Quantum

Myriad All-Flash, Scale-Out File and Object Storage Software

A Technical Deep Dive Into Myriad's Modern Cloud-Native Architecture

WHITE PAPER

Contents

Introduction

Technological progress never follows a straight line. Like a good novel, it twists and turns, full of dead ends, red herrings, cliff-hangers, and colorful characters. Unlike a novel it never truly ends, but there are times when it's clear that a major chapter is coming to a close. Today, a major chapter in the story of data storage is ending, and an exciting new chapter has begun.

Non-volatile solid-state mass storage has more than 30 years of history behind it, dating to SanDisk's introduction of the flash SSD in 1991. Until recently, two characteristics of SSDs have held them back – interface technology, and relatively high cost.

SSDs evolved opportunistically, using common storage interfaces like SATA for convenience, but these interfaces were always a bottleneck. The NVMe specification finally unchained flash memory, enabling applications to access all the parallelism and performance of the underlying chips.

Meanwhile, HDD technology achieved incredible success, but the laws of physics are slowing further advancement, while SSDs are commoditizing. Storage pundits tracking $/TB curves and issuing breathless predictions of 'crossover' miss the point. It's not about price, it's about value.

The trick with SSDs has always been to find where their benefits outweigh the upcharge. In the recent past it only made sense to deploy SSDs in limited cases where they would generate net savings or provide a competitive advantage. Today, they are inexpensive enough to use to store backups(!) and their non-performance related benefits, such as low power consumption and compact size, are often enough to justify their price premium. Flash now makes sense for most active data. The HDD chapter of this story is ending.

For decades the bottleneck in every system design has been the HDD-based storage. Software didn't have to be highly optimized, it just had to run faster than the storage. NVMe and RDMA changed that. To fully take advantage of NVMe flash requires software designed for parallelism and low latency end-to-end. Simply bolting some NVMe flash into an architecture designed for spinning disks is a waste of time.

Quantum Myriad™ is cloud-native, all-flash file and object storage software designed for this new era. It takes advantage of modern software design, development, and deployment techniques to deliver the promise of NVMe flash for workloads of any size and composition, anywhere, while setting a new standard for simplicity. This paper provides a technical overview of Myriad's architecture. Turn the page to see how the story unfolds.

Architecture Overview

SOFTWARE

Myriad software is designed using cloud-native principles. A fully containerized microservices architecture orchestrated by Kubernetes provides scalability, flexibility, and ease of use. As with any software, Myriad consists of many components that interact to provide internal functions and customer-facing services, in this case the storage and retrieval of files and objects. These components and functions are most easily depicted as layers, roughly paralleling the flow of data through the system from top to bottom. Every component has been designed to minimize latency and maximize parallelism. Figure 1 is a high-level depiction of the layers in the Myriad software stack.

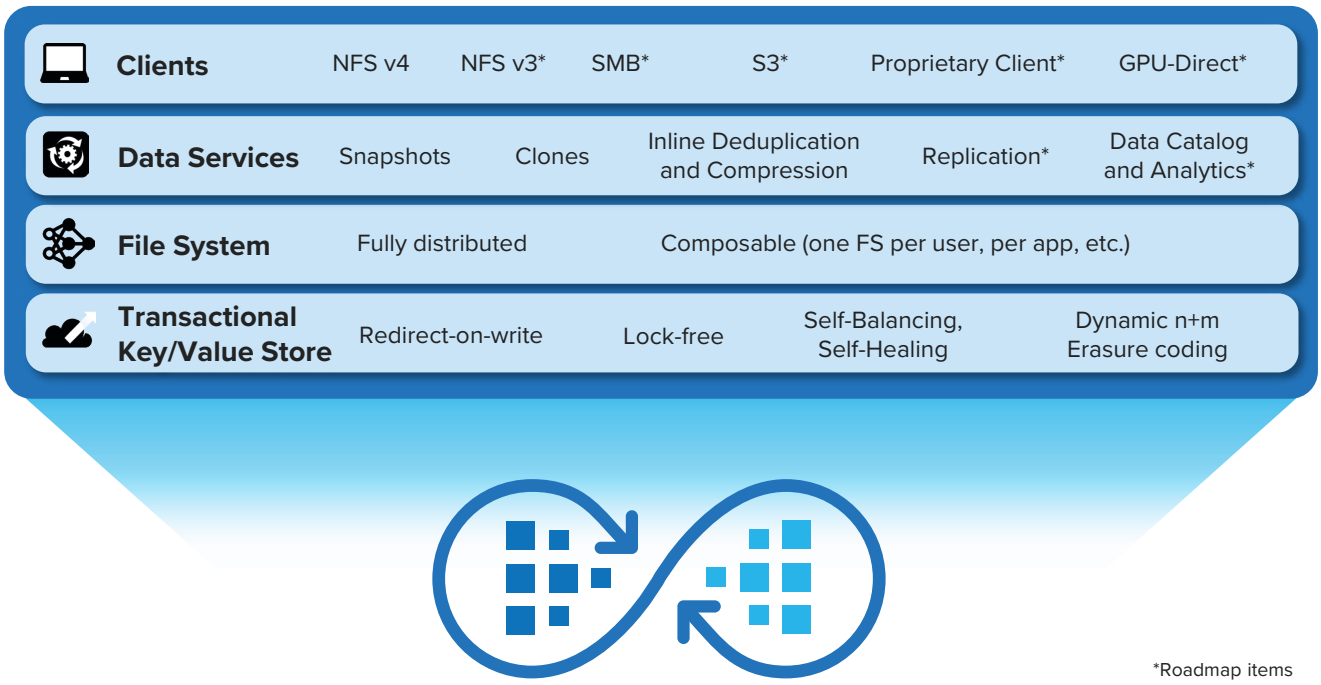


FIGURE 1 - MYRIAD HIGH-LEVEL SOFTWARE ARCHITECTURE

At the top of the stack are client presentations, the components that connect customer systems to Myriad. Initially Myriad supports NFSv4 clients, with support for many additional client types planned.

Client-facing components, such as NFS server processes, interface with an instance of the custom Myriad file system. An arbitrary number of file systems may be defined, each with one or more associated presentation layers. For example, a single file system may be defined with both NFS and S3 presentations, enabling access to the same data as both files and objects. Because there is no hard limit on the number of file systems that may be created, it's easy to provision them to meet business needs, with one or more per application, project, or even per user.

The heart of a Myriad system is the transactional key-value store, the central repository for all data and metadata. This component is distributed across all the storage nodes in a Myriad cluster, and scales linearly as new nodes are added. It's redirect-on-write nature naturally supports snapshots and rollback and being lock free it sidesteps a common point of contention in shared storage systems.

The key-value store ultimately stores data onto NVMe flash SSDs using a unique dynamic erasure coding (EC) scheme that adjusts to events in real time, maintaining data protection while maximizing efficiency.

Data services supported include always-on inline deduplication and compression, snapshots, and clones. Additional exciting and empowering services including replication and data analytics are on the roadmap.

CLUSTER ARCHITECTURE

Myriad is software, but all software needs hardware on which to run. Though it is designed to run "anywhere," including public clouds, initially Myriad is offered on Quantum-supplied appliances. No custom or exotic hardware is required, simply standard x86 servers and switches. The cluster architecture is shown in Figure 2, below.

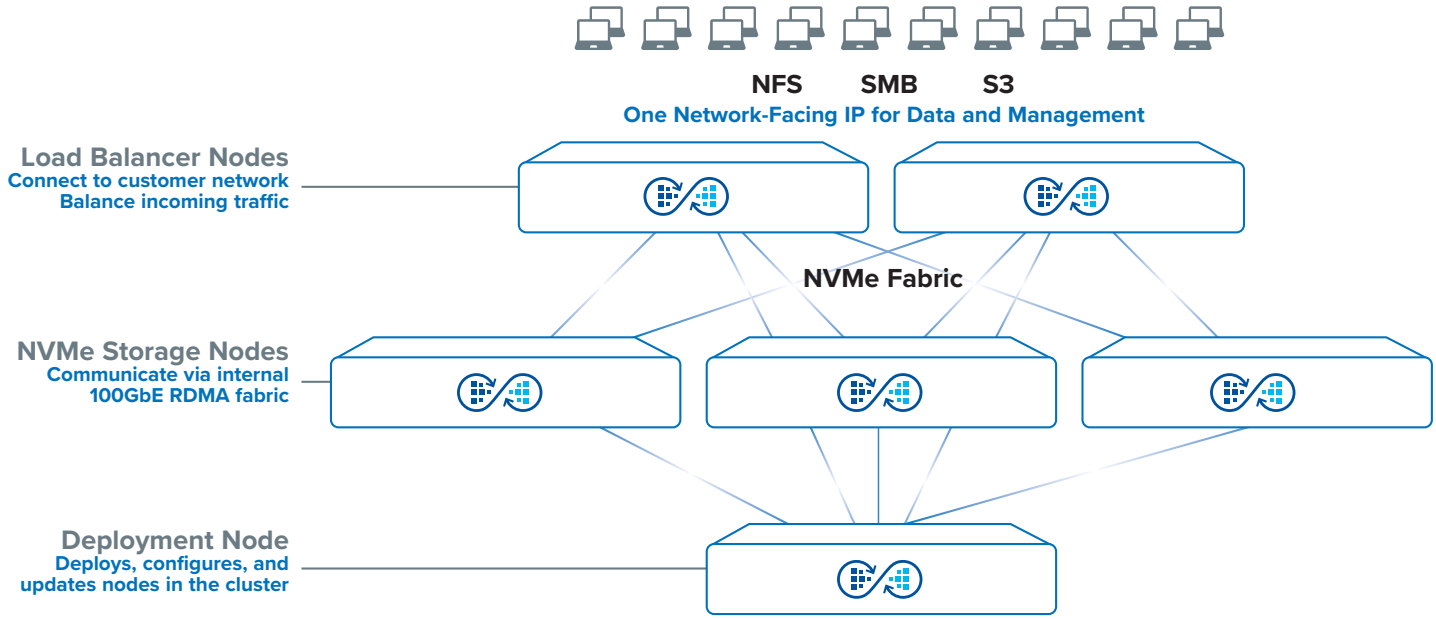


FIGURE 2 - MYRIAD CLUSTER ARCHITECTURE

At the core of the cluster are the NVMe storage nodes. These nodes house the NVMe storage devices and run Myriad software and services. The storage nodes are fronted by a pair of load balancer nodes which connect the Myriad cluster to the customer network. As indicated by their name, these balance traffic inbound to the cluster, as well as traffic within the cluster.

The deployment node is responsible for cluster deployment and maintenance, including initial installation, capacity expansion, and software upgrades.

Myriad nodes are connected internally and externally via 100Gb Ethernet networks. Legacy storage clusters require heavy customer involvement to create and maintain the related networking infrastructure. In contrast, with Myriad the equipment for cluster networking is an intrinsic part of its design and automatically configured, requiring zero additional customer investment or effort.

External networking is simple as well. A Myriad cluster of any size may be accessed and managed via a single IP address. Perhaps the best part is that in most cases storage administrators don't need to involve the network team when it's time to expand the system, providing welcome flexibility.

Configuration and control of Myriad is performed via a modern GUI, with comprehensive API support planned to enable custom automation. Traditional monitoring via SNMP is available, as well as sFlow.

Now it's time to dive into the details.

Myriad Software

MYRIAD FILE SYSTEM

A file system serves as a key intermediary between applications and storage devices. POSIX provides a common set of operations applications can use (open, close, stat, etc.) so that they don't have to care about the implementation details of the underlying file system. The Linux VFS layer enables new file systems to be 'plugged in' to the operating system in a standard way and enables multiple file system types to coexist.

Most file systems are built on volumes presented by a hardware controller or software-defined storage interface, and these volumes are often protected with RAID. Most file systems also assume that the underlying storage devices are HDDs, because for decades that's almost always been the case.

Myriad uses a unique and innovative design, because Myriad isn't a HDD + RAID-based storage system. Instead, Myriad is designed to take full advantage of the inherent characteristics of NVMe flash. It protects data using erasure coding performed by its data repository, the key-value store. Files are stored by the file system into the key-value store either whole, as single objects, or in the case of larger files, broken into several smaller objects. File system metadata items such as inodes, ACLs, extended attributes and directory entries are also stored as objects in the key-value store, as are internal system metadata values. The file system uses flags to indicate to the key-value store whether objects should be deduplicated, compressed, or both. In general, file data objects will be flagged for deduplication and compression, and metadata objects will not.

File system snapshots and clones are supported thanks to the redirect-on-write nature of the key-value store, providing protection against human error and resistance to ransomware.

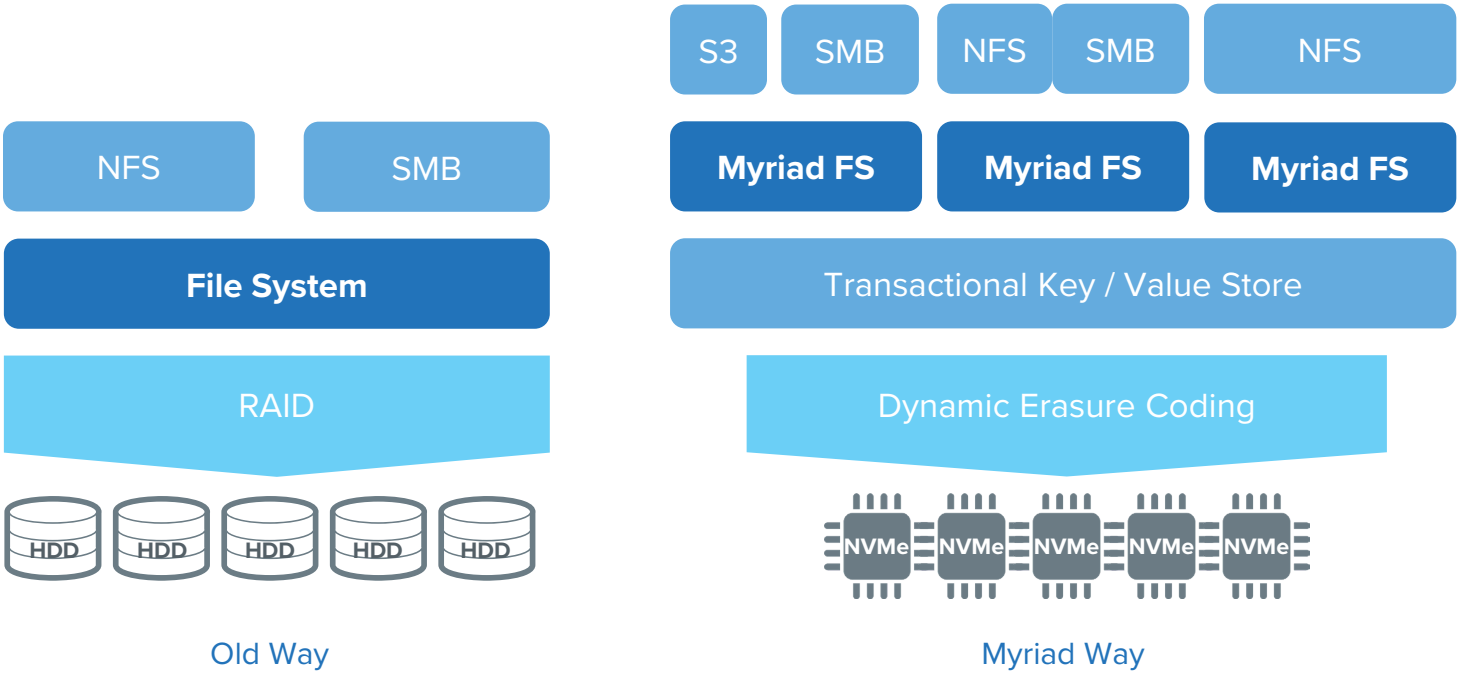


FIGURE 3 - OLD FILE SYSTEM VS. MYRIAD FILE SYSTEM

KEY-VALUE STORE

Key-value stores are not new, but they've become popular lately because they store data in an extremely efficient, highly scalable, high-performance way, without the limitations of traditional databases. Myriad uses a custom key-value store as its storage engine. This is where all of the heavy lifting happens. The key-value store is responsible for organizing, deduplicating, compressing, erasure coding, and writing data to the physical NVMe devices. It's distributed, it's transactional, it's redirect on write, it's lock free, it's efficient with small I/O sizes, and it operates with very low latency.

Distributed

As shown in Figure 2, a Myriad system stores data on a cluster of server nodes. Each of these storage nodes contains (at time of publication) ten NVMe flash drives, and they are all networked together with a 100GbE network. Every storage node has a target pod that presents its local NVMe flash drives on the network, and a source pod that enables it to access the NVMe flash drives presented by all the other storage nodes. Thanks to RDMA, every storage node has access to all NVMe drives across the system essentially as if they were local.

Incoming writes are load balanced across the storage nodes. Every node can write to all the NVMe drives, distributing EC chunks across the cluster for high resiliency.

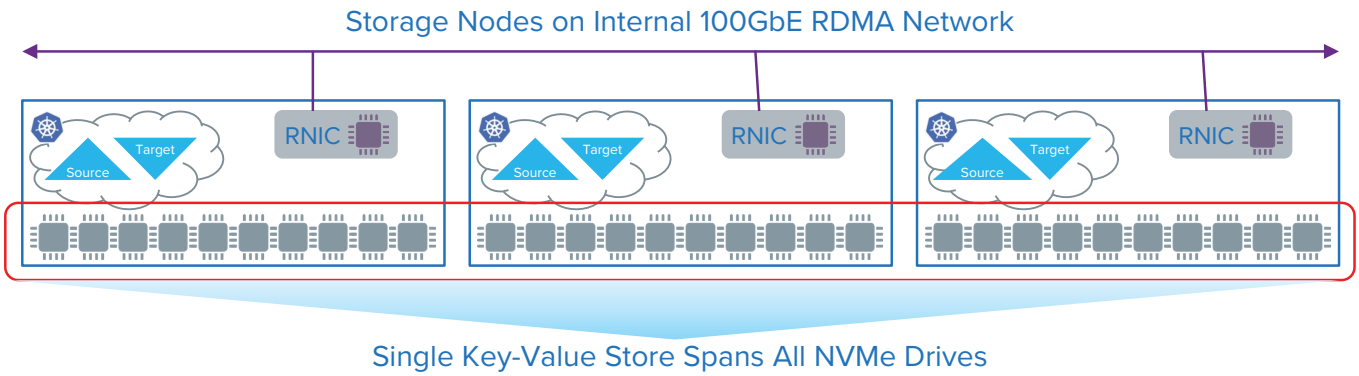


FIGURE 4 - DISTRIBUTED KEY-VALUE STORE

Transactional

Another term used to describe a key-value store, is 'object store'. When most people think of object storage, they think about AWS S3 or similar public cloud varieties. Object stores such as AWS S3 are designed so that operations are independent. They do not support transactions. Transaction support requires the ability to specify multiple changes that are committed to storage all at once, or not at all. The multiple operations must be treated in an atomic way, in that nothing can 'get in the middle.' If all the changes don't succeed, any that did are rolled back. Transactionality is common in databases and file systems, and is required to create POSIX correct behavior in a file system. Because most key-value stores aka object stores do not support transactions, it's not possible to use them as the sole basis for a POSIX file system.

A key-value store was chosen as the basis of Myriad due to the inherent scalability and high performance possible with this model, but this had to be reconciled with the fact that applications expect POSIX correct behavior from file systems. The key-value store created for Myriad elegantly merges the requirements for scalability and performance with the transaction support required to host a POSIX correct file system.

Redirect On Write

There are multiple ways a storage system may handle things when changes are made to a file or object it has already stored. One is to overwrite changes in place, leaving only the newly changed file. This works, but it's final. If that change was made accidentally by a human, or on purpose by malware, recovery means breaking out the backups. Copy on write systems first copy the about-to-be-overwritten information to a separate location, then overwrite the original blocks. Pointers are used to keep track of the original data. This approach has the great benefit that changes may be rolled back, but it also greatly increases the amount of write activity the storage must accommodate.

Redirect on write systems like Myriad simply write changes to available free space, again using pointers to keep track of older versions. As with copy on write, redirect on write inherently supports snapshots, but doesn't incur extra writes, maintaining low latency and high throughput.

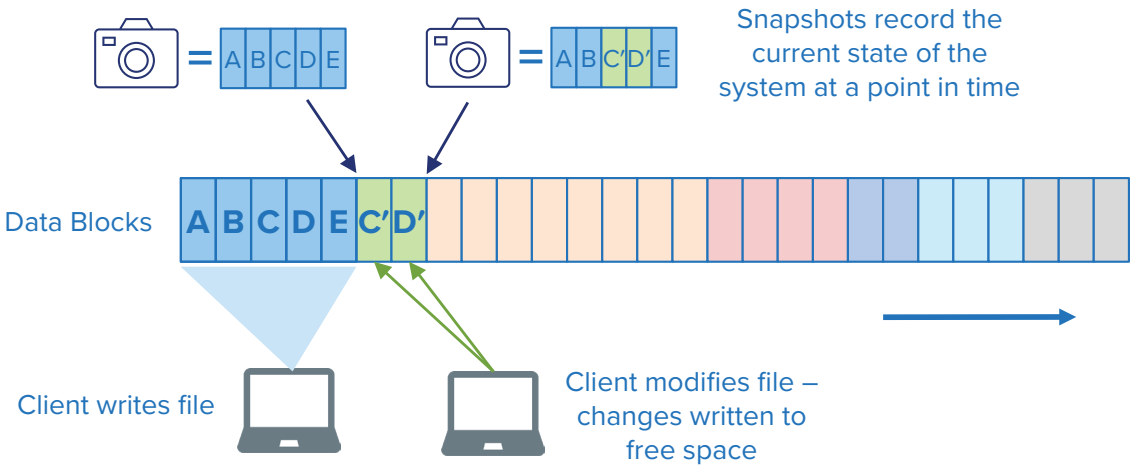


FIGURE 5 - REDIRECT ON WRITE

Lock Free

Every shared storage system needs some way to ensure that multiple actors working on the same file or region of a file cannot conflict in a way that results in data corruption. These conflicts are normally prevented proactively by using locks. Before writing, an application or process will obtain a lock on the area it intends to modify. While the lock is held, other actors are prevented from modifying the same area. Once the write is complete, the lock is relinquished. This works but puts a constant burden on the system to manage locks, and the process of obtaining, checking for, and relinquishing locks adds latency to every write transaction. It's a very heavy-handed approach considering that in the vast majority of use cases, write conflicts are actually very rare.

Using NVMe flash further reduces the chances of write conflicts compared to a HDD-based system, simply because individual writes complete so quickly. This enables Myriad to take a novel approach. Rather than trying to prevent conflicts, Myriad's design allows all actors to write freely, without locks. When a conflict occurs, the key-value store detects it and resolves it - inline. This exacts a small performance penalty for conflicting writes but captures the enormous performance benefit for all other writes. This reduces overall write latency considerably.

FIGURE 6 - LOCK FREE

DYNAMIC ERASURE CODING

Erasure coding applied to data storage is old news – even RAID uses EC – but over time EC has acquired more flexibility and visibility. When RAID happened on a hardware card, the options were often limited to “on” and “off.” Modern storage products offer knobs and dials to control EC spread and safety level. This isn’t as empowering as it might feel. Even if one chooses the “perfect” configuration on install day, as soon as the system grows that configuration is no longer optimal, and often can’t be changed.

There’s no need for EC to be complicated, or even visible. Administrators don’t need another ‘nerd knob’ to manage. They need to be assured that data is being protected from a reasonable number of simultaneous failures without unnecessarily wasting space in the process. This is what Myriad’s dynamic EC does. It provides protection from drive and node failures, while dynamically adjusting to optimize EC efficiency as nodes are added to or removed from the cluster, and as drives and nodes fail and are replaced. No drama, no ‘tuning’ needed, just automatic protection.

Myriad divides each NVMe flash drive logically into “zones.” Zones across multiple drives and nodes are grouped dynamically into “zone sets.” EC is performed at the zone set level. For example, in a system with five storage nodes (aka a “five node system”), each zone set by default will consist of one zone on one drive in each node.

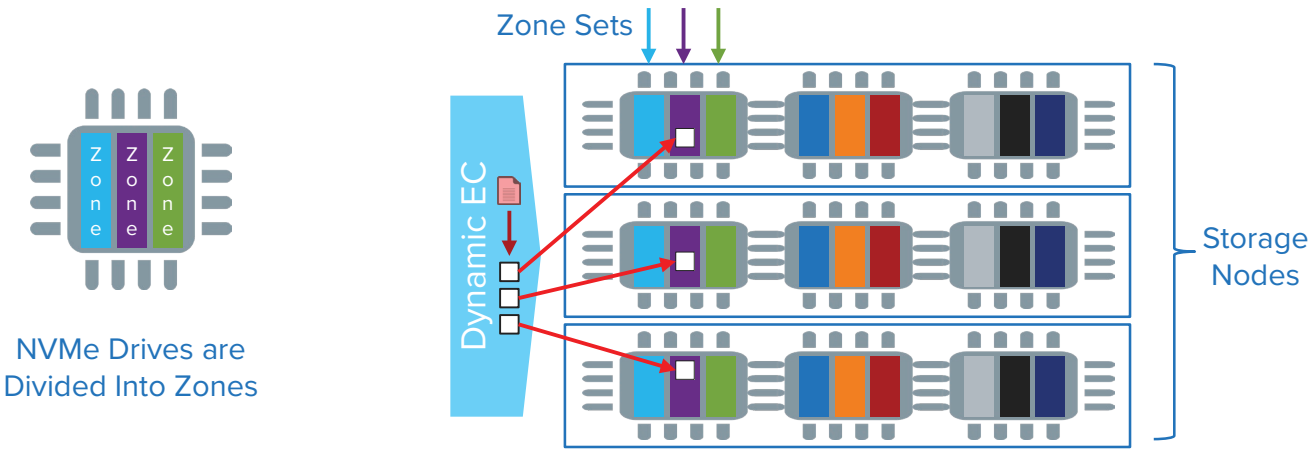


FIGURE 7 - ZONES AND ZONE SETS - 3 STORAGE NODE EXAMPLE

The default EC spread is based on the number of storage nodes in the cluster, as shown in Figure 8. For the initial release of the product, the spread will always equal the number of storage nodes, and the safety level will always be +2. The safety level will be selectable in the future. Because zone sets are spread across nodes, not just drives, this provides protection against two simultaneous node failures or the failure of two drives holding parts of the same zone set or sets.

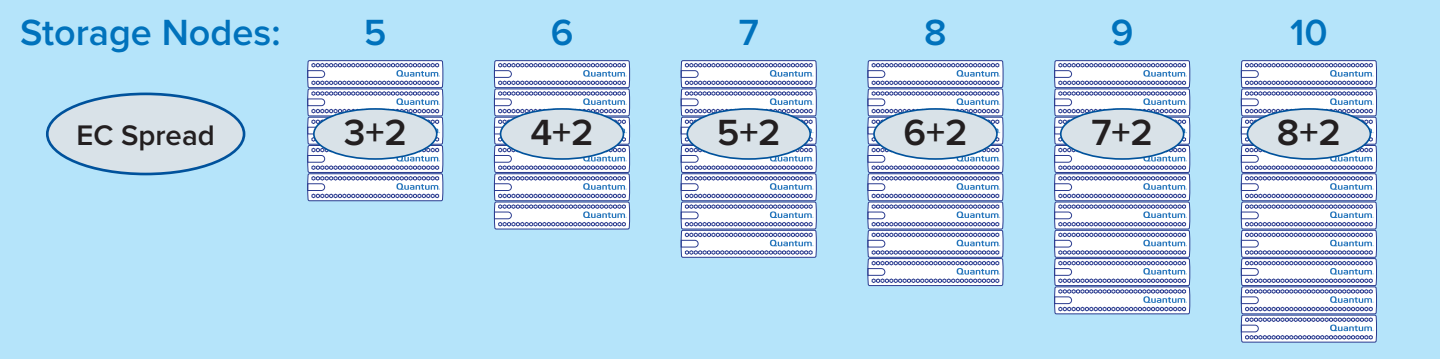


FIGURE 8 - DYNAMIC EC DEFAULT SPREADS

The Dynamic in Dynamic EC

Myriad EC becomes "dynamic" in response to changes in the system, both positive and negative. This includes expanding or shrinking the cluster, drive or node failures and corresponding repairs or replacements.

To illustrate this, consider a very simple three node system at steady state that is storing data in 1+2 zone sets, as in Figure 9. When expanded to (in this example) five nodes, the default EC scheme becomes 3+2. After the expansion, new zone sets are created at 3+2 across all five nodes, and all new incoming data is stored in these new 3+2 zone sets. The system has dynamically adjusted its EC scheme in response to the additional nodes.

But what about the existing data that has been written at 1+2? It is not left behind. As a low-priority task relative to incoming I/O, Myriad will convert existing data from the old 1+2 zone sets to new 3+2 zone sets, eventually migrating all data to the new EC scheme.

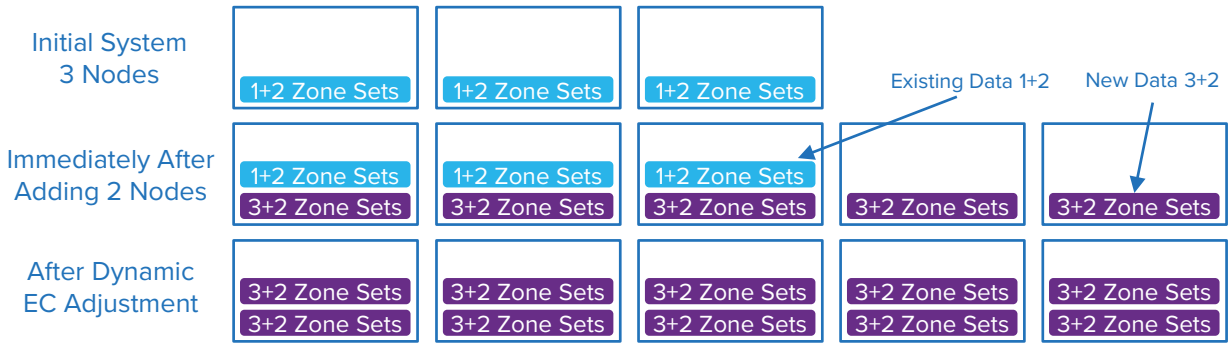


FIGURE 9 - DYNAMIC EC: SYSTEM EXPANSION

It's also possible to shrink a Myriad cluster, and the same type of dynamic adjustment to the EC scheme occurs as just described, but in reverse. If an eight node system is permanently reduced to a six-node system, the default EC scheme will drop from 6+2 to 4+2, with new data immediately written at 4+2, and the 6+2 data converted to 4+2.

A similar combination of events occurs if a node fails and is replaced. All that differs is the timing. If a node fails in a five node system, new data will be written at 2+2. Once the failed node is repaired or replaced, new data is again written at 3+2, and data that was written at 2+2 is converted to 3+2. Note that the +2 safety level is always maintained – Myriad never writes degraded stripes, such as 3+1.

Drive failures are dealt with similarly to node failures, just at a different level of granularity and timing. When a drive fails, it impacts only the zone sets in the cluster with zones on that drive. New data is written only to intact zone sets, and data on the affected zone sets is immediately reconstructed onto intact zone sets. When the failed drive is replaced, it simply becomes part of the pool of available storage.

DATA SERVICES

Myriad inherently and automatically provides some data services as it operates, others are optional. The containerized architecture is intentionally flexible, enabling additional capabilities to easily be added in the future.

Deduplication

Each data and metadata object stored by the Myriad file system into the distributed key-value store has its checksum computed. If the deduplication flag was set by the file system for the object, the key-value store compares the checksum and object length with an in-memory index. If a match is found, the new key is simply pointed to the existing value. The addition of the object length check to the basic checksum verification provides additional insurance against hash collisions.

Compression

When the key-value store receives an object with the deduplication flag set, it will attempt to deduplicate it. If the object represents new information and does not deduplicate, or the file system did not set the deduplication flag for the object, and the compression flag is set, compression is attempted. If successful, the compressed object is then stored. If the data does not compress – indicated by its size remaining unchanged or growing when compression is attempted – the native, uncompressed object is stored. Combined deduplication and compression can significantly reduce the amount of storage required for some data sets, which reduces the $/TB cost of the solution.

Replication

Though often considered just a data protection method, a flexible replication engine has many uses. Data migration, data protection, populating test/dev systems, copying data to and from analysis clusters, and moving data to and from cold archives are just a few. Today it's especially important that replication tools are able to migrate data to and from the cloud as well as between on-prem systems.

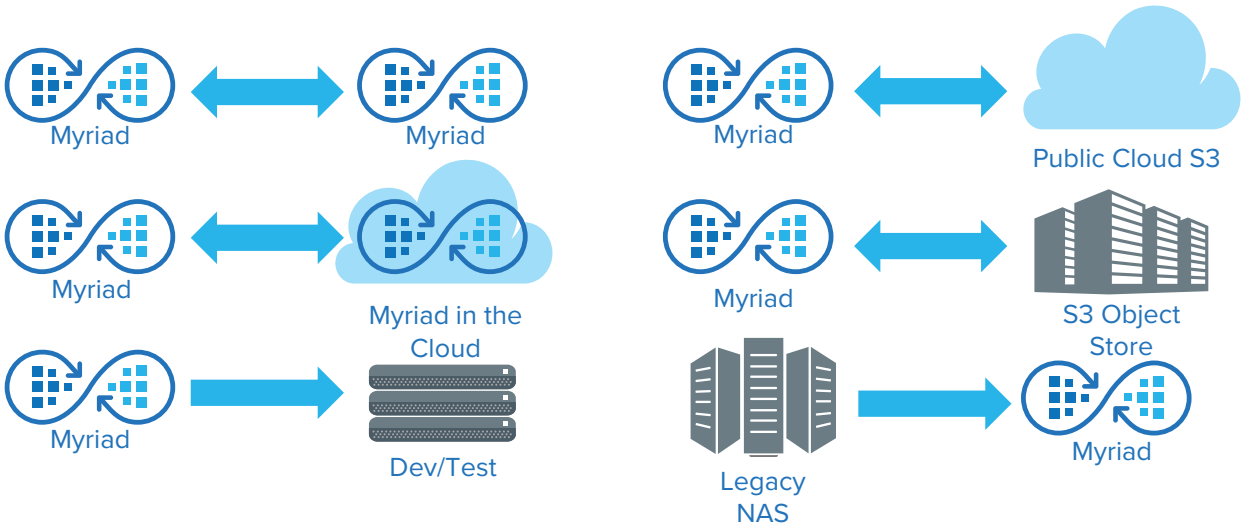


FIGURE 10 - MYRIAD REPLICATION EXAMPLES

Quantum's replication experience spans many products and many years. This experience was leveraged for Myriad, resulting in a replication capability that is flexible and easy to use, whether simply migrating content from a legacy storage system into Myriad, or replicating one local data set to an S3 target in the cloud.

Data Classification, Index, Content Intelligence

The flexibility and performance of Myriad's architecture provides the opportunity and ability to run additional data services directly on the storage cluster. Indexing and automated data classification will be offered, leveraging unique Quantum IP. Myriad's API will also allow applications deeper integration and customization.

LOW-LATENCY DATA PATH

As mentioned several times above, to fully take advantage of the performance of NVMe flash storage requires storage system software that minimizes latency wherever possible. This can require some invention, as existing processes and protocols aren't necessarily designed with latency minimization as a key design criterion. To construct Myriad's low-latency data path required thinking outside the proverbial box. Three key aspects of this data path are described below.

Lock Free

This aspect of the data path was detailed above in the Lock Free subsection of the key-value store discussion. Writers within a Myriad system, such as NFS server processes, all write simultaneously and without locks to space that has been previously allocated to them. When existing data is changed, write conflicts may occur. Write conflicts involve two or more actors attempting to change the same data in different ways. This could be conflicting edits, one actor deleting a file another is attempting to modify, and other similar occurrences.

Conflicts, which are rare, are detected by a Coordinator component within the key value store as they occur. The Coordinator will allow the first write to complete and inform conflicting writers that their changes are invalid. Those writers then must re-do their writes (after checking to ensure the proposed actions still make sense) based on the new state of the system.

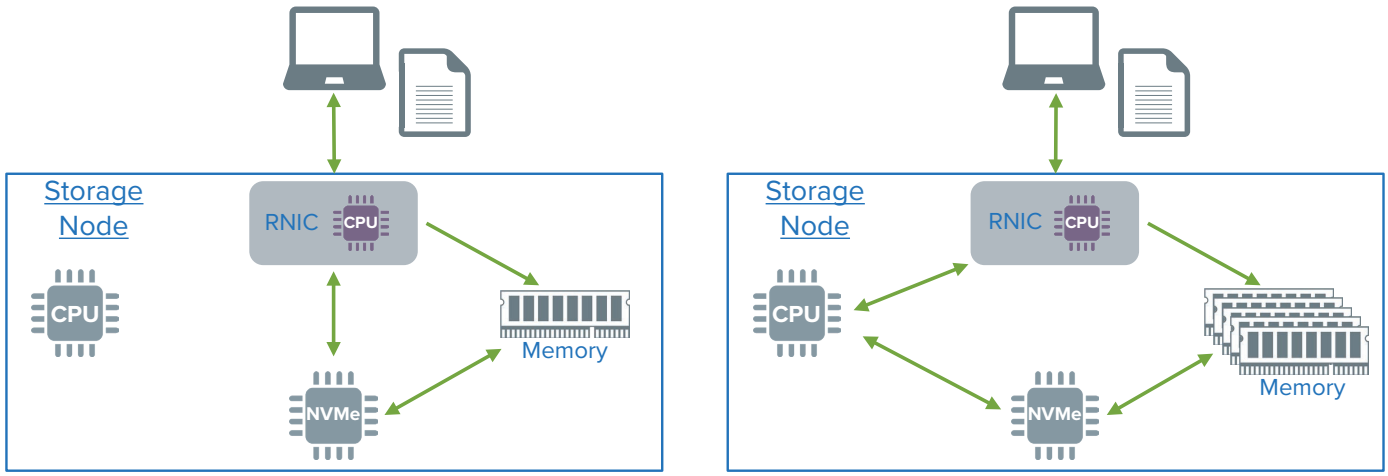
A good analogy for this process is that of a Git repository for software development. Multiple developers may “check out” the same code and independently modify it, writing their changes directly to the repository without locks and without consulting with each other. Conflicting changes are detected at “check in.”

This method works in Myriad because writes to NVMe flash complete very quickly, minimizing conflicts, and the “check out” and “check in” aspects are handled in an extremely efficient manner.

Custom NVMe Protocol

Offloading is a great technique to give system performance a boost or conserve server CPU for more important tasks. RDMA is an example of a technology that relies on offloading. Instead of the server CPU being used to complete storage networking tasks, a dedicated CPU on the network interface card (called an RNIC for RDMA-capable NIC in this case) is responsible, offloading work from the server CPU.

Offloading work to an RNIC can be useful if you need to free the server CPU to do more valuable things, but it comes with a downside - the server CPU has no visibility into the work the RNIC is doing. It’s literally “out of the loop.” The server can’t monitor or control this hidden I/O load in any way. This matters if it is important to monitor, instrument, or exert control over that I/O, as might be expected of a modern storage system.



NVMe-oF: Storage Node CPU “Out of the Loop”
Small, Transient Memory Buffer

VS.

Myriad NVMe: Storage Node CPU “In the Loop”
Large, Long-Lived Memory Buffer/Read Cache

FIGURE 11 - NVME-OF VS. MYRIAD NVME

NVMe-oF leverages RDMA offloading, with the goal of conserving server CPU cycles normally used for storage networking operations so they can be used by applications instead. That's great if the application is a database, or a machine learning process, or a business application. But what if the "application" the server is dedicated for is being a node in a networked file and object storage cluster?

With Myriad, a primary concern is minimizing latency, which means either executing fewer operations, executing them more quickly, or both. Fewer operations are what you get with tightly written software, but how does one execute them more quickly?

Consider that a 100GbE RNIC has a street price of hundreds of dollars. If one were to buy the RNIC offload CPU chip, just the chip, what would it cost? Perhaps twenty bucks? Let's be generous and say $100. Now consider that a 64-core X86 CPU has a street price of multiple thousands of dollars, and it's obvious these two chips are not the same. To get something done quickly, it helps to use the faster tool, and even a modest server CPU is orders of magnitude faster than a cheap RNIC offload CPU.

Myriad uses RDMA, but unlike NVMe-oF, the custom NVMe protocol in Myriad lets the RNIC handle the communications while leveraging the storage node CPU to do the data movement. A process continually polls the RNIC to determine when there is data to move. When new data arrives, the storage node CPU places it in a memory buffer and notifies the NVMe drive. When the write is complete the storage node CPU notifies the RNIC, which notifies the writer over the network. Write operations complete more quickly compared to NVMe-oF because the storage node CPU is so much faster than the RNIC CPU. And because the storage node CPU is integral to the data movement, it can collect stats and control the write as needed. With NVMe-oF, the small RNIC does all of this work and the storage node CPU is not involved at all.

Global Read Cache

The lock-free nature of the key-value store and Myriad's unique NVMe protocol are very effective for reducing write latency, but what about read latency? Myriad's protocol has an advantage here as well. With NVMe-oF, the memory buffer used to transfer data to and from the NVMe drives is small and transient. It exists for the duration of a data transfer and is then emptied. With the Myriad protocol, this buffer is very large – most of the PCIe4 ECC RAM in each storage node is dedicated to this buffer. It's also long-lived, only being cleared when a node is restarted. This provides an enormous, and extraordinarily fast global read cache. NVMe drives are fast, but DRAM is much faster. Serving data from a large cache with latency measured in microseconds results in reduced overall read latency for many types of workloads.

This large buffer has another advantage that's a bit more subtle. Traditionally, read caches are positioned close to the initiator side of a transaction, as the network between the initiator and the target is historically slow. But this means when many initiators are accessing the same data set, their caches fill with duplicate data. This arrangement doesn't enable the cache as a whole to scale linearly with the number of initiators, blunting the potential benefits.

Remember that in Myriad, the 'initiator' is a source pod running in Kubernetes on one of the storage nodes. The 'targets' are the target pods running on all of the other storage nodes, as shown in Figure 4. The network between the initiator and targets is not slow – it's the 100GbE RDMA cluster network. Because the initiator and targets are near each other and connected by a low-latency network, the buffer / read cache can be located on the target side without paying a large performance penalty. Each storage node caches only information residing on its own NVMe drives in this buffer. As storage nodes are added, then, the buffer size scales in a linear way, with no duplication of data in the cache.

Myriad Hardware

NODE ROLES & FUNCTIONS

As mentioned in the overview, a Myriad system is comprised of a number of different node types, including storage nodes, load balancer nodes, and deployment nodes. Each node type has a unique role to play, contributing to the storage of data, simplicity of operations, or both. All nodes use hardware commonly available from multiple suppliers, nothing custom or exotic. Initially, Myriad will be available on Quantum appliances, but options for customers to supply their own hardware are planned. Myriad software deployed in a public cloud will look a bit different as described later in this document.

Detailed specifications for current generation nodes may be found on the Myriad datasheet.

Storage Node

The main players in a Myriad appliance are the storage nodes. The storage nodes are 1U Linux servers that form a Kubernetes cluster to run the core Myriad software. Everything from the presentation layers to the erasure coding and the UI runs on the storage nodes. Future options will include larger and denser storage nodes holding more or higher-capacity drives as the market makes these available. Mixed storage node types will be supported. Myriad's architecture supports as few as three storage nodes, and as many as needed. Cluster size is not architecturally limited. Figure 12 shows an example of a 40-node Myriad cluster having 4.9PB of usable capacity (not including deduplication and compression) with today's 15.36TB drives.

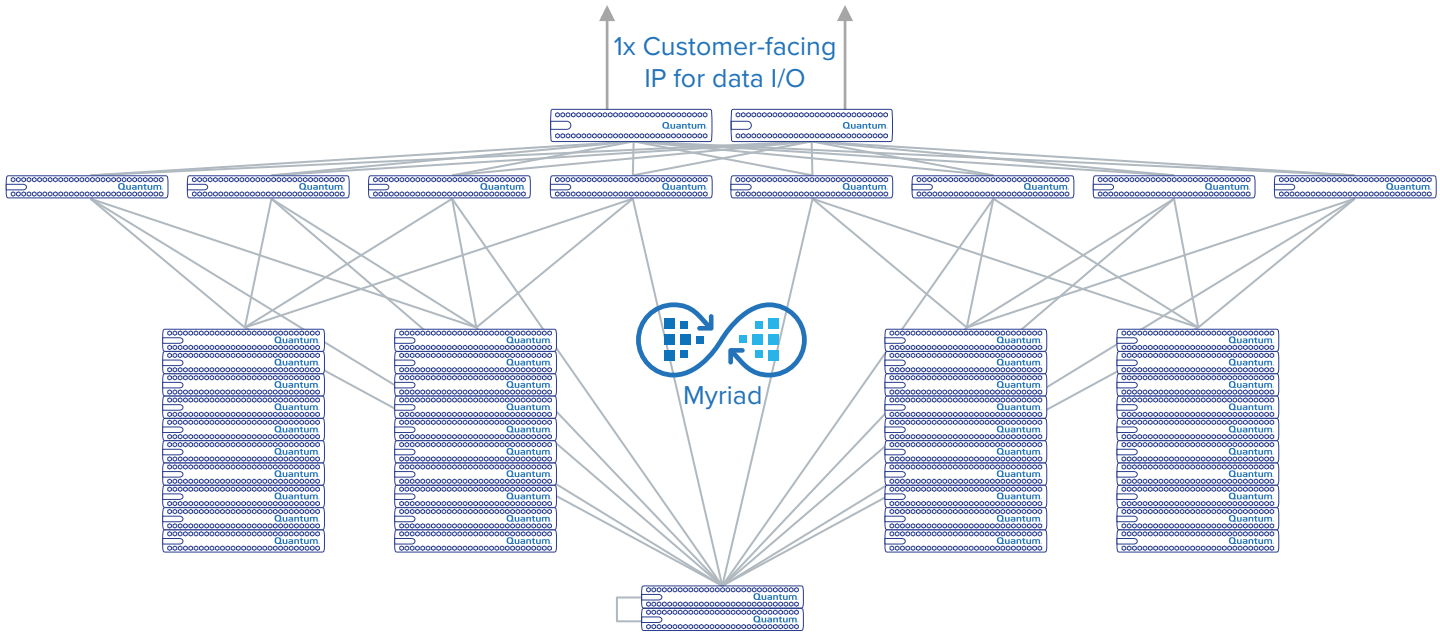


FIGURE 12 - MYRIAD CLUSTER WITH 40 STORAGE NODES

Load Balancer Node

The storage nodes need to communicate with each other and connect to the customer network. The cluster also requires a way to load balance incoming connections across all of the storage nodes. With traditional NAS clusters, these functions are the responsibility of the customer, who must manage scores of IP addresses, virtual IPs, and old-school round-robin DNS mappings. Instead, Myriad provides these functions as part of the package, in the form of load balancer nodes. These nodes are 100GbE Linux-based switches that are entirely managed by Myriad software. Some of the ports on each load balancer node are used to connect the storage nodes together, and some are used for external network connectivity. Two load balancer nodes support systems with up to ten storage nodes. Adding another pair of load balancer nodes enables scaling up to 20 storage nodes. As more storage nodes are added beyond 20, additional load balancer nodes and associated spine nodes are added as shown at the top of Figure 12. Over time, higher-density switches and faster network speeds will change the number of connections and load balancer nodes needed, but the concepts will remain the same.

Deployment Nodes

Simplicity was a key design criterion for Myriad. Simplicity in all aspects of installation, expansion, management, and use. The final node type, the deployment node, is one of the main drivers of the operational simplicity of Myriad. Similar to the load balancer nodes, deployment nodes are Linux-based switches, but only require 1GbE. The management and IPMI ports on every storage node and load balancer node are connected to the deployment node.

As the name indicates, deployment nodes are responsible for installing and configuring the other cluster nodes at the time of initial system deployment. When the cluster is expanded with additional nodes, or when failed nodes are replaced, the deployment node detects the new bare

metal nodes and leaps into action. It directs them to install their operating system, Kubernetes and Myriad software components and configures them to join the cluster. When software updates are required, the deployment node coordinates them in a rolling fashion, avoiding downtime.

Clusters with fewer than 20 storage nodes require only a single deployment node. This might appear to be a 'single point of failure', but this is not the case because the deployment node is not in the data path. Losing the deployment node does not interrupt I/O or impact data protection or performance.

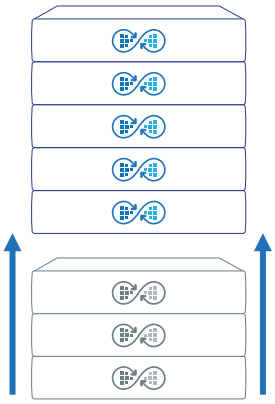
NODE EXPANSION

Myriad features "zero click" storage expansions. To add more storage capacity, more storage nodes are added. The new nodes are simply unboxed, racked, cabled, and powered on. As described in the previous section, the deployment node will detect the new nodes, ensure they are provisioned with the correct software, and join them to the cluster. There is no need to connect to a console port, fire up a UI, or even submit a ticket to the network team to get more IP addresses or reconfigure a DNS. Dynamic EC will begin immediately spreading data across all nodes, old and new. With Myriad, storage expansions are 100% automatic.

5-Step Upgrades

1. Unpack
2. Rack
3. Cable
4. Power
5. Walk Away

Zero Clicks!



Myriad Automatically...

- Detects Nodes
- Initializes Nodes
- Configures Nodes
- Adds Nodes to Cluster
- Rebalances Storage

Without User Intervention

STORAGE OF THESEUS

Storage technology changes rapidly, with storage density and performance constantly increasing, and cost per TB constantly decreasing. Even just within the realm of solid state storage, new devices and form factors frequently appear. Taking advantage of the new tech often requires buying a new platform and migrating large amounts of data, which is painful.

Myriad can't stop technology change, but it can help harness it, offering more choice in addressing changing and growing storage requirements without needing to interrupt service or perform a 'forklift upgrade'. With Myriad, as storage technology advances, older storage nodes can be replaced with newer ones. As networking speeds increase, older load balancer nodes and deployment nodes can be replaced with newer ones. Imagine multiple rounds of upgrades and changes, over time, while data stays in place and available throughout.

Like the ship of Theseus, a Myriad system may have every one of its component parts replaced over time, but it will still retain its identity and carry on its mission, it's cargo intact.

Myriad Networking

One of the areas where traditional NAS clusters dump responsibility onto the customer is networking. The NAS vendor supplies a stack of nodes and a document that lays out the requirements needed to connect them together. It's up to the customer to supply the networking equipment, figure out an addressing scheme, and deal with load balancing traffic.

In addition to the typical networking elements, Myriad uses RDMA and Kubernetes within the cluster, two advanced technologies that have additional networking requirements.

Instead of providing a kit of parts the customer must wire up themselves, Myriad is a fully integrated, self-configuring system. Internal cluster networking is automatically configured, invisible, and requires no customer management. Connecting the cluster to the customer network is surprisingly simple, even as the cluster grows and changes. Once the cluster initial configuration is complete, customers use a single IP address for both data I/O and management, no matter how large the cluster grows.

NETWORKING NAS THE OLD WAY

- Customer provides cluster networking equipment
- Customer configures cluster networking
- Dozens of front-end IP addresses required
- IP addresses manually assigned & configured
- Customer round-robin DNS for load balancing
- Requires expertise configuring RDMA networking
- Upgrades require network team involvement

NETWORKING MYRIAD THE MODERN WAY

- Cluster networking equipment included
- Cluster network automatically configured
- One ingress IP regardless of cluster size
- Fully automatic IP assignment & configuration
- Zero-management ECMP load balancing
- RDMA configuration is automatic and invisible
- Zero click upgrades w/o calling the network team

FIGURE 13 - MYRIAD NETWORKING IS SIMPLE

INTERNAL CLUSTER NETWORKING

Cluster nodes are interconnected by two networks: A 100GbE lossless network with RDMA for data and management traffic, and a 1GbE network for deployment tasks. At install the customer supplies a non-routable subnet for the internal network or accepts the default of 172.16/16. Internal interfaces for all cluster nodes and Kubernetes pods are assigned automatically from this range.

When additional nodes are added to the system, they are also assigned addresses from the pre-specified range. Expanding an old style NAS cluster required planning ahead and submitting a ticket to the network team for more IP addresses and adjustments to the round robin DNS.

Because a subnet is defined at initial install, with Myriad the storage administrator simply plugs in the new nodes and addressing is handled automatically.

To isolate the internal cluster network from the wider datacenter network, Myriad load balancer nodes run Network Address Translation (NAT). Internal source addresses and port numbers are mapped to external IPs and ports as packets head out from Myriad, and the external source addresses and port numbers of incoming packets are translated to internal IPs and ports.

Load balancer nodes use a technique known as Equal Cost Multipath routing (ECMP) to spread incoming connections across all of the storage nodes. This divides the workload across the cluster without the latency and configuration headaches of a round-robin DNS.

EXTERNAL CONNECTIVITY

Similar to the internal networking, the installation process requires the customer to specify a subnet for external cluster connectivity, but this one may be very small. For a Myriad cluster having up to ten storage nodes, only three external IP addresses are needed. One ingress IP, which is the address used for all data traffic and management, and two egress IPs for use by NAT, one for each load balancer node.

BGP

The technology that enables Myriad's external networking to be so simple is not new, it's BGP, Border Gateway Protocol. BGP is the routing protocol that runs the global Internet – it's how major ISP networks 'peer' with each other to exchange routing information. More recently, it's been adopted by organizations for routing within datacenters because of its simplicity and robustness.

BGP connects "autonomous systems," (ASs) each represented by an autonomous system number (ASN). In the case of Internet routing, the ASs are the networks of each ISP, which use unique public ASNs. Datacenter networking may use BGP in a number of configurations, with one or multiple ASs, either public or private. In BGP terms a Myriad cluster is an autonomous system of its own, with an ASN from the private range assigned by the customer.

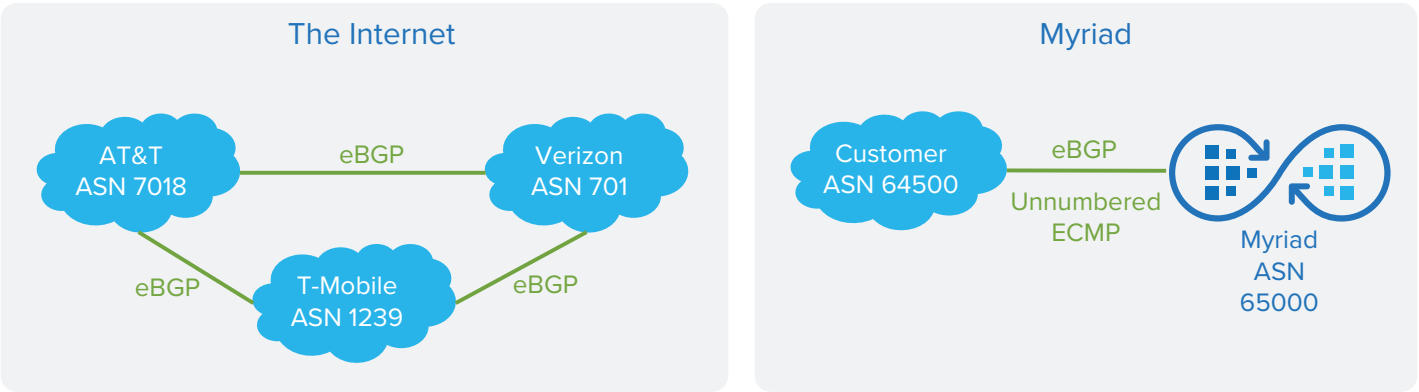


FIGURE 14 - MYRIAD BGP NETWORKING

To connect the Myriad cluster, the customer switches simply require the Myriad-facing ports to be configured for BGP unnumbered with ECMP. Using unnumbered interfaces eliminates the need to explicitly assign IP addresses, saving time and conserving addresses. ECMP provides load balancing of incoming connections across all Myriad load balancer node interfaces.

Myriad in the Cloud

Although initially offered on dedicated hardware appliances, Myriad's architecture makes it simple to deploy on public cloud infrastructure such as AWS. Its cloud-native design makes this a natural fit. Cloud deployments will take advantage of underlying cloud platform services, but the core software and experience will be the same.

Myriad software runs in a Kubernetes cluster. To enable Myriad to run on-premises on a hardware appliance, Quantum needed to develop a Kubernetes platform that could be automatically deployed and provisioned without the end user needing to know anything about Kubernetes. When running in the cloud, Myriad will instead use cloud Kubernetes services such as AWS EKS as its platform.

Similarly, Myriad appliances require load balancer nodes for connectivity and traffic balancing. Myriad in the cloud will use cloud networking and load balancing services.

Truly cloud native, Myriad is built for the cloud, in the image of the cloud, to provide the scalability, flexibility, and ease-of-use of the cloud, anywhere.

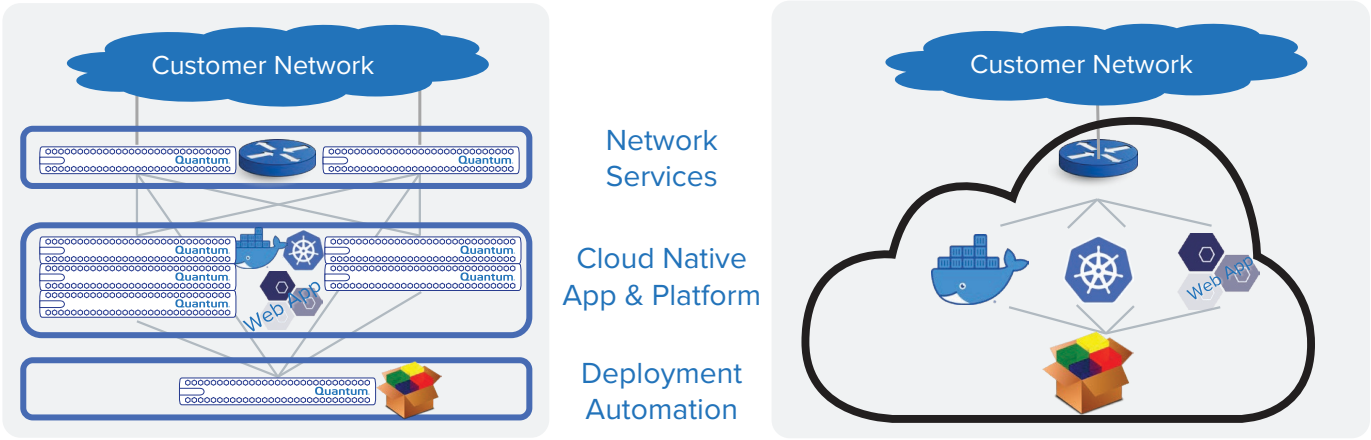


FIGURE 15 - MYRIAD IS BUILT LIKE THE CLOUD

Quantum technology, software, and services provide the solutions that today's organizations need to make video and other unstructured data smarter – so their data works for them and not the other way around. With over 40 years of innovation, Quantum's end-to-end platform is uniquely equipped to orchestrate, protect, and enrich data across its lifecycle, providing enhanced intelligence and actionable insights. Leading organizations in cloud services, entertainment, government, research, education, transportation, and enterprise IT trust Quantum to bring their data to life, because data makes life better, safer, and smarter. Quantum is listed on Nasdaq (QMCO) and the Russell 2000® Index. For more information visit www.quantum.com.

www.quantum.com | 800-677-6268

WP00277A-v01 March 2023