of fabric connectivity.

Server Pooling and Quality of Service

The VAST Servers in a cluster can be subdivided into Server Pools that create isolated failover domains, making it possible to provision the performance of a arbitrarily-sized pool of servers to a set of users or applications to isolate application traffic and ensure a quality of ingress and egress performance that's not possible in shared-

nothing or shared-disk architectures.



Server pooling also provides the advantage of being able to support multiple networks, simultaneously. Users can build their backend Fabric from a common Ethernet or InfiniBand storage network, while also provisioning additional Server front-end ports to talk across multiple, heterogenous, Infiniband or Ethernet subnets. Pooling makes it easy to add file services to hosts on different networks that all access a global namespace.

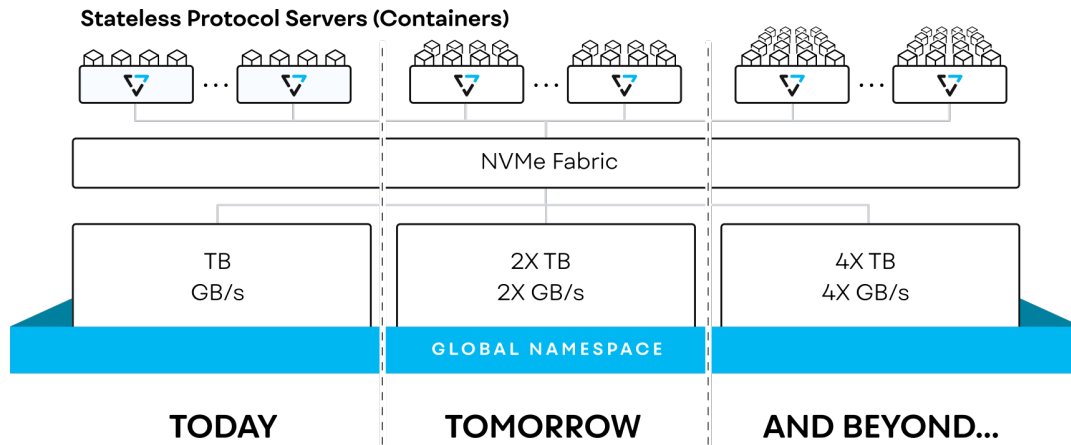
VAST user can also limit access to data views by server pools limiting for example access to the MERGERS view to users connected to the MERGERS server pool enforcing the "Chinese Wall" between groups.

The statelessness of the VAST Server architecture makes it easy to dynamically provision server pools (even programmatically by API) to adapt to the needs of an evolving application stack.

Asymmetric Scaling

Legacy architectures frequently require clusters to be built from homogenous pools of servers and storage. This process of infrastructure pooling often creates rigid boundaries which data can be stripped within and often forces forklift upgrades of whole clusters to add system capacity or performance. VASTOS, on the other hand, is designed to fully support asymmetrical expansion making it simple to add heterogeneous resources to the shared pool of CPUs and SSDs that compose a VAST cluster.

The path to asymmetry is paved with a few architectural advantages that VAST enjoys, namely that flash storage breaks a long-standing tradeoff between performance and capacity that has been exhibited by HDDs. Unlike HDDs which long ago shed any illusions that performance is increasing across HDD generations, the long-term evolution of flash has shown that performance (in terms of IOPS and bandwidth) has evolved proportionately with capacity. By applying this concept at the system level – as VAST storage grows denser, it will also grow proportionately in terms of performance... such that you can think of an asymmetrically scaled flash architecture as intrinsically well-balanced over time.



To accomplish the objective of asymmetry and erase the boundaries of infrastructure utilization in an evolving cluster, VASTOS virtualizes underlying hardware as much as possible in order to enable intelligent scheduling of system services and data placement at two levels:

Server Hardware:

- VAST’s microservice-based architecture enables any stateless storage container to access and operate on any flash or Storage Class Memory device. Storage tasks are allocated globally according to system load; balancing tasks such as data reduction or data protection based on server ability.
- VAST’s global load scheduler is resource-aware, assigning tasks to VAST Servers in the cluster based on their availability. As you add more powerful VAST Servers to a cluster, those servers automatically take on a larger portion of the VIPs and more work in proportion to the fraction of total CPU horsepower they provide in the cluster. Your new and old servers work together, each pulling its weight until you decide they should go.

SSDs:

- Unlike classic storage systems that manage data and failures at the drive level, VAST’s architecture manages data at the Flash erase block level – a much smaller unit of data than a full drive. Data and data protection stripes are simply virtualized and declustered across a global pool of flash drives and can be moved to other drives without concern for their physical locality.
- Since the system doesn’t manage data by the drive, as new drives come into the cluster – the system simply views the new resources as more capacity (more erase blocks) that can be used to create data stripes. When larger SSDs are added to a cluster, those SSDs will be selected for more data stripes than the smaller drives so the system will use the full capacity of each drive.
- The average ratio of IOPS and GB/s will continue to grow proportionately with the amount of TBs you can derive from a flash device. Because flash gets both faster and denser over time, this leads to a natural

balance that is created as more heterogeneous storage resource is added to the cluster over years of operation.

DASE Architecture Benefits Summary

VAST's DASE architecture has several distinct advantages when compared to more traditional shared-nothing and dual-controller architectures. These include:

Disaggregated, Independent Scale	<ul style="list-style-type: none">• Scale the logic and the state of the system independently• Enclosures: Start with as little as one, scale to 1,000• Servers: Start with as little as 2 (for HA), scale to 10,000
Flexible Topologies	<ul style="list-style-type: none">• Server pooling enables tenant/application QOS• Multiple subnets and networks can simultaneously access a common namespace without needing gateways or routers• Asymmetric scaling preserves capital investments and eliminates the forklift upgrade
Dynamic Provisioning	<ul style="list-style-type: none">• Stateless servers can be incrementally added very easily• Recompose pools on the fly as applications evolve• Docker-based storage services lay the foundation for on-the-fly adaptive performance scaling
Enhanced Resilience	<ul style="list-style-type: none">• No DRAM write caching, zero volatility in the data path• No need for cache coherency, data structures are atomic• No need to rebuild data when Servers fail• Up to N-1 Servers can fail while preserving 100% uptime

Linear Scale

- By eliminating east-west cluster traffic, every Server and Enclosure adds a proportionately linear amount of performance to the VAST Cluster

HOW IT WORKS

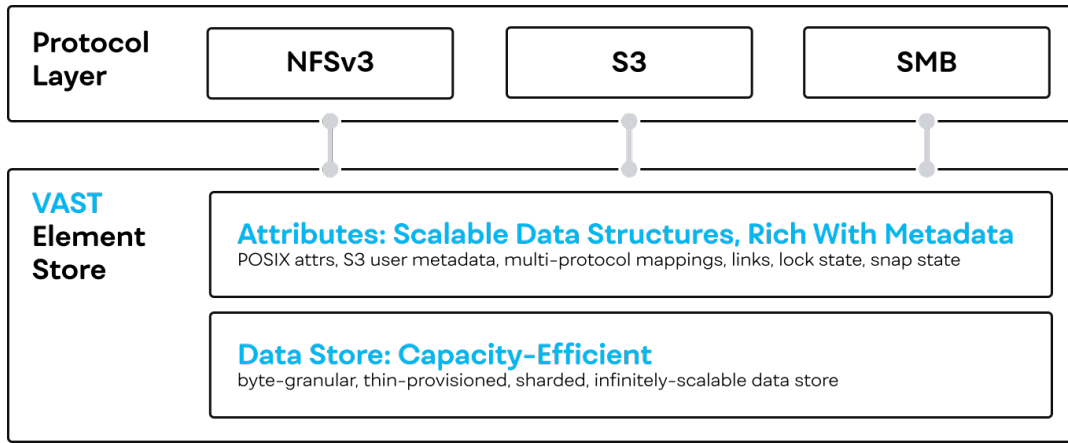
Any modern storage system is defined by its software, and VAST Universal Storage is no exception to this rule. VAST's Universal Storage offering is a SW-defined, API-driven architecture implemented on commodity infrastructure.

The Element Store is a combination of capabilities and inventions that make the Universal Storage concept possible, including:

- A new multi-protocol, exabyte-scale storage namespace called the VAST Element Store, designed to bridge the file and object methods of storing and accessing data without compromising on performance, data access or efficiency.
- A new data layout that leverages deep write buffers to write in free space, providing the transactional semantics of a database while also minimizing flash wear
- A new global flash management system that eliminates write amplification by predictively placing data of a common life expectancy in retention-aware data protection stripes, making it possible to field low-cost, hyperscale flash drives for over 10 years
- A capacity-efficient and performance-efficient approach to snapshots
- A new approach to data protection that ends decades of tradeoffs between data protection cost and system resilience

The Element Store: the VAST Universal Storage Namespace

The Element Store is the heart of VAST Data's Universal Storage. To create a global storage namespace, The Element Store abstracts the hierarchical structure of a file system and the flat structure of object buckets onto a unified set of file/object metadata. The Element Store treats file and object protocols as co-equal conduits to access a common set of data elements.



The VAST Element Store manages all the SSDs in a VAST Cluster to form a single pool of storage and manages that capacity as a single namespace accessible as a file system and/or object store.

Inherently Persistent

All the Element Store's metadata, from basic file names to multiprotocol ACLs and locks, is maintained on shared media in VAST enclosures. This allows the mirrored and distributed metadata to serve a consistent single source of truth regarding the state of the Element Store.

Eliminating the need for server-side caching also eliminates the overhead, and once again complexity, of keeping cached data coherent across multiple storage controllers. VAST systems store all their system state in the shared enclosures that are globally accessible to each server over NVMe-oF. Since each VAST Server accesses a single source of system state directly, they don't need to create any east-west traffic between nodes that shared-nothing systems require in order to update each other's caches. This has 2 significant advantages:

- Since the Storage Class Memory that the metadata is stored on is inherently persistent, there's no data in DRAM or NVRAM cache. Destaging data from a volatile cache is often the largest contributing factor to enterprise storage data loss – as cache management and graceful destaging are simply difficult problems to solve. VAST Universal Storage avoids this issue altogether (even while providing support for exceptionally large and efficient write stripes), and since there's no cache, there's also no need for power-failure protection such as batteries which need to be periodically changed or expensive ultra-capacitors.
- Without the need to coordinate caches, it is much easier to scale I/O services across a scale-out namespace. The Universal Storage architecture eliminates the need for east-west traffic in the synchronous write and read path, thus eliminating a number of operations that would otherwise need to be coordinated across the cluster (metadata updates, lock management, etc.). As server CPUs are added, the cluster benefits from a linearly scalable increase in performance – whereas other systems frequently experience the law of diminishing returns as global update operations need to be shared across an increasingly larger number of nodes.

The VAST Element Store maintains its persistent metadata in V-Trees. VAST's V-Trees are a variation of a B-tree data structure, specifically designed to be stored in shared, persistent memory. Because VAST Servers are stateless, a new metadata structure was needed to enable them to quickly traverse the system's metadata stored on remote SCM devices. To achieve this, VAST designed a tree structure for extremely wide fan-out, each node in a V-Tree can have 100s of child elements – thus limiting the depth of an element search and the number of round trips over the network to no more than seven hops.

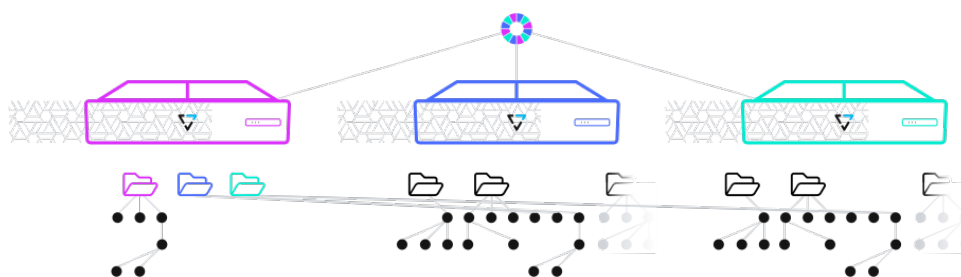
While it isn't organized in tables, rows, and columns, the Element Store's V-Tree architecture enables the metadata store to act in many ways like a database – allowing VAST Servers to perform queries in parallel and locate, for example, an object in an S3 bucket, or the same data as a file, as it existed when the 12.01 AM January 1, 2024 snapshot was taken.

Just as CPUs add a linearly scalable unit of capacity, Element Store metadata is distributed across all the cluster's Storage Class Memory which enables the namespace to scale as well enabling performance to scales as more and the VAST Cluster scales.

Transactionally Consistent

The VAST Element Store was designed to marry the transactional guarantees of an ACID database with the performance of a parallel file system and the scale of an object store. To achieve this goal, VAST Data needed to create a new distributed transaction model, hybrid metadata structures that combine consistent hashing and tree-oriented metadata and new locking and transaction management.

At its core, the Element Store manages its metadata across a shared pool of Storage Class Memory using consistent hashing of each element's (file, object, folder, Etc) handles (unique identifiers), The hash space is divided into ranges with each range assigned to two of the cluster's enclosures. Those two enclosures then hold the metadata roots for elements whose handles hash to values in ranges they are responsible for.



VAST Servers load the 1GB consistent hash table into memory as they re-join the cluster at boot. When a VAST Server wants to find data within a particular file or object, it calculates the hash of the element's handle, and performs a high-speed memory lookup to find which enclosure(s) hold that element's metadata by hash range. The VAST Server can then read the element's V-Tree from the enclosure that is responsible for that portion of the hash space.

By limiting the use of consistent hashing to only the root of each data element, the dataset size per each hash table is very small and the Element Store eliminates the risk of hash collisions as a result. Because hash tables are only required for the root of each data element – the system scales while minimizing the amount of hash

data that has to be recalculated when VAST clusters expand. When a new enclosure is added to a cluster, it assumes ownership of its share of the consistent hash space and only the metadata from those buckets migrated to the new enclosure.

Database Semantics

VAST Element Store's namespace metadata can be thought of in some ways as a database against which the system makes queries to locate pieces of data by file or object name, snapshot time and other metadata attributes. That database metaphor also extends to how the VAST Element Store uses transactional semantics to ensure that the VAST Element Store, like a relational database, is fully ACID (Atomic, Consistent, Isolated, Durable).

Unlike eventually-consistent systems (such as object storage), the VAST Element Store provides a consistent namespace through all the VAST Servers in a cluster to all the users. Changes made by a user on one node are immediately reflected and available to all other users.

To remain consistent, the VAST Element Store ensures that each transaction is atomic. To achieve this, each storage transaction is either applied to the metadata (and all of its mirrors) in its entirety, or not applied to the metadata at all (even if a single transaction updates many metadata objects). With atomic write consistency, the need for classic file system check tools (such as the dreaded fsck) is no longer relevant, and systems can be instantaneously functional upon power cycle events.

Bottom-Up Updates

One key to designing a high-performance transactional namespace is to minimize the exposure to failure during any given transaction by minimizing the number of steps in a transaction that could cause the namespace to be inconsistent if the transaction didn't complete. The VAST Element Store minimizes this exposure by making its updates from the bottom of the V-Tree and working up. When a client overwrites data in an existing file, the system performs the following steps (simplified for the purposes of this document):

1. Data is written to free space on mirrored Storage Class Memory SSDs (via indirection)
2. Metadata objects and attributes are created (BlockID, life expectancy, checksum)
3. The file's metadata is changed to link to any new metadata objects
4. This new write is then, and only then, acknowledged to the client

For operations that fail before step #4 is successfully completed, the system will force clients to retry a write and any old/stale/disconnected metadata that is remaining from the previous attempt will be handled via a background scrubbing process.

If, by comparison, the system updated it's metadata from the top down, that is first add a new extent to the file descriptor followed by the extent, and block metadata; a failure in the middle of the transaction would result in corrupt data from pointers to nothing. Such a system would have to treat the entire change as one complex transaction, with the file object locked the whole time. Updating from the bottom the file only has to be locked when the new metadata is linked in (3 writes vs. 20) shorter locks reduce contention and therefore improve performance..

Transaction Tokens

VAST V-Tree update transactions are managed using transaction tokens. When a VAST Server initiates a transaction, it creates a transaction token metadata object, and increments the transaction token counter. The transaction token contains a globally unique identifier that is accessible from all VAST servers and is used to track updates across multiple metadata objects. This token is infused with the identity of the VAST Server that owns the transaction as well as the transaction's state (ongoing, canceled, committed), for purposes of enabling the system to avoid complications from parallel operations.

As the VAST Server writes changes to the V-Tree, it creates new metadata objects with the transaction token attached. When another VAST server accesses an element's metadata, it checks the transaction token state and then takes the appropriate action using the latest data for committed transactions. If a VAST Server (requesting Server) wants to update a piece of metadata that is already part of an in-flight transaction, it will poll the owning Server to ensure its still operational and will subsequently stand down until the owning server completes the in-flight transaction. If, however, a requesting Server finds in-flight data that is owned by another non-responsive owning Server, the requesting Server will access the consistent state of the namespace using the previous metadata and can also cancel the transaction and remove the metadata updates with that token.

Element Locking

While each read operation within the metadata store is lockless, the VAST cluster employs internal write locks to ensure parallel write consistency across the namespace. Element Locks differ from Transaction Tokens in that while Transaction Tokens ensure consistency during Server failure, Element Locks ensure consistency while multiple writers attempt to operate on a common range of data within the Element Store.

Metadata locks, as with Transaction Tokens, are signed with the ID of the VAST Server that reserved a lock. When a VAST Server discovers a metadata object is locked, it contacts the server identified with the lock, while preventing zombie locks, without the bottleneck of a central lock manager. If the owning Server is unresponsive, the requesting Server will also ask another non-involved Server to also poll the owning Server as a means to ensure that the requesting Server does not experience a false-positive and prematurely fail an owning Server out of the VAST Cluster.

To ensure that write operations are fast, the VAST Cluster holds read-only copies of Element Locks in the DRAM of the VAST Enclosure where the relevant Element lives. In order to quickly verify an Element's lock state, a VAST Server performs an atomic RDMA operation to the memory of an Enclosure's Fabric Module to verify and update locks.

While the above Locking semantics apply at the storage layer, the VAST Cluster also provides facilities for byte-granular file system locking that is discussed later in this document (see: Universal Storage: Protocols).

A Thin-Provisioned, Byte-Granular Data Store

While much of the focus of the above has been about system metadata, and how to ensure consistency of operations in a scale-out cluster – it's important to point out the innovations that can be found in the data

store of the VAST Cluster:

- **Byte Granularity:** There is no fixed data block size within a VAST Cluster. Instead, the VAST Cluster collects data at arbitrary block sizes (as little as one byte) and compacts small writes into larger data blocks that are stored according to the standard block sizes of a modern SSD. As such, the system avoids the capacity loss that can occur in fixed-block storage systems where writing to less than the system's block size can result in sometimes significant capacity waste.
- **Sharded:** While small writes get compacted as they are migrated from Storage Class Memory to flash, writes that are larger than a standard SSD block are sharded and distributed across the cluster such that parallel writers and readers can benefit from the additional infrastructure that comes to bear with large writes (as in: more devices, more servers).
- **Thin Provisioned:** VAST Clusters are thin provisioned such that any data construct (file system exports, directories, files etc.) are all virtualized and distributed across all of the cluster resources. Thin provisioning eliminates much of the headaches that can be created with legacy systems with a unit of namespace scale or administration needs to be defined along some hardware boundary.

Inherently Scalable Namespace

Users have long struggled with the scaling limitations of conventional file systems for a variety of reasons: from metadata slowdowns when accessing folders that contain more than a few thousand files, other systems deal with challenges around having preallocated inodes that limit the number of files that can be stored in a filesystem as a whole. These limitations were one of the major drivers behind the rise of object storage systems in the early 2000s. Freed from the limitations of storing metadata in inodes and nested text files, object stores scale to billions of files over petabytes of data without being bogged down.

The VAST Element Store provides the essentially unlimited scalability promised by object stores for applications using both object and file access methods and APIs. The Element Store's V-Trees expand as enclosures are added to the VAST cluster, with consistent hashes managing the distribution of V-Trees across enclosures (all without explicit limits on object sizes, counts or distribution).

A Breakthrough Approach to Commodity flash Management

In order to build a system that provides game-changing flash economics, VAST Data engineered a system architecture to accommodate the absolute lowest-cost, commodity flash.

While commodity flash has an economic appeal that makes it the media du jour in the hyperscale market – the hyperscale companies that have embraced this style of low-cost flash also write their own applications to overcome the performance and write endurance limitations of these devices. Since most organizations don't

have the luxury of rewriting their applications to benefit from commodity flash; the VAST Data team endeavored to build a system that could serve as an abstraction between any application and commodity flash that certainly was not designed for legacy applications.

In order to enable customers to adopt this new class of flash technology – VAST Data designed a new flash translation layer and data layout to accommodate the unique geometry of high bit density, flash and a new approach to data protection that minimizes namespace fragmentation.

Challenges With Commodity flash

The low cost of commodity, hyperscale-grade SSDs is an important part of how a VAST Universal Storage cluster can achieve archive economics. There are a few barriers that present challenges for legacy systems to use this new type of flash technology:

1. **High Availability:** Hyperscale flash drives are typically single ported, designed only for servers that cannot internally failover and instead stripe data across multiple servers using erasure codes, this limits the wide write striping that VAST clusters aim toward in order to minimize the cost of data protection overhead. To solve this problem, VAST Enclosures can fail over single-ported drives in ways discussed previously in this paper.
2. **Write Performance:** While Hyperscale drive write bandwidth is about $\frac{1}{2}$ of what is achievable as compared to read bandwidth, the number of write IOPS that can be delivered by a Hyperscale drive is, astoundingly, only 2% of the total number of read IOPS that drive delivers. The VAST Architecture overcomes this limitation by outfitting a VAST Cluster with a Storage Class Memory write buffer that is 100x larger than what is typically found in enterprise storage write-back caches and that does not exhibit any of the endurance or write performance limits of hyperscale flash.
3. **Endurance:** Write endurance is the single biggest challenge when considering the use of commodity flash. Where HDDs and legacy generations of flash have near-infinite levels of write endurance, new hyperscale Flash drives have endurance ratings of as low as a few hundred full drive writes before they are worn out, and these drive write ratings also do not account for the data movement that storage systems need to do internally to deal with namespace churn. Much of the rest of this section will discuss how VAST makes it possible to deploy low-cost commodity flash in environments that can enjoy over 10 years of endurance.

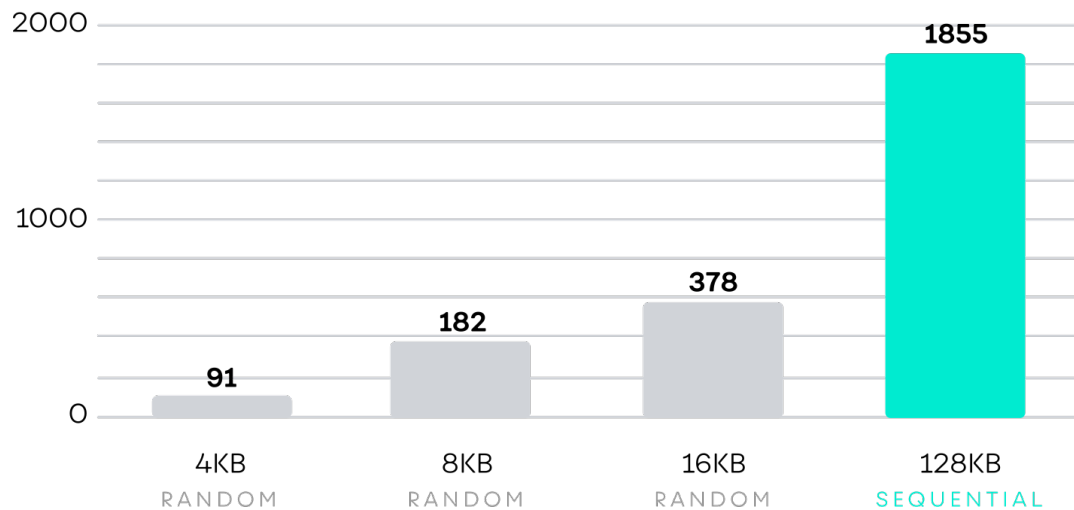
Hyperscale flash Page Size and Erase Block Challenges

SSD endurance is the total amount of data that an SSD's manufacturer warrants a user can write to the drive before the drive fails. Until recently, SSD vendors reported the endurance of their products in DWPD (Drive Writes Per Day for 5 years) or TBW (Terabytes Written) based on a JEDEC standard workload that is based on 77% of the writes being 4 KB or less. This makes sense because many legacy storage products were designed to write in 4KB disk blocks and 4KB is how people tend to think about IOPS workloads – which is what people have been buying flash for in the first place. However, as QLC flash matures in the market and SSD vendors

wrestle with how to position low-endurance media for enterprise workloads, they've now started to reveal more information about how their SSDs react to different write patterns.

As seen in the chart below the endurance of the Intel P4326 SSD varies significantly based on the size and distribution of writes:

QLC SSD drive writes (lifetime) by workload



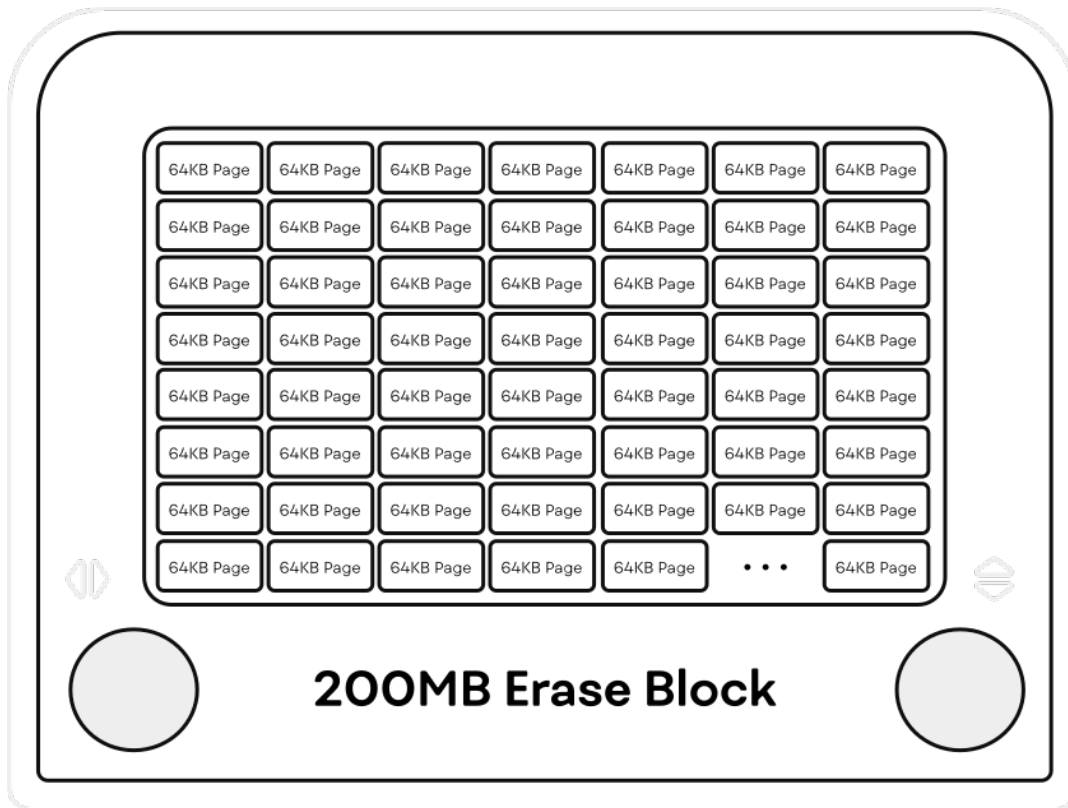
QLC SSD Endurance by workload 1

QLC SSDs provide 20 times more endurance when they're written to in large sequential stripes as compared to being written to in the 4KB random writes, as is often common in many enterprise storage systems.

Why Write Geometry Matters for hyperscale SSDs

To understand why small writes consume so much more of a SSD's endurance than large writes, we have to dive into the internal structure of the flash chips themselves. While flash is often referred to as flash memory, the NAND flash in a data center SSD is very different from DRAM. Where DRAM and Storage Class Memory are bit-addressable media, flash, like HDDs, can only be addressed in discrete blocks.

The flash on each chip in an SSD is organized into a hierarchy of pages, the smallest unit of data that can be read from or written to flash. flash pages are then assembled into much bigger erase blocks. The SSD controller can write to any blank flash page once, but the page must be erased before it can be written to again. To erase a flash page and free up capacity for a subsequent write, a flash chip applies a high voltage to all the cells in an erase block, erasing every cell in the block. In many ways, a flash erase block is like an Etch-a-Sketch. You can draw on each area of the screen once before you have to turn it over, shake and erase the whole screen... you can't just erase the flash pages that you want to recover capacity from.



Flash Pages Organized in an Erase Block: Writing is Easy, Fragmentation is Hard

The pages in today's high bit density flash chips are sized anywhere between 64KB to 128KB, and the erase blocks are up to 200MB. This unit of capacity management is monumentally larger than what classic storage systems were ever designed to deal with. When a system is not designed to deal with these two levels of capacity management, problems creep in.

When the SSD receives large, sequential writes it can write full flash pages to the SSD. When the SSD receives a 4KB write, the SSD is forced to only save that 4KB to a 64KB flash page. Since flash pages are write once, any 4KB write results in 60KB of wasted flash space and more importantly wasted endurance because the unused capacity must be fully compacted by an SSD in order to reclaim unused capacity. Each time the SSD compacts data and moves it from one set of pages to another it consumes one of the limited number of write/erase cycles the flash can endure. This behavior that occurs, when a storage subsystem needs to move data more frequently than when a user writes data in order to free up fragmented flash capacity, is commonly referred to as write amplification... and write amplification is the enemy of low-endurance flash media.



As seen, the geometry, or size, of a flash write is a critical aspect of determining SSD longevity, but it is not the only issue. As explained below, data protection (eg:: RAID striping) is also an area where storage systems can create data fragmentation, causing write amplification.

Universal Storage Data Structures: Optimized for Hyperscale flash

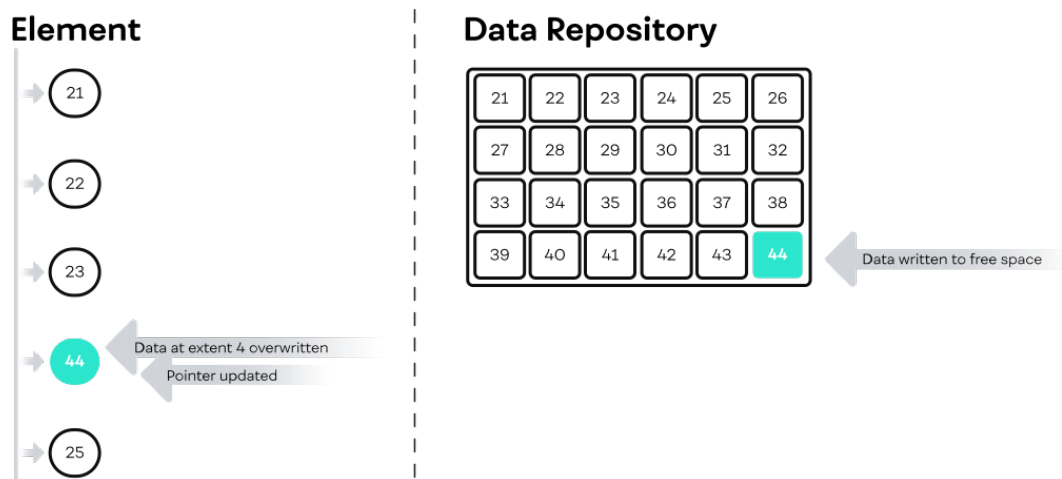
In order to ensure that data is appropriately placed in flash according to a drive's optimized geometry, it is critical that an application is never able to write into a region of a flash drive in place. Since the applications don't know about the limitations of the flash and the flash doesn't know what the applications are trying to accomplish, an abstraction needs to be created between these two in order to ensure that the SSDs realize the best possible longevity.

VAST's design goal has been to use commodity media and avoid imposing upon customers the tax associated with proprietary flash drives or blades. In order to marry this objective with hyperscale low-endurance COTS flash drives, the architecture has been designed to essentially 'spooF' the commodity drives into not knowing what's happening at the logical namespace layer – they therefore never need to manage data at the block level. When it comes to data placement, The VAST Element Store takes many of the concepts that have been popularized by log-structured file systems and extends them to maximize the efficiency of VAST's Similarity-Based Data Reduction, enable the use of very wide write stripes and minimize write amplification to commodity SSDs. Because write cycles are precious, it's critical that a system control writes globally with absolute precision... and controlling data placement starts with indirection.

Indirection

Conventional storage systems maintain static relationships between logical locations, like the 1 millionth-4 millionth bytes of Little_Shop_of_Horrors.MP4, and the physical storage locations that hold that data. As such, when an application writes to that location, the system physically overwrites the old data with the new in place. This, of course, becomes problematic for commodity SSDs when these overwrites aren't the same size as a flash erase block – as anything less than a perfect large (200MB) write will create write amplification.

The VAST Element Store performs all writes by way of indirection. Rather than overwriting data in place, the Element Store writes data to free space on SCM SSDs and builds metadata pointers associating the logical location that the new data was written to with its dynamic physical location. When the system later reduces, erasure-codes, and migrates that data to free space on flash SSDs, it updates the metadata pointers to indicate the new location of data.



If we oversimplify a little, an element (file, object, folder, symlink, etc.) in the VAST Element Store is defined as a series of pointers to the locations on Storage Class Memory or flash... where the data is stored at any given time.

With Large Data Stripes, Drives Never Need To Garbage Collect

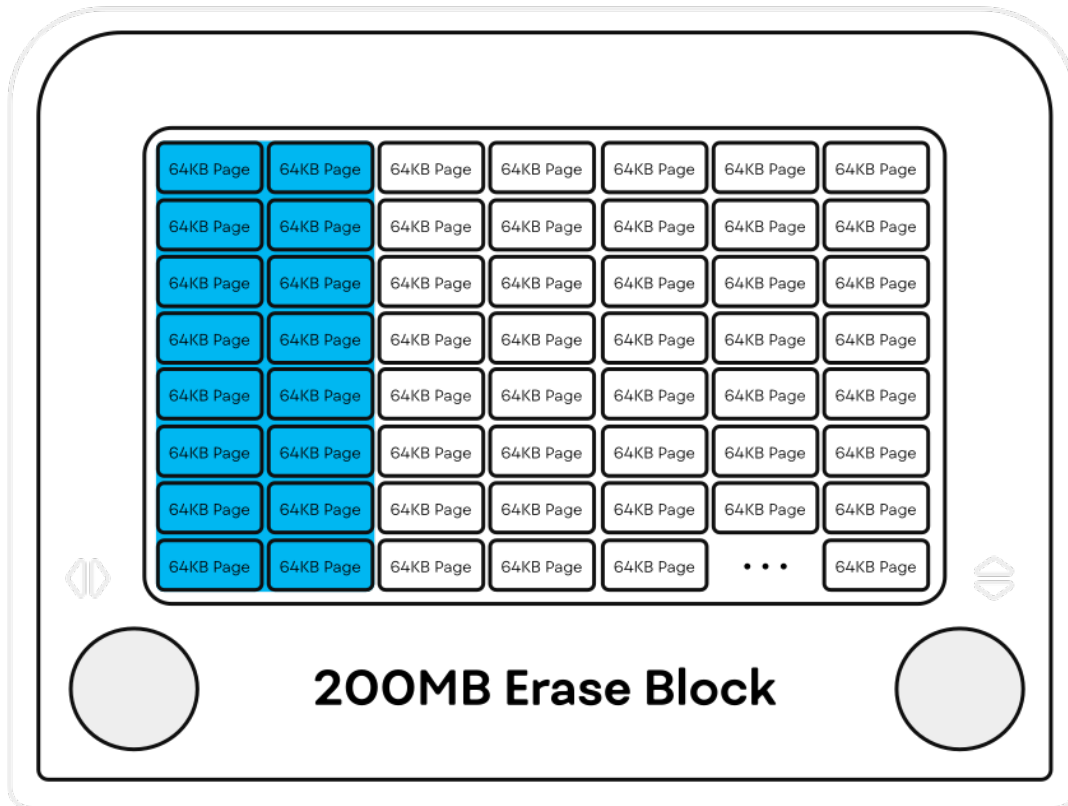
Legacy indirection-based file systems use a single logical block size through all their data services. Using 4KB or 32KB blocks for data reduction, data protection and data storage – this fixed unit of data size keeps things simple for traditional storage systems, especially when sizing file system metadata stores because the systems can always point to a consistently-sized data block.

The problem with this approach is that it forces the storage architect to compromise between the low-latency and fine-granularity of deduplication of small blocks vs. the greater compressibility and lower flash wear of larger blocks. Rather than make that compromise, the VAST Element Store manages data at each stage of its journey:

- the VAST namespace is byte-granular, so flash drive capacity is never wasted
- data reduction pattern matching is byte-granular, and also global across the namespace
- data on the SSDs is managed at much larger units, to align with the SSD's internals

The Element Store writes to SSDs in 1MB I/Os, a significant multiple of the underlying flash's 64KB-128KB page size, thereby allowing the SSD controller to parallelize the I/O across the flash chips within an SSD and also completely fill flash pages, preventing the write amplification that is caused by partially written pages.

1MB Write



1 MB Writes Align to Flash Pages

The Element Store manages data in deep data strips that layer 1MB I/Os into a larger 1GB strip of data on each SSD. Just as writing 1MB I/Os to the SSDs aligns the write to a number of full flash pages, managing data in 1GB strips aligns the deletion of data from the SSDs with the SSD's internal 200MB erase block boundary.

Remember, the VAST Cluster writes in free space, such that any block write or overwrite never happens within a pre-existing strip of data. As with many other operations in the VAST Cluster, garbage collection is performed global across the shared-everything cluster. When enough data from a stripe has been deleted such that the Element Store needs to perform garbage collection, it deletes a full 1GB of data that was previously written to the SSD sequentially. If all of the pages in a block have not been invalidated, the remaining data is written to a new data stripe – but this is an uncommon occurrence thanks to Universal Storage's Foresight feature (discussed later). In essence, this approach allows the system to cleanly write to, and subsequently delete data from, a large number of flash erase blocks while avoiding the SSD from wearing itself out by doing its own unoptimized garbage collection.

The 1GB strip depth and 1MB sub-strip depth are not fundamentally fixed values in the VAST Universal Storage architecture but were determined empirically by measuring the wear caused by writing and erasing multiple patterns on today's generation of QLC SSDs. As new SSDs are qualified and as PLC flash enters the market, the VAST Cluster write layout can adjust to even the larger writes and erases future generations of flash will require to minimize wear.

The underlying architecture components make all of this intelligent placement possible:

- A large distributed buffer, built from Storage Class Memory, provides the system with the time needed to form a collection of erasure encoded write stripes as large as 146+4 – giving the application snappy response times without the need to hold up writes before flushing data down to hyperscale flash
- NVMe-oF and NVMe drives provide the shared-everything cluster foundation that makes it possible for multiple writers to gracefully fill erase blocks with 1MB strips

Endurance is Amortized: Wear-Leveling Across Many-Petabyte Storage

Universal Storage also extends flash endurance by treating the petabytes of flash in a cluster as a single pool of erase blocks that can be globally managed by any VAST Server. The VAST Cluster performs wear-leveling across that pool to allow a Universal Storage system to amortize even very-high-churn applications across petabytes of flash in the Cluster. Because of its scale-out design, the VAST Cluster only needs to work toward the weighted overwrite average of the applications in an environment, where (for example) a database that continually writes 4KB updates will only overwrite a fraction of even the smallest VAST cluster in a day.

The large scale of VAST clusters is also key to the system's being able to perform garbage collection using the 150GB stripes that minimize flash wear and data protection overhead. Unlike some other write-in-free-space storage systems, the VAST Element store continues to provide full performance up to 90% full.

VAST Foresight: Retention-Aware Data Protection Stripes

While VAST's data layout writes only in large 1GB strips in order to ensure that the SSDs don't wake up and perform their own unoptimized garbage collection, this does not solve the problem created by having multiple different streams of data write data of different varieties into a common erase block or erasure code stripe. It could be argued that, with 146+4 striping, VAST Clusters create more opportunity for write amplification by writing erasure code stripes that are comparatively much larger than legacy storage systems. The reason for this is simple: larger write stripes are filled with data from many different ingest streams... and these streams don't write data of a common lifetime retention period, so – when short term data is deleted, the still-valid data in a write stripe must be re-written to a new write stripe, resulting in write amplification.

[Legacy flash Garbage Collection](#)

To address this problem, VAST created Foresight, its method of intelligently placing data into erase blocks according to the expected lifespan of a write. Foresight is based on the simple truth that write amplification is minimized when write stripes consisting of invalidated, or mostly invalidated, flash pages are deleted at the same time.

[VAST Foresight](#)

Until Foresight, journaled and log-structured storage systems basically wrote data to their SSDs in the order the data was written to the system. With SSDs that have large (many-100MB) erase blocks, data from multiple application streams can become intermixed. For example, a single erasure stripe will contain items that will have a long life expectancy, like the contents of a media database, and other data items that have a very short life expectancy, like the indices, transaction logs and scratch files created by that same database engine. Even if this data is stored in different locations to the storage, internal methods of data aggregation provided by storage arrays and flash SSDs will ultimately write the data as it arrives on adjacent pages within common erase blocks that form an erasure stripe.

When a system subsequently performs garbage collection, the short-lived data will have mostly expired, but these data blocks will have been intermixed with the more permanent data written at the same time – leaving a lot of garbage to be collected, and the still-valid pages will create write amplification as the system reclaims SSD capacity. This process repeats over time as the system continues to perform garbage collection of data that has variable retention lifespans – and can result in the same data being moved 10 or 20 times over the lifespan of an SSD, consuming more of the SSDs endurance each time.

Foresight uses the information provided by the data's attributes as well as application overwrite history in order to predict the life expectancy of each new piece of data as it's ingested. In the VAST Cluster's SCM buffer, the system manages a series of buffers that enable the system to fill and flush these buffers asynchronously – in this way Universal Storage breaks from the classical definitions of a log-structured storage system. These buffers are filled with data of a common retentive period – determined by VAST's Foresight algorithms – so each erasure-coded data stripe can be laid down to storage and not result in much data movement when the data in the stripe is ultimately expired, as it should all expire around the same time.

Ephemeral data, such as temporary files and transaction logs, will be written to one "hot" stripe of short-lived data, whereas other stripes can be formed concurrently and consist of "cold" or inactive/archive data which is expected to be retained forever. The VAST Cluster's SCM write buffer is so large that it enables the system to create a multitude of retention-based write stripes, in fact very wide write stripes, while also enabling these stripes to flush to commodity flash only when a full-stripe is created.

The life expectancy prediction is stored in the V-Tree as a metadata value for data, based on:

- VAST Clusters collects statistics about overwrites by file name and extension, for example: tmp files are short-lived
- VAST Clusters collects statistics about Folder, Sub-Folder and Bucket overwrites

While the system cannot be completely omniscient, such that it can absolutely know the life expectancy of any write, a VAST Cluster will also optimize data placement once it has to move a block of data via a garbage collection process. If that piece of data outlived the rest of the data in its original write stripe, it's more likely to live even longer. When the Element Store performs garbage collection, it chooses the stripes with the least remaining data, which usually turn out to be the stripes that were originally filled with ephemeral (short-lived) data. As discussed in detail in the Garbage Collection section below, the garbage collection process will copy

whatever data was incorrectly tagged as ephemeral to a new stripe, with a higher life expectancy.

Hyperscale flash and 10-Year Endurance

One of the primary design tenets of the VAST Element Store is to minimize the write amplification caused by normal namespace churn. The Element Store writes and manages data on SSDs in alignment with the internal page and erase block structure of the flash, wear levels across SSDs in the cluster and acts as a global flash translation layer extending flash management from being strictly an SSD function to managing a global pool of flash.

The result of this combination of inventions is a Cluster architecture that requires 1/10th of the endurance that can be realized from today's generation of commodity SSDs. While this super-naturally high level of system-level endurance paves the way for VAST's future use of even denser flash media, such as PLC flash, it also enables customers to rethink the longevity of data center infrastructure. The net result is that since SSDs don't have the moving parts that cause HDD failure rates to climb after 5 years of use, and VAST's global flash management limits flash wear VAST systems can run in user data centers for 10 years. VAST will replace any failed component, including SSDs that have exhausted their flash endurance, on any system under a Gemini subscription and will offer Gemini subscriptions for up to 10 years from appliance installation.

A Breakthrough Approach to Data Protection

Breaking the Resilience vs. Cost Tradeoff

Protecting user data is the primary purpose of any enterprise storage system, but conventional data protection solutions like replication, RAID, and Reed-Solomon erasure coding force difficult trade-offs between performance, resiliency, storage overhead, and complexity. As referenced in our previous blog on this topic², the VAST Element Store combines Storage Class Memory and NVMe-oF with innovative erasure codes to deliver high resiliency and performance with unprecedentedly low overhead and complexity.

One of the primary initial design tenets of the VAST Cluster architecture was to bring the system overhead down from 66% (an acute case... triplication overhead) to as little as 2% while also increasing the resiliency of a cluster beyond what classic triplication and/or erasure codes today provide. The result is a new class of error correction codes that deliver higher resiliency (Millions of hours Mean Time To Data Loss) and lower overhead (typically under 3%) than previously possible.

VAST Checksums

To protect user data against the silent data corruption that can occur within SSDs, the VAST Element Store keeps a checksum for each data and metadata block in the system. These checksums are CRCs calculated from the data block or metadata structure's contents and stored in the metadata structure that describes the data block. The checksum for a folder's contents is stored as part of its parent folder, the checksum for a file extent's contents in the extent's metadata and so on.

When data is read, the checksum is recalculated from the data block and compared to the checksum in the metadata. If the two values are not the same, the system will rebuild the bad data block using locally decodable parity data.

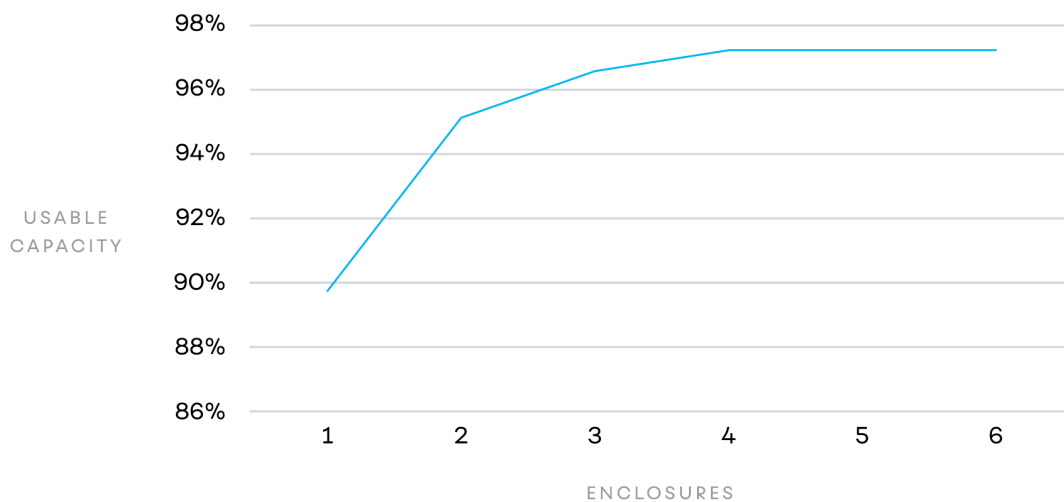
Solutions that store checksums in the same block as the data, such as T10-DIF, only protect data from SSD bit rot, but they do not protect the data path from internal system data transfer errors. If, in the case of T10-DIF, there is as little as a 1-bit error when transmitting an LBA address from an SSD, and the SSD actually reads LBA 33,458, a T10-based system will return the wrong data, because it's reading the wrong LBA, but the data and checksum will match because they're both from LBA 33,458 even though we wanted the data from 33,547.

To protect data being stored on the system from experiencing an accumulation of bit errors, a background process also scrubs the entire contents of the system periodically. Unlike the CRCs that are attached to VAST's variable-length data blocks, the system creates a second set of CRCs for 1MB flash sub-strips in order to swiftly perform background scrubs. This second level of CRCs solves the problem of customers who store millions, or billions, of 1-byte files - VAST's background scrubber can deliver consistent performance irrespective of file/object size.

VAST Locally-Decodable Error Correction Codes

As discussed, the VAST Cluster architecture uses highly-available storage enclosures to hold all the system's flash and Storage Class Memory SSD. Because the VAST enclosure is fully-redundant, with no single points of failure, it is possible to implement a stripe with lower overhead than a shared-nothing architecture where a node is a unit of failure. For example:

- A small collection of VAST JBOFs can provide redundant access to a collection of drives striped across the enclosure in a 146+4 write stripe, but
- If a shared-nothing cluster were to implement this same stripe width, a system would require at nearly 40 storage servers to achieve this same level of low-overhead



Wide write stripes are the means to get the storage efficiency to an all-time low level, but wide stripes also increase the probability of multiple device failures within a stripe. While flash SSDs are very reliable devices,

especially when compared to HDDs, it's simply more statistically likely that two of the 152 SSDs in a 146+2 stripe will fail than 2 strips out of the 12 SSDs in a stripe coded 10+2 .

To solve the problem of increased failure probability, VAST Clusters:

- Generate four parity strips per erasure-coded write stripe, allowing a VAST Cluster to suffer up to four concurrent SSD failures without data loss.
- Stripe width increases from 36 data strips and 4 parity strips per stripe (36+4) on single enclosure systems to 146 data strips and 4 parity strips per stripe (146+4) on systems with four or more enclosures. At scale, n+4 data protection only costs 2.7% of a system's raw capacity

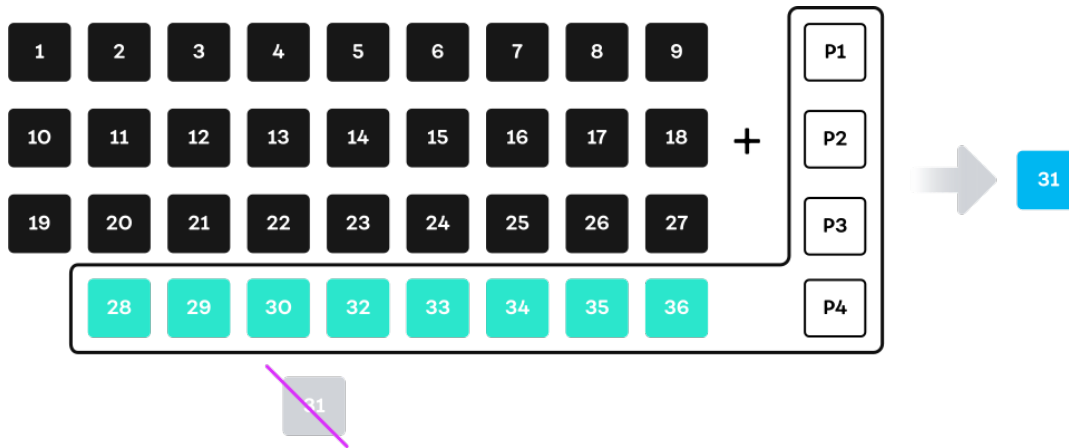
While adding additional parity strips to a write stripe helps increase the stripe's resilience, this is not enough to counterbalance the higher probability of failure that results from large write striping. The other aspect of ensuring high-resilience in a wide write striping cluster is to minimize the time to recovery.

Storage systems have historically kept erasure-coded stripe sizes narrow because the RAID and erasure codes they use, often based on the Reed Solomon error correction algorithm, require systems to read all the surviving data strips in a protection stripe, and one or more parity strips, to regenerate data from a failed part of the protection stripe. To understand this problem in practice, consider the VAST Cluster's 146+4 stripe width – in order to rebuild 1GB of data on an encoded stripe of 146+4, a rebuild event would require reading 150 GB of data from the surviving SSDs if such a system was built to provide a Reed-Solomon style of recovery.

To get around this problem, VAST Data designed a new class of erasure code borrowing from a new algorithmic concept called Locally Decodable Codes. The advantage of Locally Decodable Codes is that they can reconstruct, or decode, data from a fraction of the surviving data strips within a protection stripe. That fraction is proportionate to the number of parity strips in a stripe. That means a VAST cluster reconstructing a 146+4 data stripe only has to read 38 x 1GB data strips, just 1/4th of the survivors.

How Locally-Decodable Erasure Codes Work

VAST's locally decodable codes calculate each protection strip from slightly different sets of data strips across more than just one stripe of data. This allows protection strips to "stand-in" for large groups of data strips during rebuild calculations. By having three parity strips "stand-in" for $\frac{3}{4}$ of a stripe's data strips, the system only needs to read from 1/4th of the surviving data strips in a rebuild in order to perform a recovery within a stripe.



Storage Class Memory Eliminates the Need for Nerd Knobs

Too many storage systems present administrators with a dizzying assortment of stripe widths and RAID protection levels. The VAST Element Store can maintain a fixed selection of n+4 data protection because the very large Storage Class Memory write buffer in a VAST system decouples write latency from the backend locally decodable stripe layout.

Writes to Storage Class Memory are mirrored and the low-latency of Storage Class Memory combines with the fact that unlike flash, Storage Class Memory devices don't have the LBA, page, block hierarchy or the complications that hierarchy creates so their prodigious endurance isn't dependent on write patterns the way flash SSDs are. This means the Storage Class Memory write buffer provides consistently high write performance regardless of the I/O size mix. Any data written by an application is written immediately to multiple Storage Class Memory SSDs. Once data is safely written to Storage Class Memory the write is acknowledged to the application. Data is migrated to hyperscale flash later, long after the write has been acknowledged.

On the flip side, the random access of flash combines with VAST's wide write stripes and parity declustering to ensure that reads are very fast because all writes are parallelized across many flash devices. That said, Locally Decodable erasure codes do not require a full-stripe read on every operation – to the contrary, reads can be as little as one SSD block and as large as a multitude of write stripes. In this way, there is only a loose correlation between the width of a stripe and a width of a file, object, directory or bucket read.

Extensible Erasure Codes

While current VAST systems write all their data in N+4 coded stripes, VAST's Locally Decodable Codes aren't limited to only creating four parity strips. The coding could be extended to 8, or even 16, parity strips per stripe, which would both increase the number of device failures before data loss and reduce the number of data strips the system has to read to rebuild after a device failure to 1/8th or 1/16th of the total. This could be used to support ultra-wide stripes of 500+8 for even greater efficiency or other high-availability use cases.

Intelligent, Data Only Rebuilds

Since the VAST Element Store knows which blocks on each SSD are used, which are empty, and even which are used but are holding deleted data. When an SSD fails, we only have to copy the data and can ignore the deleted data and empty space.

Declassed Parity

Rather than clustering SSDs to hold data strips, parity strips, or act as spares, the VAST Element Store distributes erasure-coded stripes across all the SSDs in the system. The system selects which SSDs to write each stripe of data to based on SSD space and wear, not the locations being written. Data is then layered into the 1GB strips via a collection of sub-strips which can each be written by different VAST Servers.

Because of the shared-everything nature of the DASE architecture, recovery process are sharded across all the VAST Servers in the cluster and VAST's declustered approach to data protection ensures that every device has some amount of data and parity from an arbitrary collection of stripes – making it possible to federate a reconstruction event across all of the available SSDs and all of the Server resources in a VAST Cluster. A large Universal Storage System will have dozens of VAST Servers sharing the load and shortening rebuild time.

A Fail-in-Place Cluster

As storage systems scale to hundreds of SSDs failures are inevitable. VAST systems are designed to allow SSDs to fail in place, rebuilds, even of many SSDs in large cluster are not dependent on failed devices being replaced. Multiple device failures will cause a reduction in the available capacity. That capacity returns when failed devices are replaced with similar or larger devices through the course of a normal replacement event.

Rack Scale Resilience via Enclosure HA

While VAST enclosures, both LightSpeed and Ceres, are highly resilient with no single point of failure, any single appliance, like a VAST enclosure, remains vulnerable to failures of power or network service to the rack they're housed in. Larger VAST systems can remain available even through the loss of a full enclosure by adopting the Enclosure HA option.

A VAST system with 15 enclosures will use erasure code stripes of 146D+4P with 2.7% overhead but since each enclosure holds more strips of each erasure code stripe than the four losses the system can correct for; the system will go offline if any of the 15 enclosures goes offline.

VAST clusters with Enclosure HA use narrower erasure code stripes than VAST systems normally do; writing only two strips of each erasure code stripe to each enclosure. Our 15 enclosure system will now write stripes with 22 data strips and 4 parity strips this raises the protection overhead to 15.4% but since an enclosure failure only removes 2 strips from each stripe, and the locally decodable codes can reconstruct data from as many as four erasures the system can run right through an enclosure failure continuing to serve data.

Low-Overhead Snapshots

A write-in-free space storage architecture is especially suitable for high-performance snapshots because all writes are tracked as part of a global counter and the system can easily maintain new and old/invalid state at the same time. VAST brings this new architecture to make snapshots painless, by eliminating the data and/or metadata copying that often occurs with legacy snapshot approaches.

The VAST Element Store was designed to avoid several of the mistakes of previous snapshot approaches and provides many advantages for Universal Storage users.

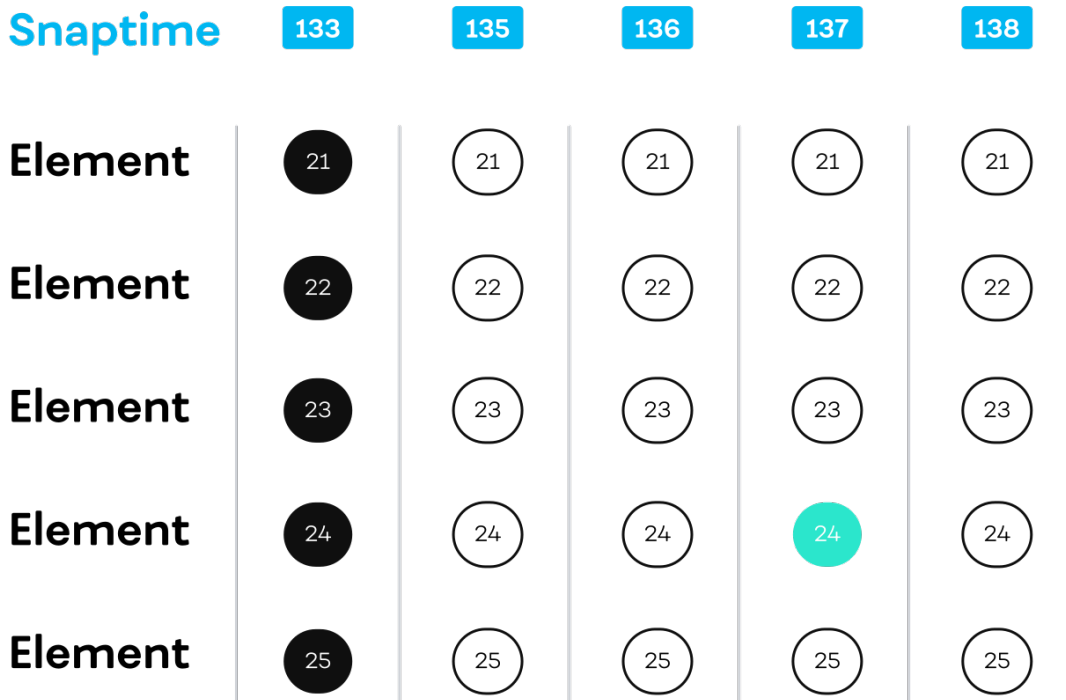
- VAST Snapshots leverage zero-copy mechanisms for both data and metadata in order to minimize the bad

performance effects of storing and deleting snapshots

- VAST Snapshots do not require snapshot space/reserves. Snapshots use any free space and provide greater flexibility of media utilization as compared to more rigid approaches
- VAST Clusters experience negligible performance impact when they have an arbitrary number of snapshots active

Instead of making clones of a volume's metadata every time a snapshot is created, VAST snapshot technology is built deep into the metadata data structure itself. Every metadata object in the VAST Element Store is time-stamped with what is called snaptime. The snaptime is a global system counter that dates back to the installation of a VAST cluster and is advanced synchronously across all the VAST Servers in a cluster approximately once a minute.

As with data, metadata in the VAST Element Store is never directly overwritten. When an application overwrites a file or object; the new data is written in free space across the Storage Class Memory write buffer. The system creates a pointer to the physical location of the data and links that pointer into the V-Tree metadata that defined the object.



Following VAST's general philosophy of efficiency; VAST snapshots and clones are finer-grained than those on many conventional storage systems – allowing users to schedule, or orchestrate, snapshots of any arbitrary file system folder or S3 bucket in the cluster's namespace without forcing artificial abstractions like RAIDsets, volumes, or file systems into the picture.

Indestructible Storage Protects from Ransomware and More

Users today face a plethora of ever more sophisticated attacks on their sensitive data. Snapshots provide protection against ordinary users encrypting or deleting data. Today's ransomware attacks are more sophisticated, acquiring administrator/root/uberuser privileges and deleting or encrypting snapshots and backups as, or before, they attack user's primary data.

VAST's Indestructible Snapshots protect against these sophisticated attacks, and any other attacks that depend on elevated privilege like disgruntled administrators turning rouge by preventing anyone, regardless of their administrative privileges on the system from deleting indestructible snapshots before their expiration date.

While truly indestructible snapshots that no one could ever delete sound appealing to corporate CISOs in the real-world there may be situations where customers need to delete snapshots they've marked as Indestructible or run out of space. In those cases VAST support can provide a time limited token to the user that will temporarily enable deleting indestructible snapshots.

Snap to Object

Snap to Object extends the data protection provided by VAST snapshots to include replicating snapshot data off-site to an S3 compatible object store, which could literally be Amazon S3, or a second VAST Universal Storage cluster.

Users can schedule the times of day and frequency of snapshots to be replicated down to every 30 minutes. We write the snapshot data to an S3 bucket, compressed, in large objects for efficiency.

Once snapshot data is in the cloud, or the other VAST cluster, the snapshot contents are available through the read-only `/.vastremote/policy/snapshot_time/` folder. All a user has to do to restore a file or a thousand files is copy the preserved copy from `/.vastremote`. It's simple and safe enough for end-users to use to restore their own files.

Snap to object, and the `/.vastremote` protect the files on a VAST cluster and make it easy to restore them when the VAST system that made the snapshots is still available. For those cases where the VAST system isn't available, VAST will provide the VAST Reader a Linux virtual machine that can copy files out of the snapshot data objects.

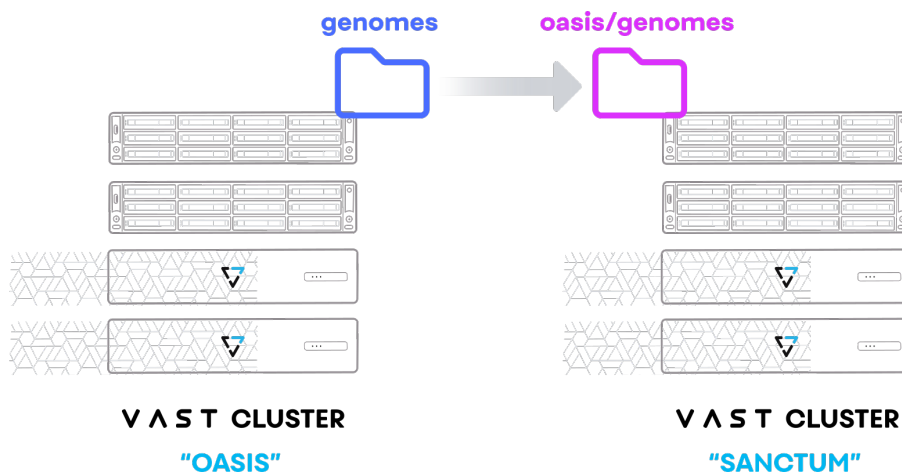
Multiple VAST Readers can read from the same snapshot so users can perform large restores in parallel for better performance. VAST Readers running in the public cloud can export files from VAST snapshots in object storage to a higher performance tier of cloud storage for cloud processing.



Backup Target Examples

Native Replication

Snap-to-Object provides off-site data protection and long term retention but since Snap-to-Object stores its data optimized for object storage in large objects users have to restore their files/objects through the `/.vastremote` folder or VAST cloud reader before they can use their data. As its name implies Native Replication replicates data between folders on two VAST clusters maintaining the data in its native format on the target cluster where it can be accessed directly.



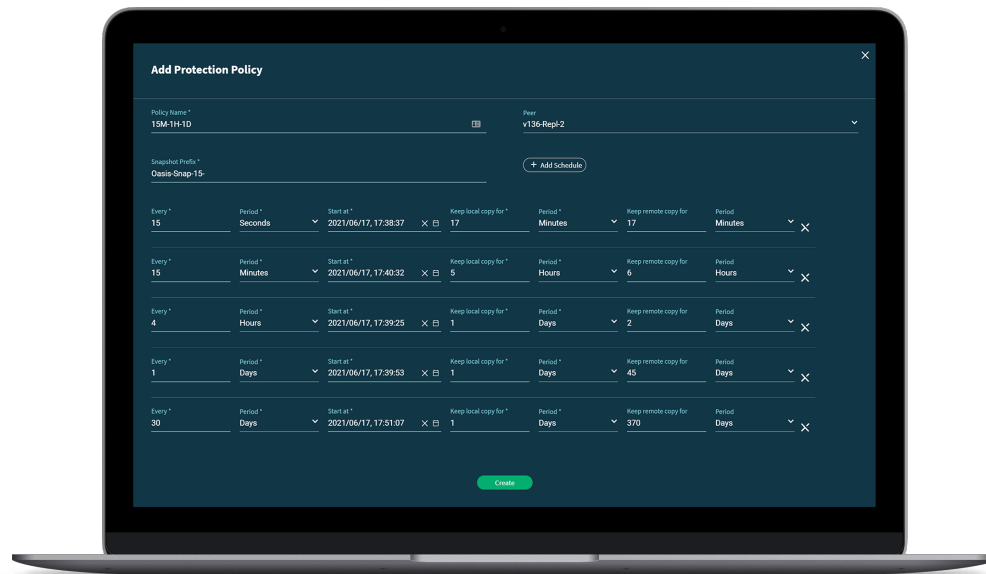
Like Snap-to-cloud, Native Replication, leverages our **snapshot engine** periodically replicating the changed data between snapshots to the target cluster where the changes are applied. The target directory presents the last replicated state of the source directory, read-only to prevent data integrity issues. Since the target folder is on an all-flash VAST system its immediately available for analysis.

There's full support for both graceful failovers, when both systems can still communicate, and graceless failovers for when they can't. Graceful failovers turn the source read-only, perform a final replication, make the former target read-write, and reverse the direction of replication. Administrators can fail-over individual replicated directories for testing or to shift active workloads from one datacenter to another.

The two systems will automatically resynchronize when reconnected after a graceless failover.

Simple, Powerful Policies

Since they're based on the same underlying technology VAST systems manage native replication as an extension of the protection provided by VAST Snapshots. A protection policy defines a schedule of snapshots with multiple tiers based on the frequency and retention periods desired, local and/or remote.



Minimizing RPO

Most storage systems that perform snapshot based asynchronous replication allow users to take, and replicate, a snapshot as frequently as once every 15 minutes. While storage vendors like to call their snapshot frequency the system's recovery point objective (RPO) that ignores the time it takes to take and transfer each snapshot. More realistically a 15 minute replication frequency would allow a user to expect to lose no more than 30 minutes of data, a 30 minute RPO.

While a 30 minute RPO is good enough for many use cases other applications demand smaller RPOs. The low cost of VAST's snapshots allows a VAST cluster to replicate data as every 15 seconds reducing the minimum RPO to under a minute.

VAST systems also monitor replication pairs for RPO compliance generating alerts any time a replication pair fails to sync for two replication periods.

Setting Up Native Replication

Setting up native replication is a simple four step process:

1. Define a VIP Pool for replication
2. Define two VAST Clusters to be a replication pair
3. Define one or more protection policies as described above
4. Define the source directory, target directory and policy for the first protected pair

While protected pairs are replicated in one direction, users can replicate some directories from the VAST cluster named Post to the Pillar cluster while replicating other directories from Pillar to Post.

A Breakthrough Approach to Data Reduction

Storage vendors early-on realized that flash devices provide the benefit of random access, and random access is key to cutting up data and reducing it using classic deduplication and compression approaches – such that data reduction has been used to reduce the effective cost of a flash GB since the first generation of true all-flash arrays. These approaches have proven effective for data that is easily compressible or datasets that exhibit a high occurrence of exact matches, but the conventional wisdom has always been that these data reduction methods have been ineffective for datasets with a higher level of entropy, such as unstructured data.

VAST's new approach to data reduction goes beyond classic data reduction techniques to provide new mechanisms that can find and isolate entropy in data at a much finer level of granularity than previously possible – to make it possible to find and reduce correlations across structured, unstructured and big data datasets.

Beyond Deduplication and Compression

Historically, applications and storage systems have employed two complementary technologies to reduce data: compression and data deduplication.

Traditional compression operates over the limited scope of a block of data or a file. The compressor identifies small sets of repeating data patterns, such as the nulls padding out database fields or frequently used words, and replaces those repeating patterns with smaller symbols. Decompression uses the data to symbol dictionary used during compression to reverse the process and replace symbols with larger data strings. Compression, particularly with unstructured data, is often applied by the application such that storage systems are rarely able to provide additional benefit. Moreover, because the application dictates the content – application-level compression can be semantically optimized for various file types, such as image (JPEG), video (H.264), genome (CRAM) compression, etc..

Data deduplication, by comparison, identifies much larger blocks of data that repeat globally across a storage namespace. Rather than storing multiple copies of the data, a deduplication system uses pointers to direct applications to single physical instance of a deduplication block.

The coarse block hashing that is employed by deduplication is much more sensitive to very small differences in data and therefore force a deduplication system to store a new full block even when there is just a single byte of difference between deduplication blocks. To minimize this sensitivity to entropy in data, deduplication systems can hash to smaller block sizes – but this has the negative effect of driving up the size of the hash index. Smaller deduplication blocks create the need for more metadata, most significantly the used hash tables that must be kept in DRAM for decent performance and CPU overhead, which limits the size of a deduplication realm for single-controller approaches to deduplication and, more importantly, presents a law of diminishing returns as the investment in DRAM at some point overcomes the savings delivered by a deduplication appliance.

Similarity Reduction to the Rescue

While the combination of hyperscale flash, 10 years of system longevity and breakthrough erasure code efficiency all compound into an economically-compelling proposition for organizations who are looking to

reduce the cost of all-flash or hybrid flash+HDD infrastructure, VAST's new approach to global data reduction, Similarity-Based Reduction, further redefines the economic equation of flash to make it possible to build a system that provides a total effective capacity acquisition cost that rivals or betters the economics of HDD infrastructure. The objective is simple: to combine the global benefits of deduplication-based approaches to data reduction with the byte-granular approach to pattern matching which until now has only otherwise been found in localized file or block compression.

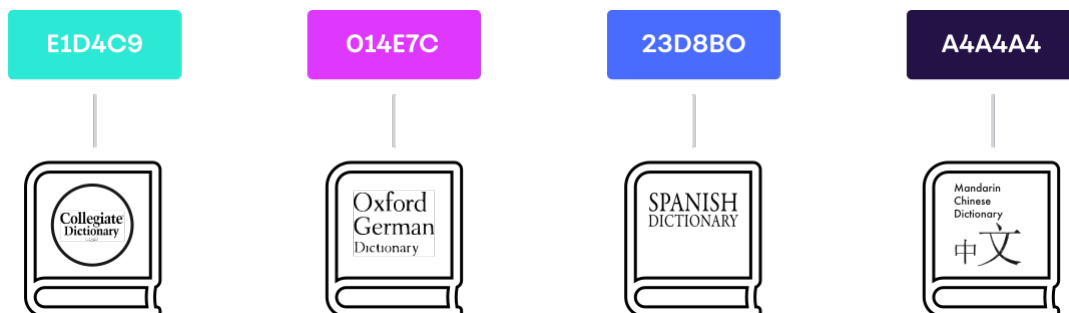
It's counter-intuitive, but the only way to build a system that can beat the economics of an HDD-based storage system is with commodity flash – as the routines that a flash system can use to cut up and correlate data for purposes of data reduction result in a block size on disk that is extremely small, turning every workload into an IOPS workload – where such a block size on HDDs would result in tremendous fragmentation and HDDs simply can't deliver enough IOPS to support fine-grained data reduction.

Similarity Reduction: How it Works

Much like traditional deduplication/backup appliances, VAST's Similarity-Based Data Reduction starts by breaking data into coarse blocks and hashing these blocks after new writes have been persisted into the Storage Class Memory write buffer. This, however, is where the parallels to deduplication appliances, end – as the hash function executed by a VAST Cluster is very different from the types of hashing implemented by deduplication systems.

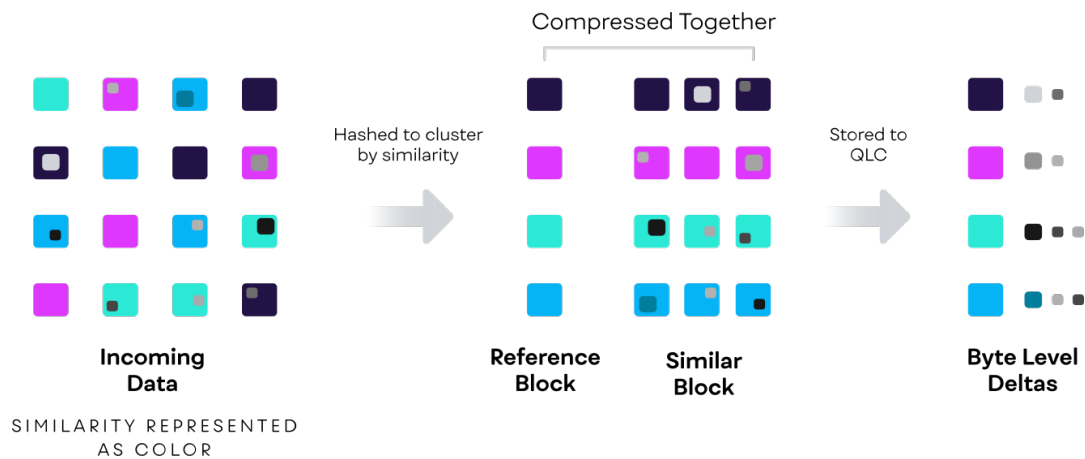
- Deduplication systems use cryptographic hashes, such as SHA-1 or SHA-256, that are designed to be collision-resistant, generating very different hash values from blocks that have small differences in their respective data
- VAST's Similarity Hash, on the other hand, is designed to enable a hashing function to measure the approximate distance between two blocks and generates the same hash from similar blocks of data.

It's a bit of an oversimplification, but Similarity Hashes can be thought of as language clues or semantic markers that suggest a high degree of correlation between different blocks. If two blocks generate similar hashes, they're likely to reduce well when compressed with a common compression dictionary.



To create a cluster of Similarity-hashed and compressed blocks, VAST Clusters compress the first block that generates a specific hash value and declare that block a reference block for that hash. When another block hashes to the same hash ID, the system then compresses the new block against the reference block to create a difference (or delta) block. The data that remains from this global reduction method is a dictionary, which is stored in Storage Class Memory as an attribute of the compressed reference block, and the delta objects that

are stored as reduced symbols that can be as little as a single-byte and who do not also require their own dictionary. If two blocks are exactly the same, there is simply no delta to store.



When applications read from a VAST Cluster, Servers traverse to the different compression dictionaries that are distributed across VAST V-Trees... enabling the system to minimize the scope of decompression to just a single reference block, in essence enabling the cluster to read 4KB objects within 1ms because the cluster doesn't have to decompress the whole namespace per every read as would be the case if all of the data was managed in a single dictionary.

Imagine being able to zip your entire namespace and read from this namespace with millisecond latency... this is the design point for VAST's Similarity-Based Data Reduction.

Similarity Reduction in Practice

The efficiency that can be gained from Similarity-Based Data Reduction is of course data-dependent. While encrypted data will see almost no benefit from this approach to data reduction, other applications will often see significant gains. Reduction gains are, of course, relative – and where a VAST Cluster may be 2x more efficient than a legacy deduplication appliance for backup data (at, say, 15:1 reduction), a reduction of 2:1 may be as valuable in an unstructured data environment where legacy file storage systems have not ever been able to demonstrate any reduction.

Some examples of VAST's similarity reduction, derived from customer testing, include:

- AI/Machine Learning Training Data: 3:1
- Enterprise Backup Files: up to 20:1
- Log Stores (Indexes and Compressed Data): 4:1
- Multi-Tenancy HPC Environments: 2:1

- Pre-Compressed (GZIP) Financial Services Market Data: 1.5:1
- Pre-Compressed Video: 1.1:1
- Seismic Data: 3:1
- VFX and Animation Data: 3:1
- Weather Data: 2.5:1

VASTOS: the VAST Operating System

VASTOS, or the VAST Operating System, is the distributed software framework that exposes the namespace to client applications and provides the service of scale-out cluster management.

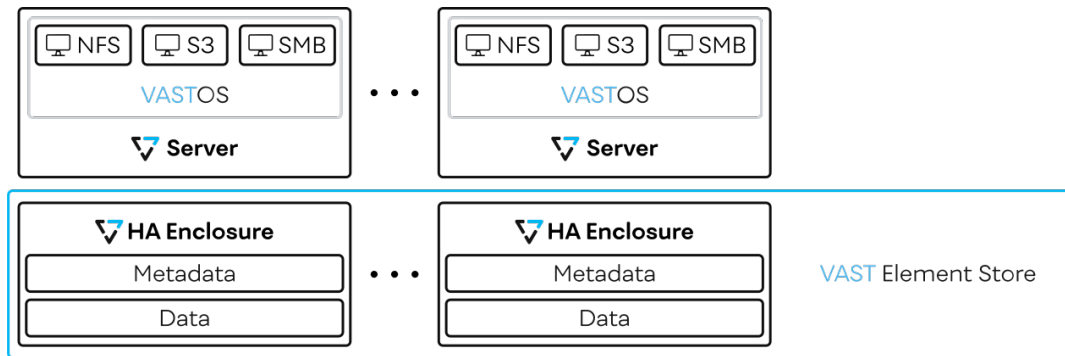
A Realtime Operating System in Userspace

VASTOS is written in Docker and distributed across Linux cores as a sort-of real-time userspace operating system. The real-time capabilities come from how the system manages all of the fibers in a container. VASTOS manages processes at the fiber level because, unlike threads, fibers are a lightweight thread of execution within a server that do not depend on the kernel's thread scheduler; instead fibers are designed to orchestrate the future use of other fibers while they execute, and are therefore less prone to interruption from normal kernel housekeeping. The result of managing CPUs at the fiber level is that the system avoids jitter and can deliver very predictable time-to-first-byte latency.

VASTOS: Multi-Protocol Support

While the VAST Element Store manages a VAST Cluster's media and data, there's more to a storage system than just storing data. The system also needs to make that data available to users and applications, define and enforce access security and enable cluster administration. VASTOS, more specifically, the VASTOS protocol manager does just that.

Just as The VAST Element Store is a namespace abstraction that combines the best parts of a file system with the scale and extended metadata of an object store; the VASTOS Protocol Manager provides a protocol-independent interface to the individual elements in the Element Store. All supporting protocol modules are considered equals, eliminating the need for gateways, data silos and other protocol conversion hacks. The namespace abstraction provided by the Element Store enables VAST Data to add support for additional file, block, big data and yet-to-be-invented protocols over time simply by adding additional protocol modules.



The individual elements in the VAST Element store are also protocol-independent, all elements, and the entire capacity of the VAST cluster can be made accessible by any supported protocol. This allows users to access the same elements over multiple protocols. To provide an example of multi-protocol access in practice: a gene sequencer can store genomic data via NFS, the same application writing data in NFS can enrich that write with additional metadata via an S3 API call, and this data can be made available via S3 to a pipeline build from some cloud framework.

VASTOS supports versions 3 and 4.1 of NFS (Network File System) for Linux and other open systems clients, versions 2.1 and 3.1 of SMB (Server Message Block) the preferred file protocol for Macintosh and Windows computers and S3, the defacto standard for object storage.

One of the challenges to providing truly useful multi-protocol storage is controlling access through the very different security models expected by Linux and other open systems, NFS users, and Windows SMB users.

Classic NFS follows the Unix file security model. Each file has a nine-bit security mask that assigns read, write, and execute permissions to the files owner, a group, and any user that's not a member of the group, known as Other. By only granting permission to a single group, this model made aligning security groups with business organizations difficult.

Posix ACLs add flexibility by adding support for multiple named users and multiple named groups in the access control list for a file or folder.

The access control lists in Windows NTFS are significantly more granular, allowing administrators to control whether users can list the files in a folder, delete subfolders or change the permissions of the folder for other users. Most significantly, Windows ACLs have a deny attribute that prevents a user or group from inheriting permissions from a folder's parents.

NFS 4.1 defines a new ACL format that's granular like Windows ACLs but just different enough to add another level of complication.

As we spoke with users, it became clear that most datasets have a dominant security point of view. Some shared folders will be used primarily by Linux servers with occasional access by Windows systems. Users wanted to manage access to these data sets with POSIX or NFS 4 style ACLs.

The users we spoke to also had other folders primarily used by humans operating Macs and Windows PCs.

Here they wanted the finer-grained access control and familiarity for the users provided by Windows ACLs.

Just as the VAST Element store abstracts data from the file system and object store presentations, it also stores access control lists in an abstracted form. Those abstracted ACLs can be enforced as Unix Mode Bits, POSIX, Windows, or NFS 4 format ACLs as clients access the data via multiple Even with that abstraction, the SMB and NFS views of how ACLs should be processed, modified, and inherited are different enough that we let users assign any View in the system to present NFS or SMB flavor ACLs.

Note that a view's flavor determines which type of ACLs are dominant, not whether the view presents itself as an NFS Export, an SMB share, or both, which is controlled separately.

NFS flavor Views act as NFS clients would expect in every way. Users can query and modify ACLs using the standard Linux tools over NFS. Those ACLs will be enforced on users accessing over both SMB and NFS.

SMB flavor Views are managed like Windows shares allowing users to set fine-grained Windows ACLs through PowerShell scripts and the file explorer over SMB. Those ACLs, including denies, are enforced on NFS as well as SMB access.

VAST NFS: Parallel File System Speed, NAS Simplicity

NFS, the Network File System has been the standard file sharing protocol for open system since 1984. VAST's NFS 3 implementation includes important, standardized extensions to NFS for security (POSIX ACLS), data management (NLM) and several extension for performance including support for NFS over RDMA, and multi-pathing.

POSIX Access Control Lists (ACLs)

VAST is one of the few companies to implement ACLs that conform to the POSIX specification, allowing a system administrator to define broader and more detailed permissions to files and folders than what is possible with the simplistic Unix/Linux model (which is limited to defining read/write and execute permissions to the root, a single user "owner" of the file or folder, and a single group). Unlike Linux mode bits POSIX ACLs are very flexible and allow administrators to assign permissions to multiple users and groups.

NLM Byte-Range Locking

VAST NFS also supports the NLM byte-range locking protocol defined in the standard Linux NFS-util. NLM, which originally stood for Network Lock Manager, defines a mechanism for NFS clients to request and release locks on NFS files and byte ranges within the files. NLM locks are advisory, so clients must test for and honor locks from other clients. NLM provides the support for shared and exclusive locks to applications and is designed for parallel applications where many byte-ranges can be locked concurrently within a single file.

VAST's approach to NLM locking is inherently scalable because locking and lock management is fully distributed across the VAST Cluster. VAST Clusters don't implement a centralized lock manager function or process. Instead, lock information is stored as extended file system metadata for each file in the VAST V-Tree.

Since all the system metadata is available to all the VAST Servers in the cluster, each VAST Server can create, release, or query the lock state of each file it's accessing without the central lock manager server that can so often become a bottleneck on other systems.

NFS Version 4

NFS version 4 was a major re-write of the NFS protocol and NFS v4 provides a significant improvement over NFS v3, especially where security is concerned. Unlike NFS v3 (and S3) NFS v4 is a stateful protocol that includes secure authentication via Kerberos and the detailed ACLs mentioned above.

VAST systems support the session-based NFS v4.1 over both TCP and RDMA including support for NFS 4 byte-range locking, Kerberos authentication, encryption in flight, and NFS v4 ACLs.

NFS over RDMA (NFSoverRDMA)

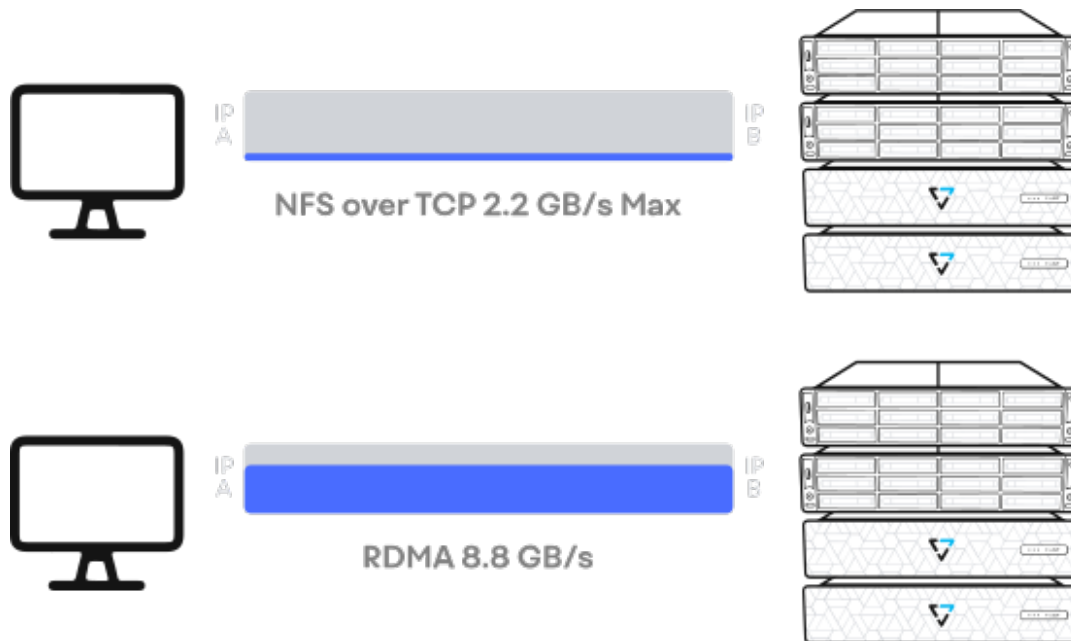
Traditional NFS over TCP has been performance-limited because the TCP transport protocol will only send a limited amount of data before it receives an acknowledgment from the receiver. This limit on data in flight, sometimes called the bandwidth-delay product, restricts a single NFS over TCP connection from a client to a server to approximately 2GB/s even on 100Gbps networks.

Users have traditionally taken one of two complex approaches to solving this problem. They can use multiple mount points, and therefore multiple connections to the NAS or switch to a complex parallel file system. Multiple mount points provide a limited performance boost at the cost of more complicated data management and does nothing to provide a single application accessing a single mount more bandwidth. Parallel file systems require client-side file system drivers that limit and complicate client OS choices and updates.

In the mid-2010s – Oracle and several other key contributors got together and NFS support using RDMA to transport NFS RPCs between the client and server instead of TCP. Thanks to their work, NFSoverRDMA has been pushed upstream to become part of all the major Linux distributions.

RDMA (Remote Direct Memory Access) transfers data across the network directly into the memory of the remote computer, eliminating the latency of making copies of the data in TCP/IP stacks, or NIC (Network Interface Card) memory. The RDMA API, known as RDMA Verbs is implemented in the RNIC (RDMA NIC) offloading the transfer from local memory directly into memory on a remote computer and eliminating the user space/kernel space context switches that add overhead to TCP connections.

RDMA Verbs were first developed for InfiniBand so NFSoverRDMA can, of course, run over standard InfiniBand networks. More recently RDMA Verbs have been built into Ethernet RNICs using the RoCE (RDMA over Converged Ethernet) bringing NFSoverRDMA to Ethernet as well as InfiniBand data centers. RoCE v2 runs over UCP and doesn't require any special network configuration beyond enabling ECN (Explicit Congestion Notification) a standard feature of enterprise Ethernet switches since 10Gbps ruled the roost.



The net result is that where a single NFS over TCP session to a VAST Cluster is limited to 2GB/s a single NFSoRDMA session can deliver up to 70% of a 100Gbps network's line speed, or 8.8 GB/s.

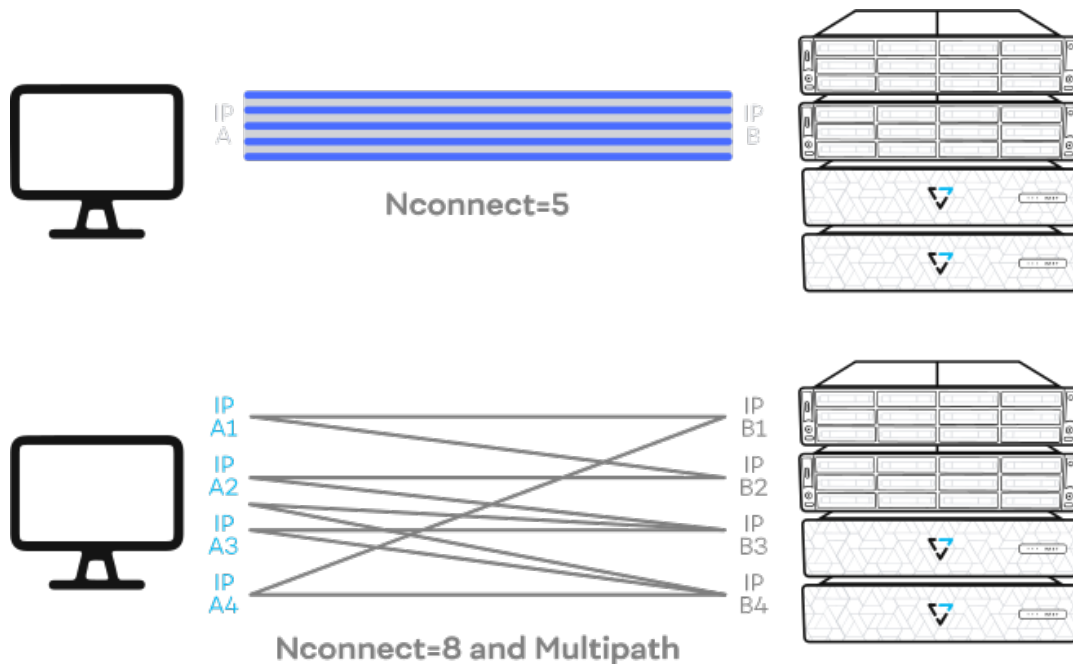
But the real beauty of NFSoRDMA lies in the simplicity of deployment, the NFSoRDMA client is standard in major Linux distributions, it doesn't require any of the kernel patches or client agents that can make deploying and maintaining high-bandwidth parallel file system storage problematic.

Multipathing for Bandwidth and Fault Tolerance

While NFSoRDMA eliminates the TCP session bottleneck it's not the ultimate solution for NFS performance. Running requires RNICs in client systems and while RDMA will saturate a 100Gb/s network connection the NFS 3 client will only use one network connection. The next step is to load balance, and just as importantly provide resiliency, across multiple network links with a multipath driver the way block SAN protocols do.

The first step to multipathing NFS is the `nconnect` NFS mount option, which was added to the NFS client with the Linux 5.3 kernel. When the NFS client mounts an export with the `nconnect=n` option it will load balance access to that export across `n` TCP sessions, not just one. Since the NFS client is still using the kernel TCP/IP stacks, which comes with a significant amount of the context switch overhead and latency, it's slower than NFSoRDMA but can be a big improvement for some clients.

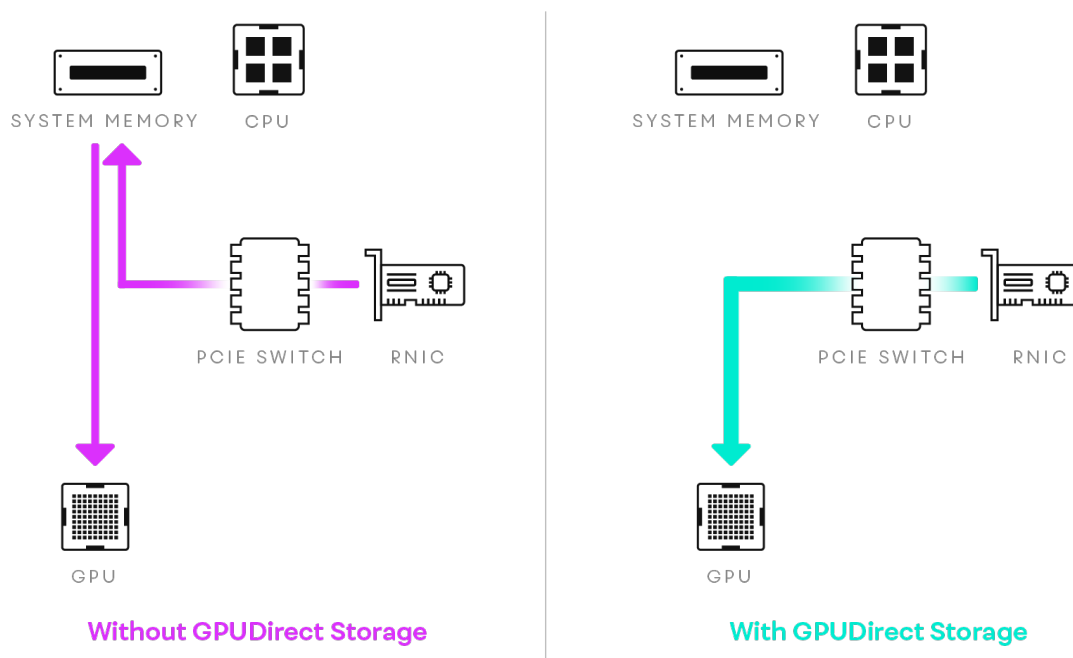
The next step, developed at VAST and now being pushed upstream, was to extend the multiple connections that `nconnect` provided from multiple TCP connections from one IP address to another to multiple connections from a range of client IP addresses on the client to a range of server IP addresses. The NFS client manages IP address pair paths that can be used for both TCP and RDMA.



When the client IP addresses are assigned to separate NICs it will load balance across two 10 or 25 gbps Ethernet ports for an edit station. If any path from client to server fails, the client just directs traffic to the surviving paths.

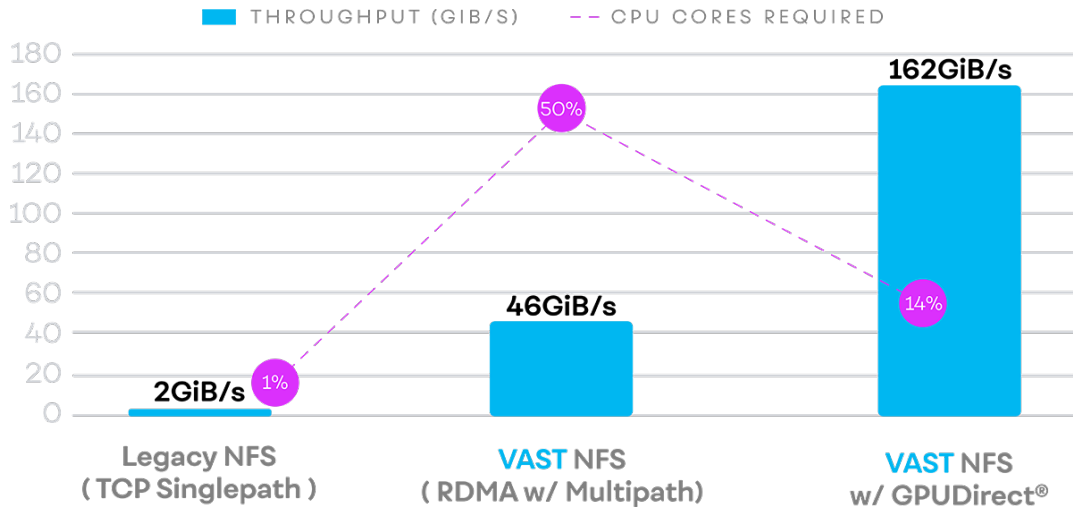
GPU Direct Storage

The GPU servers running artificial intelligence applications process orders of magnitude more data than general purpose CPUs possibly could. Even with multiple 100 Gbps NICs the GPUs would spend too much time waiting for data to be copied from a NIC data buffer into CPU memory and only then into the GPU's memory where it could be analyzed.



[NVIDIA's GPUDirect Storage](#) provides a direct RDMA path for NFS data from an RNIC into the memory of a GPU bypassing the CPU, and the CPU's memory. GPU Direct Storage (GDS) vastly increases the amount of

data GPUs can access by eliminating the main memory to GPU bandwidth bottleneck of 50GB/s. Using GDS the server can transfer data into the GPU from multiple RNICs in parallel boosting total bandwidth to as much as 200GB/s.



Tested With GDSIO • 1X DGX-A100 • 5 X VAST Cbox
 Server Chassis & 5X VAST Lightspeed Enclosures • 4MB
 I/O Size • 4GB File Size • 96 Threads X 8 GPUs

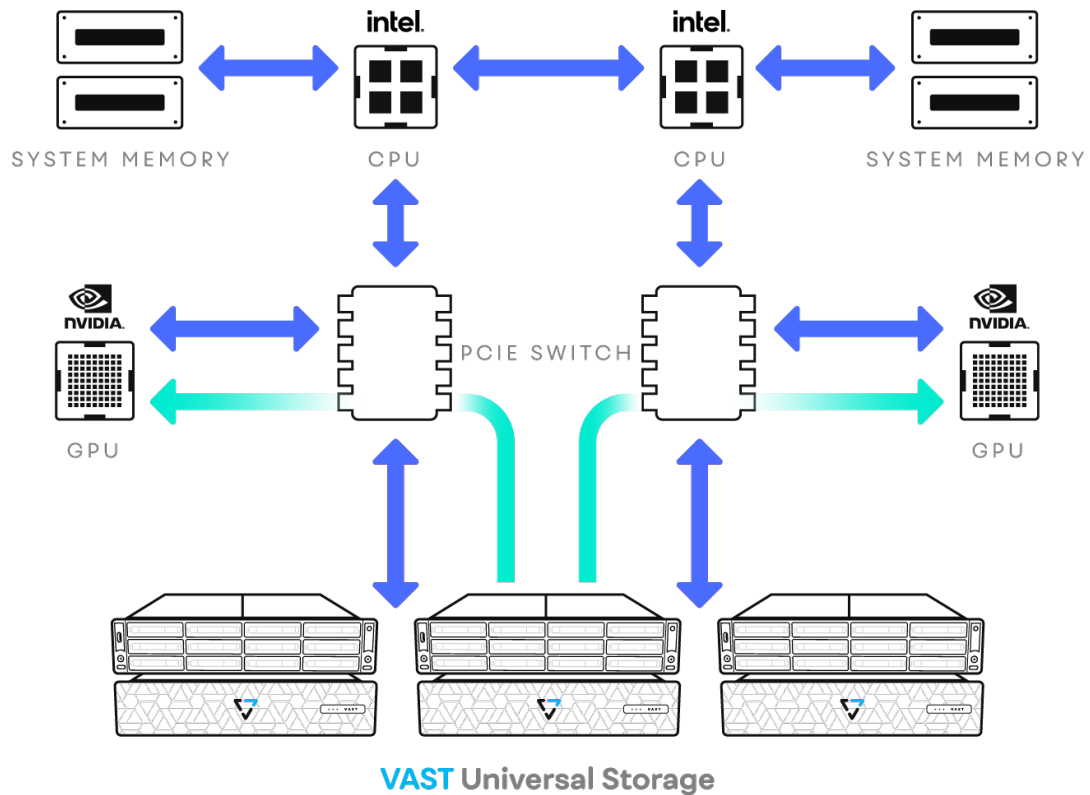
GPU Direct Storage also reduces the overhead of accessing storage. When we tested a VAST Cluster with an [NVIDIA DGX-A100 GPU server](#), as shown in the chart above, the combination of NFS over RDMA, and multipath load balancing across the DGX-A100's eight 200Gbps HDR InfiniBand boosted total bandwidth from the NFSoTCP's 2GiB/s to 46GiB/s but moving all that data through main memory to the GPUs eat up half the DGX-A100's 128 AMD ROME cores.

Switching to GPU Direct Storage not only raised throughput to 162GiB/s but also reduced the amount of CPU required from 50% to 14%.

NUMA Aware Multipathing

The PCIe slots, like the memory sockets, in a modern server are directly connected to one of the two CPUs. The result is NUMA (Non-Uniform Memory Access) where a process running on CPU-A can access memory directly connected to CPU-A faster than memory connected to CPU-B. Accesses to remote memory has to cross the bridge between the two CPU sockets and the bandwidth of that channel can become a bottleneck.

GPU Direct Storage accesses perform RDMA transfers from the RNIC in one PCIe slot to the GPU in another slot. The NFS multipath driver is NUMA aware and will direct data from the VAST cluster to an RNIC on the client that's connected to the same CPU/NUMA node as the GPU that will process the data. This keeps GDS traffic off the CPU-CPU bridge and away from that bottleneck.



VAST SMB: File services for Windows and Macintosh

While NFS is the native file access protocol for Linux and most other Unix derivatives NFS isn't the only widely used file protocol. Windows and Macintosh computers use Microsoft's Server Message Block (SMB) protocol instead of NFS. The SMB protocol is also sometimes known as CIFS after an unsuccessful proposal by Microsoft to make SMB 1.0 an internet standard as Common Internet File System.

SMB is such a complex, stateful protocol that many NAS vendors choose to acquire through the open-source SAMBA project or from one of the handful of commercial vendors that sell SMB code. Unfortunately, all the available SMB solutions have limitations, especially around scaling, that keep them from being good enough for VAST.

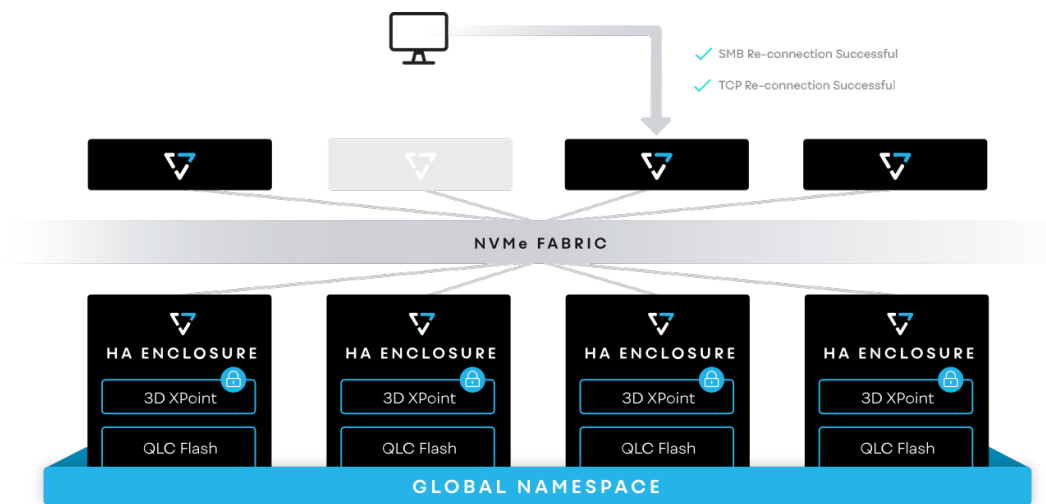
Instead VAST developed our SMB protocol in-house to take full advantage of the DASE architecture storing session state in the cluster's shared SCM pool to allow SMB service across all the cluster's CNodes. Since Universal Storage is fully multi-protocol, SMB users can access the same files and folders as users accessing the system via NFS and/or S3.

File access to the VAST namespace is controlled by VAST Views. A VAST View is a protocol independent version of an NFS Export and SMB Share. Administrators can enable NFS and/or SMB access to a view and set the access control flavor of the View to NFS or SMB.

VAST systems support SMB 2.1 and 3.1 including SMB Multichannel for enhanced throughput.

Every time an SMB client opens a file on an SMB server, the two systems assign an SMB handle to identify the

connection. This requires both the client and SMB server to retain state information relating clients, files, open modes, and handles.



State, locks, lease: fail over in microseconds. Upgrades are simple, uptime is mastered.

Many scale-out storage systems force users to manually retry or reconnect after a node failure, because while another node may take over the virtual IP address from the failed one, the dynamic state information, like handles, is lost. For an SMB client to automatically recover from a node failure, the state information has to be shared, and the shared-nothing model breaks down.

In a VAST server, dynamic handles, file leases, and all the other state information that defines the relations between SMB clients and servers is stored in shared Storage Class Memory in the cluster's VAST Enclosures. This allows the surviving VAST servers in a cluster to not only assume the IP addresses of a VAST server that goes offline, but also to resume where the offline server left off, reading the connection's state from Storage Class Memory.

The SMB client sees the equivalent of a transient network glitch as the system detects a failed VAST server and recovers, exactly what SMB 2.1's resilient handles were designed to accommodate. No lost handles, and most importantly, users and applications continue as if the failure never happened.

VAST S3: Object Storage for Modern Applications

Amazon's S3 protocol, or more accurately the protocol used by Amazon's (S3 Simple Storage Service), has emerged as the de-facto standard for object storage in no small part as it allows developers to support both on-premises object stores like VAST's Universal Storage and cloud storage such as Amazon S3 and its competitors.

VAST's Protocol Manager exports S3 objects using HTTP GET and PUT semantics to transfer an entire object via a single request across a flat namespace. Each object is identified and accessed by a URI (Universal Resource

Identifier). While the URIs identifying files can contain slashes (/s) like file systems, object stores don't treat the slash character as special, so that it has the ability to emulate a folder hierarchy without the complexity a hierarchy creates – slashes are just another character in an internal element identifier.

VAST Objects are stored in a similar manner to files, with the difference being that an S3 object includes not just its contents, but also user-defined metadata that allows applications to embed their metadata about objects within the objects themselves.

While object storage has classically been used for archival purposes – the emergence of fast object access and, in particular, all-flash object storage has extended the use cases for which object storage is appropriate for. For example, many Massively Parallel Processing (MPP) and NoSQL Databases use object storage as their underlying data repository.

VAST Universal Storage systems support a subset of the S3 verbs that are offered as part of Amazon's S3 Service. Whereas many of Amazon's APIs are specific to their service offering, VAST Clusters expose the S3 verbs that are required by most applications benefit from an all-flash on-premises object-store. That excludes tiering, a VAST system provides one tier, all-flash, and AWS specific factors, like the number of availability zones each object should be replicated across.

As with VAST's NFS offering – S3 performance scales as Enclosure and Server infrastructure is added to a cluster. For example, a Cluster consisting of 10 enclosures can be read from at a rate of up to 230GB/s for 1MB GET operations, or at a rate of 730K random 4KB GETs per second.

Batch Delete: The .Trash folder

Many user workflows include cleanup stages that delete a working directory and its contents. VAST's batch delete feature offloads this process from the NFS client to the VAST system, which both allows workflows to proceed without waiting and reduces the load on both the customer's compute servers and the VAST system.

Deleting large numbers of files via `RM -rf /foo/bar` or some equivalent can take a considerable amount of time as `RM` command walks the directory tree with a single thread looking up and deleting files one by one.

To delete a folder and its contents, a user moves the folder to be deleted to a special `.trash` folder in the root of the folder's NFS export. Once the folder has been moved, the VAST system deletes its contents in the background.

Moving a folder is a single NFS rename call, relieving the client from walking the directory tree and deleting the files one by one.

Since the actual deletion occurs in the background, the system's free space available won't reflect the space occupied by the deleted files for some minutes after they're moved to the `.trash` folder.

- The `.trash` folder can be enabled or disabled by policy at the export and tenant levels

- .trash is a hidden folder bit-bucket blackhole. Users cannot change its ACLs via CHMOD or access its contents in any way.
- Managers can set the GID to limit moving folders to the .trash to members of a group through the GUI/Rest API
- Other vendors have implemented similar functions:
 - MS/PC-DOS's DELTREE command
 - Isilon Tree-Delete command

Platform Integration Drivers

VAST's Universal Storage system was designed for modern applications in modern data centers. That means a data center where resources are allocated and managed by automated orchestration systems using APIs, not storage administrators clicking through GUIs. Universal Storage has, as discussed above, an API first design where all management functions are designed into the REST API first, and the GUI consumes the same REST APIs that our customer's platforms and scripts do.

Platform integration goes beyond merely publishing an API but instead providing a driver that connects a compute platform like vSphere (VAAI, VASA), Kubernetes (CSI), or Red Hat OpenShift. This driver tells the platform how to authenticate to a storage system and use its API provision storage volumes and allocate them to the VMs/Containers they orchestrate.

Containers are the latest, most efficient way to deploy applications, which is why we deploy VAST Servers in Docker containers ourselves. Kubernetes, originally developed at Google and now run by the [Cloud Native Computing Foundation](#), has become the dominant orchestration engine for containers automating the provisioning and management of microservices-based applications in pods of containers across a cluster of X86 server nodes.

As users deployed more complex and data-intensive applications in containers, the container community added Persistent Volumes to Docker and Kubernetes to provide storage for these applications. Kubernetes supports a wide range of file, block, and cloud storage providers for Persistent Volumes using the CSI (Container Storage Interface). In addition to Kubernetes, VAST users running Apache Mesos and Cloud Foundry can also take advantage of storage automation via CSI.

VAST's CSI driver provides an interface between the Kubernetes cluster control plane and a Universal Storage cluster. The VAST Storage Class presents the This allows the Kubernetes cluster to provision folders in an NFS Export as Permanent Volumes, define the volume's size with a quota, and publish the volume to the appropriate pods.

Following the open and open-source, philosophy behind CSI, the VAST CSI driver is open source and available to anyone interested at <https://hub.docker.com/r/vastdataorg/csi>.

VAST's CSI driver provides a standardized interface a Kubernetes cluster can use to provision Persistent Volumes, in the form of mountable folders, for pods of containers.

VAST's Manilla Plug-in provides a tighter connection between a VAST cluster and the OpenStack open-source cloud platform. In addition to the basics of publishing volumes for VMs, as CSI does for containers, Manilla also automates the creation of NFS Exports and setting the Export's access list of IP Addresses.

Manilla allows large operators to automate the process of provisioning Kubernetes, or OpenShift clusters, creating Exports private to each cluster, via OpenStack while provisioning volumes to container pods via CSI.

As Universal Storage releases include features that align with the functions defined by these platforms, VAST will continue to update these platform interfaces. VAST support for additional platforms will be based on customer demand.

Encryption at Rest

When encryption at rest, a system-wide option, is enabled, VAST systems encrypt all data using FIPS 140-2 validated libraries as it is written to the Storage Class Memory and hyperscale SSDs.

Even though Intel's AES-NI accelerates AES processing in microcode, encryption and decryption still require a significant amount of CPU horsepower. Conventional storage architectures, like one scale-out file system that always has 15 SSDs per node, can only scale capacity and compute power together. This leaves them without enough CPU power to both encrypt data and deliver their rated performance.

Conventional storage vendors resolve this by using self-encrypting SSDs. Self-Encrypting Drives (SEDs) offload the encryption from the storage controller's CPU to the SSD controller's CPU, but that offload literally comes at a price, that is the premium price SSD vendors charge for enterprise SEDs.

To ensure that we can always use the lowest cost SSDs available VAST systems encrypt data on the hyperscale SSDs in software, avoiding the cost premium and limited selection of self-encrypting SSDs. VAST's DASE architecture allows users to simply add more computing power, in the form of additional VAST Servers, to accommodate the additional compute load encryption may present.

Rather than being a performance limitation, or significant additional cost, VAST encryption becomes just one more factor in balancing the performance provided by VAST Server CPUs and the capacity of VAST enclosures.

VASTOS Cluster Management

Just as Universal Storage was designed to redefine flash economics, VAST Clusters have been equally

designed to minimize the cost of scale-out system operation by simplifying the system's full lifecycle of operation – ranging from day-to-day management tasks such as creating NFS Exports and quotas... all the way to performing automated, non-destructive updates and expanding the cluster online.

Quotas

VAST Universal Storage systems allow administrators to create absolute or user/group quotas for any folder/bucket in the VAST namespace. Absolute quotas trigger when the total amount of logical (unreduced) data in the folder, and its descendants, reaches the quota amount. User quotas trigger when the total size of the files owned by the user in the quota folder and its descendants.

When setting a quota administrators can define the quota as soft or hard:

- Hard quota, the system will generate an alert and block writes to the folder when the total amount of data in the folder reaches this value.
- Soft quotas will generate an alert when the total data in the folder reaches this value, and a grace period will define the amount of time the system will allow a quota overage before the system blocks writes to a folder.

Authentication

VAST Clusters can authenticate users, and groups, using NIS, LDAP and/or Active Directory including support for RFC2307 and RFC2307bis for Active Directory/LDAP to UID/GID mapping.

To support users who are members of more than the 16 groups that NFS clients support, VAST systems query the LDAP/Active directory service directly for group memberships including RFC2307bis nested groups.

The VASTOS Management Service

The VASTOS Management Service (also known as VMS) is responsible for all the system and cluster management operations across a VAST Cluster. VMS is a highly-available service that runs at any time on one of the Docker containers in a VAST Cluster. VMS functions include:

- Exposing the VAST Cluster's RESTful API
- Serving the cluster GUI and CLI
- Collecting metrics from the servers and enclosures in the cluster
 - Reporting metrics to user
 - Reporting metrics to VAST Insight
- Automating and managing cluster functions
 - Software updates
 - System expansion

VMS runs in a container that is independent from the container that runs the VAST Server and Protocol Manager processes. The VMS container only runs management processes that take instructions from the users, and exchanges data with the VAST Servers and Enclosures in the cluster. All VMS operations are out-of-band from the VAST Protocol Manager as to ensure consistent I/O performance.

While VMS today runs on a single server, it is also designed to be highly-available. Should a server running VMS go off-line, the surviving servers will detect that it isn't responding and first verify it is really dead before holding an election to assign a new host for the VMS container – at which point the new host will then spawn a new VMS process. Since all the management service state, just like Element Store state, is stored in the persistent Storage Class Memory, VMS takes over right where it left off when it ran on the failed host.

VMS polls all the systems in VAST Cluster every 10 seconds, collecting hundreds of performance, capacity, and system health metrics at each interval. These metrics are stored in the VMS database where they're used as the source for the VAST GUI's dashboard and other in-depth analytics which we'll examine in more detail below.

The system allows an administrator to examine analytics data over the past year without consuming an unreasonable amount of space, by consolidating samples as they age reducing granularity from a sample every 10 seconds to a sample once an hour for data over 30 days old.

VAST Insight (VAST's Remote Call Home Service)

In addition to saving those hundreds of different types of system metrics to their local databases, VAST clusters also send encrypted and anonymized analytics data to VAST Insight, a proactive remote monitoring service managed by the VAST Customer Support team. VAST Insight is available on an opt-in basis, customers with strong security mandates are not required to run Insight to receive VAST support.

VAST's support and engineering teams use this platform to analyze and support many aspects of its global installed base, including:

- System health – in many cases the VAST Support team will know of system events before customer administrators do
- Monitor performance to ensure that VAST systems deliver the appropriate levels of service quality and expected throughput, IOPS, latency
- Publish non-disruptive SW updates to VAST's global installed base
- Track the benefits of Similarity-Based Data Reduction across different customers to develop trend lines around use cases
- and more.

Insight also provides VAST's R&D team invaluable insight into how customers actually use Universal Storage Clusters, so we can concentrate our engineering toward efforts that will have the greatest positive impact on customers.

Non-Disruptive Cluster Upgrades

Too many storage technologies today still have to be shut down to update their software. As a result, storage administrators (who perform these outages during off-hours and weekends) will optimize their work/life schedule by architecting storage estates around the downtime events that their systems impose upon them. The result is the use of many, smaller systems, to limit the scale of the outages they have to endure. Another way administrators avoid the perils of upgrade-driven storage downtime is to delay performing system updates, which has the unfortunate side-effect of potentially exposing systems to vulnerabilities and fixes that their vendors have solved for in more current branches of code.

VAST Data believes that disruptive updates are antithetical to the whole concept of Universal Storage. A multi-purpose, highly-scalable storage system has to be universally and continually available through routine maintenance events such as system updates.

The Cluster upgrade process is completely automated... when a user specifies the update package to be installed, VMS does the rest. The statelessness of the VAST Servers also plays an outsized role in making cluster updates simple – as the state of the system does not need to be taken offline in order to update any one computer. To perform the upgrade, VMS selects a VAST Server in the cluster, fails the Server's VIP (Virtual IP Addresses) over to other VAST Servers in the cluster and then updates the VAST Server container and the VASTOS Linux image (in the case of an OS update) on the host. VMS then repeats this process, transferring VIPs back to the updated servers as they reboot until all the VAST Servers in the cluster are updated.

Updating the VAST Enclosure follows a similar process. VMS instructs the VAST Enclosure to reprogram an enclosure's PCIe switches to connect all the SSDs to one Fabric Module, and all the VAST Servers then connect to those SSDs through the still-online fabric module. Once the failover has completed – VMS will update the Fabric Module's software and reset the enclosure to get it back online in an HA pair.

Expansion

The ability to scale storage performance, via VAST Servers, independently from Enclosure capacity is one of the key advantages of the DASE architecture. Users can add VAST Servers and/or VAST Enclosures to a cluster at any time.

As we saw in the [Asymmetric Scaling section](#), VAST Servers and Enclosures can be composed of heterogeneous infrastructure as a system scales and evolves over time. VAST Servers using different generations (different generations or makes of CPUs, different core counts) with different VAST Enclosures (differing numbers and sizes of SSDs) can all be members of the same cluster without imposing any boundaries around datasets or introducing performance variance.

When VAST Servers are added to a cluster or Server Pool, they're assigned a subset of the cluster's VIPs and

immediately start processing requests from the clients along those VIPs, thus boosting system performance. Users can orchestrate the process of adding VAST Servers to their cluster to accommodate expected short-term demand, such as during a periodic data load and releasing the hosts to some other use at the end of that peak demand period thanks to the containerized packaging of VAST software.

When enclosures are added to a VAST Cluster, the system immediately starts using the new Storage Class Memory and hyperscale flash to store new user data, and metadata, providing a linear boost to the cluster's performance.

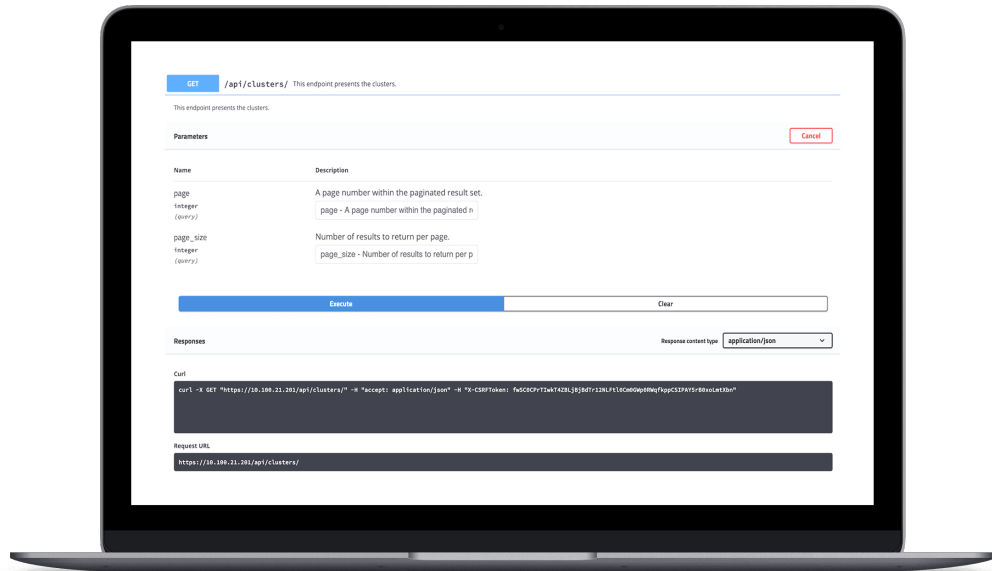
I/O and pre-existing data are rebalanced across the expanded capacity of the system:

- Writes are immediately spread across the entirety of the expanded pool of Storage Class Memory, so write performance automatically scales with the new resource
- Newly written data will be striped more widely across the expanded pool of hyperscale flash because of VAST's write-in-free-space approach to data placement, making subsequent reads to this data equally well-balanced
- Pre-existing data will remain in the SSDs it was originally written to and will, over time, be progressively restriped when the system performs routine garbage collection. VAST Systems don't aggressively re-balance data at rest on the senior enclosures because that would cause write-amplification and impact performance without any real benefit. Data on the existing enclosures is already wide striped across at least 44, and most commonly 100s of SSDs there's limited advantage to striping data even wider.

API-First Design

Twenty-first century data centers should be managed not by a high priesthood of CLI bashers who are charged with maintaining farms of data storage silos, but by orchestration platforms that manage individual devices not through a CLI but through more structured, consistent, and accessible application program interfaces (APIs).

VMS is built with an API-first design ethos. All VMS management functions on the Cluster, from creating NFS exports to expanding the cluster are exposed through a RESTful API. To take the complexity out of learning how to work with APIs and writing according to various programming languages, the VMS publishes APIs via Swagger. For the uninitiated, Swagger is an open-source API-abstraction that publishes easy tools to build, document and consume RESTful web services.



eg: a Swagger 'GET Cluster's API call: with auto-generated CURL commands and request URLs

While the VMS also provides a GUI (details below) and a traditional CLI, VMS's API-first design means the GUI and CLI consume the RESTful API rather than controlling the system directly. This approach ensures that API calls are sufficiently tested and that all system functions will always be available through the RESTful API. Systems with CLI- or GUI-first design philosophies can often treat their RESTful APIs as second class citizens.

A Modern GUI

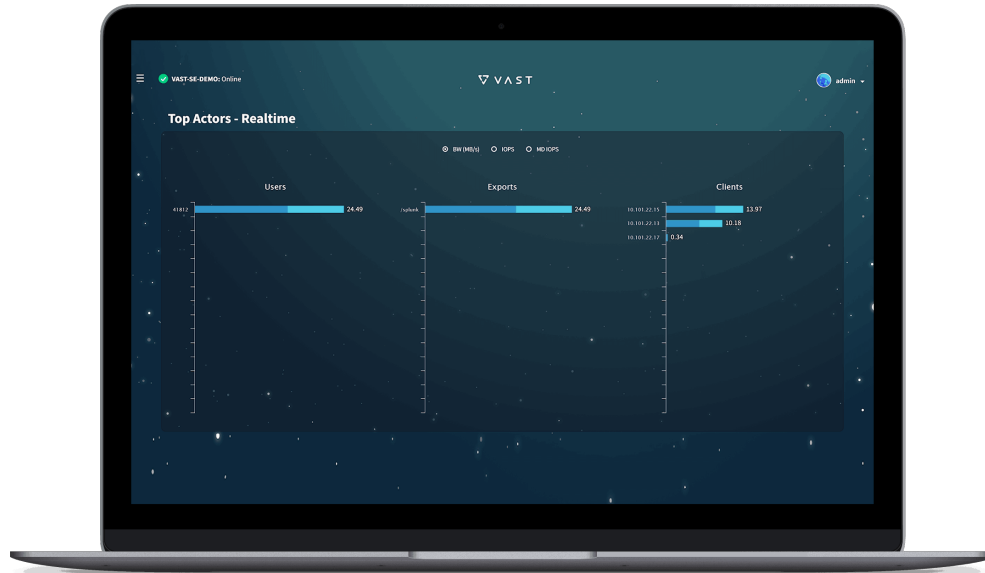
While a RESTful API simplifies automating common tasks, GUIs remain the management interface of choice for customers who want a quick and simple view of system operations.

A good dashboard allows mere mortals to quickly understand the health of their storage system in seconds, and provides a full-featured GUI that makes it easy to perform simple tasks while also allowing more curious parties to drill down into the data that feeds that dashboard.



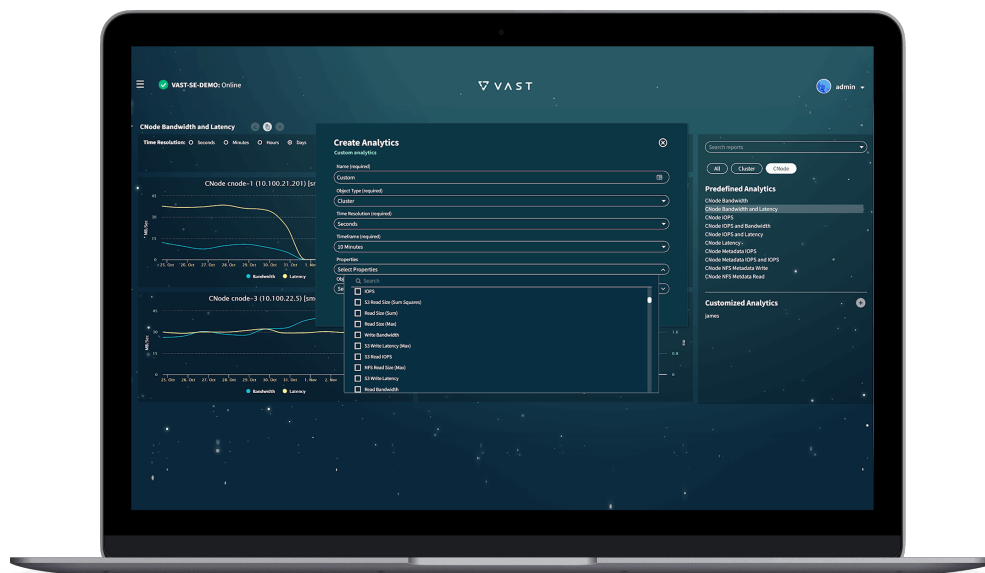
The VAST web GUI is implemented entirely in HTML5 and does not require Java, Adobe flash, browser plug-ins or any other client software. Administrators can manage their VAST Clusters from any modern browser.

The VAST GUI's main dashboard provides a system's administrator or another user who has been given RBAC permissions, a quick snapshot of the system's health. Detailed analytics are also provided to understand what's happening both at the system-level and at the application level.



VMS's User Activity screen allows administrators to reverse-engineer application performance issues by helping them understand the users, exports and clients that are interacting with the system, and the levels of traffic they're creating. User Activity provides the ability to monitor any of the users of the system in real time, not just the top 10 "bad actors".

For historical data analysis, VAST Analytics dashboard provides administrators with a rich set of metrics to monitor everything from latency, IOPS, throughput, capacity and more – all the way to the component level.



Creating a custom dashboard















Administrators can even create and save customized dashboards by combining any number of metrics they find useful in order to determine event correlations.

Role-Based Access Controls

Just as large storage systems need quotas to manage capacity across a large number of users, Universal Storage features role-based access control (RBAC) to enable multiple system tenants to work without gaining access to information that is beyond their purview.

A global administrator can assign read, edit, create and delete permissions to multiple areas of VAST system management, and establish customized sets of permissions as pre-defined roles that can be applied to classes of resources and users.

Add Role:

Realm	Create	View	Edit	Delete
Events	+			×
Hardware	+			×
Logical	+			×
Monitoring	+			×
Security	+			×
Settings	+			×
Support	+			×

Options for defining RBAC controls in a VAST Cluster

Longevity

The last step in improving the economics of an all flash storage system is extending its lifecycle, Storage systems have historically been designed for a 3-5 year lifecycle, storage vendors enforce this by inflating the cost of support in later years – making it more economical to replace a storage system than renew its support contract. These same vendors then get drunk on the refresh cycle sale and their investors are conditioned on the 3-5 year revenue cycle that comes from this refresh selling motion.

VAST's hardware appliances are designed for a 10-year deployment life. As long as an appliance is covered by a Gemini subscription VAST will provide hardware support and failed component replacement for up to 10 years from the appliance's initial installation. While Gemini subscription prices will be reset periodically for competitive reasons VAST customers will never be charged more for their subscription over time.

Customers can replace appliances, when the density, performance and power consumption advantages of new appliances dictate and transfer any remaining subscription time to the new appliances. Since customers buy hardware directly from VAST's manufacturing partner at cost there's no pressure for VAST to encourage replacement hardware purchases and the software re-buys that go with them.

While VAST Servers and VAST Enclosures have a 10-year lifetime, a VAST cluster is immortal for as long as a user wants. New Servers and Enclosures are added to clusters as performance and/or capacity is needed, and older models are aged out as their utility diminishes. While the hardware evolves over time, the cluster lives through a progressive number of expansion and decommission events without ever requiring users to forklift upgrade anything, ever.

CONCLUSIONS

Universal Storage frees users from the tyranny of storage tiers with one, single storage solution that provides the best of the capabilities of each of the legacy tiers of storage, and virtually none of the compromises:

- The performance of an all-flash storage system
- The convenience of a shared file system
- The scalability of an object store

... all at a cost that is affordable for all of an organization's applications.

By replacing this multitude of storage tiers and bringing an end to the HDD era, Universal Storage eliminates the cost and effort of constantly migrating data between tiers and the waste of managing data across multiple silos of infrastructure across each tier. Even more significantly, Universal Storage helps realize the promise of the true all-flash data center so organizations can now analyze, and extract business value from data they otherwise would have relegated to slow, archive storage that cannot support the needs of new data-intense applications.

DASE: The Universal Storage Architecture

To build Universal Storage and bring down the cost of all-flash storage VAST Data had to build a new scale-out storage architecture DASE (Disaggregated Shared Everything). DASE in turn is empowered by three new technologies:

- hyperscale flash memory SSDs which bring down the cost of flash
- Storage Class Memory, which provides high performance, high-endurance memory
- NVMe over fabrics which connects the stateless VAST Servers to the QLC flash and Storage Class Memory with just a few microseconds of latency

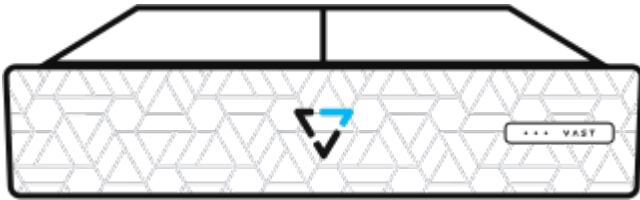
DASE disaggregates the storage media, Storage Class Memory and QLC flash SSDs in highly-available enclosures, from the VAST Servers, stateless software controllers that run in containers on commodity x86 servers and appliances.

DASE provides several advantages over conventional storage, including other scale-out systems:

- Independent scale of capacity and system compute
 - From petabytes to exabytes
- Highly efficient erasure coding
 - N+4 protection
 - As little as 3% data protection overhead
 - Fast, low-overhead rebuilds
- Highly-effective data reduction
 - Guaranteed better reduction than compression and deduplication alone
- 10-year longevity via global flash management

Hardware

NVMe Enclosure



I/O Modules	2 x Active/Active IO Modules
I/O Connectivity	4 x 100Gb Ethernet or 4 x 100Gb InfiniBand
Management (optional)	4 x 1GbE
Hyperscale Flash Storage	DF-5615: 44 x 15.36TB DF-5630: 44 x 30.72 TB
NVMe Persistent Memory	12 x 1.5TB U.2 Devices
Dimensions (without cable mgmt.)	2U Rackmount: H: 3.2", W: 17.6", D: 37.4"
Weight	85 lbs
Power Supplies	4 x 1500W
Power Consumption	1200W Avg / 1450W Max
Maximum Scale	Up to 1,000 Enclosures

VAST Quad Server Chassis



Servers	4 x Stateless VAST Servers
I/O Connectivity	8 x 50 Gb Ethernet 4 x 100 Gb InfiniBand
Management (optional)	4 x 1GbE
Physical CPU Cores	80 x 2.4 GHz
Memory	32 x 32GB 2400 MHz RDIMM
Dimensions	2U Rackmount: H: 3.42", W: 17.24", D: 28.86"
Weight	78 lbs
Power Supplies	2 x 1600W
Power Consumption	750W Avg / 900W Max
Maximum Scale	Up to 10,000 Enclosures



© 2021 VAST Data, Inc. All rights reserved. All trademarks belong to their respective owners.