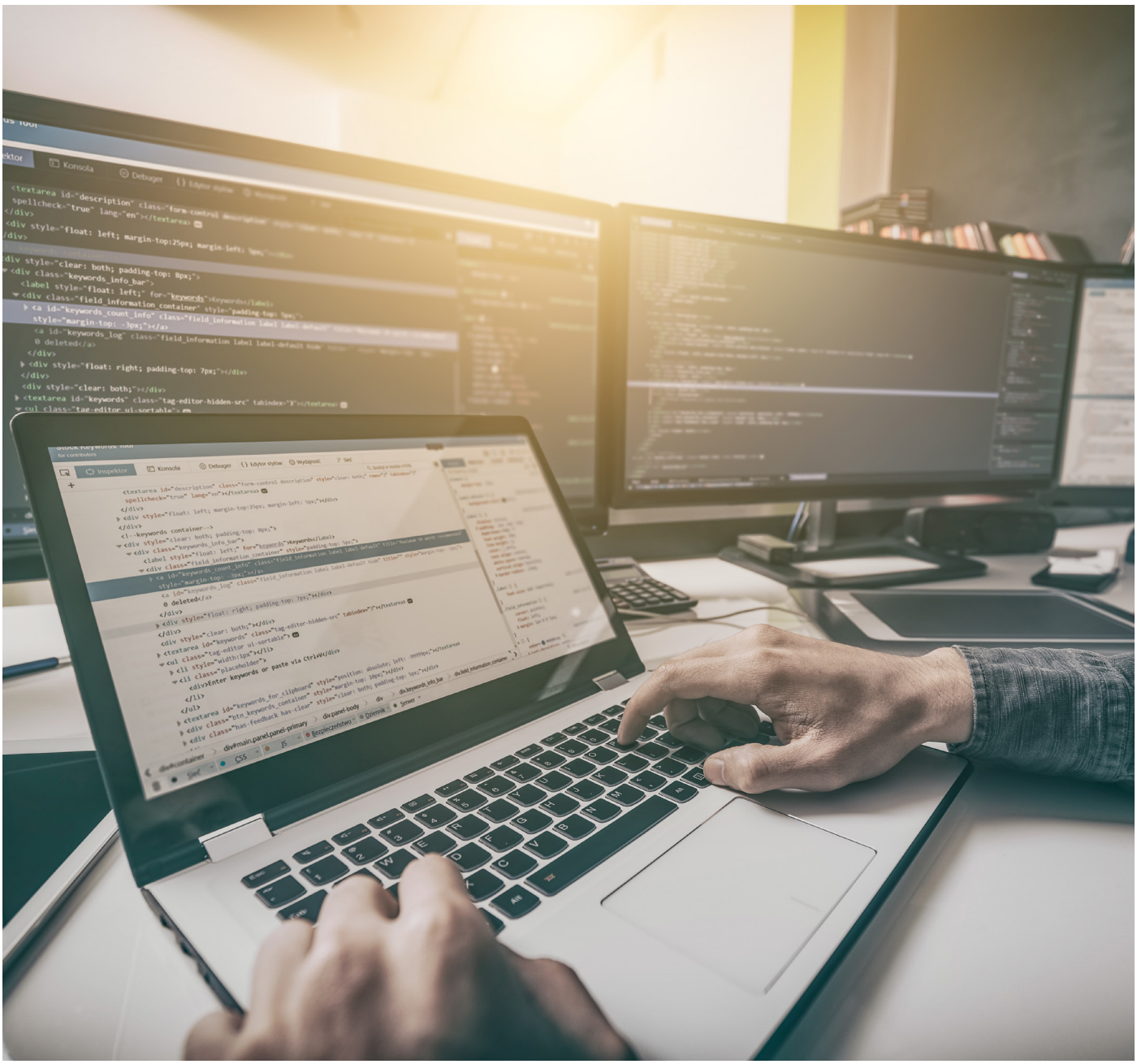
WHITE PAPER

Datrium
Always-On Erasure Coding



Contents

Introduction

Erasure Coding (EC) is a general scheme for encoding data by partitioning it into fragments augmented with parity information to enable data recovery on fragment loss. While ubiquitous in storage arrays in the form of various RAID flavors, in HCI, Erasure Coding remains an exotic optional feature applicable only to a small set of carefully selected workloads.

Enabling Erasure Coding for HCI is currently a trade-off between space efficiency and performance with the analysts and HCI vendors themselves recommending EC only for select workloads with low performance requirements and infrequent writes. This results in unnecessary wastage of storage capacity and excessive use of cooling and power in most HCI deployments.

In this paper, we show how Datrium Automatrix platform overall architectural focus on space efficiency and management simplicity turns Erasure Coding into an always-on integrated part of a system without any performance trade-offs. This results in **~3x more efficient storage capacity utilization for all workloads** and eliminates the labor intensive and error prone task of guessing workload characteristics. While DVX delivers millions of IOPS at sub-millisecond latencies and has **10x greater IOMark scores than the highest HCI result ever recorded**[1], always-on Erasure Coding also makes it cost-competitive with specialized disk backup appliances enabling primary and secondary storage convergence.

Integrating data protection and data management capabilities into Automatrix makes 3rd party backup and disaster recovery software and hardware unnecessary. Always-on capacity optimizations such as compression, deduplication and Erasure Coding are required to make software defined converged solutions cost competitive with existing optimized backup appliances. While compression and deduplication benefits depend on the underlying workload characteristics, always-on Erasure Coding delivers guaranteed workload-independent capacity savings.

Goals

1. Always-on Erasure Coding
2. Two drive failure tolerance
3. Low space overhead
4. High performance

Erasure Coding & RAID

Storage arrays and HCI systems come in all shapes and sizes with vastly different architectures tailored to a variety of use cases. One thing they all have in common is that, sooner or later, storage drives, disk or SSD, will fail. Entire drives may become unavailable or they may silently lose a few sectors to what is known as Latent Sector Errors (LSEs). LSEs and silent corruptions injected by faulty hardware or software are increasingly common despite built-in drive ECC.

A storage system must remain available and continue to serve data in the face of drive failures and LSEs. In our view, any enterprise storage system must tolerate the failures of at least two drives which will most often manifest as the failure of one entire drive plus an LSE on a second drive discovered during drive rebuild. Studies have shown that for a system that tolerates a single drive fault, the probability of data loss over 4 years is 1.95%.[2] RAID-6, an erasure code with two parity fragments to protect against two drive failures has become de rigueur for enterprise storage arrays. However, because of the difficulties tolerating two drive faults in distributed systems while meeting performance and space efficiency requirements, market leading HCI solutions still default to protection against a single drive failure. **As of 2016, 98% of customers of the leading HCI vendor used only single drive fault protection.**[3]

Storage systems rely on erasure codes to handle drive failures. Erasure Codes add redundant information to recover lost data. RAID-1 is the simplest erasure code that just replicates the data to two drives. In HCI, distributed versions of RAID-1 are known as Replication Factor 2 (RF=2)[4] and Failures To Tolerate 1 (FTT=1)[5]. Instead of replicating to two drives of the same system, HCI systems replicate to two nodes. These basic codes tolerate a single drive failure at the cost of doubling raw storage capacity to store the same amount of information.

More complex, but also more efficient Erasure Codes such as Reed-Solomon were developed to tolerate failures of more drives with less storage overhead.

1 http://www.iomark.org/content/datrium-announces-new-record-iomark-vm-hc-results
2 https://www.datrium.com/resources/single-failure-tolerance-myth-and-fact
3 http://www.yellow-bricks.com/2016/03/01/vsan-6-2-going-forward-ftt2-new-default
4 The Nutanix Bible, http://nutanixbible.com
5 VMware vSAN, Failures To Tolerate, https://storagehub.vmware.com/#!/vmware-vsan/vmware-vsan-tm-6-0-performance/failures-to-tolerate-ftt

Classic RAID 6

SNIA defines RAID-6 as any RAID system that continues to serve reads and writes in the presence of two concurrent drive failures.[6] The high-level operation of RAID 6 independent of specific codes is depicted in Figure 1.

The simplest RAID-6 implementation augments data disks with two parity disks: P and Q. During encoding, the contents of all data disks are used to calculate the contents of the parity disks. When disks fail, their contents are decoded from the surviving disks.

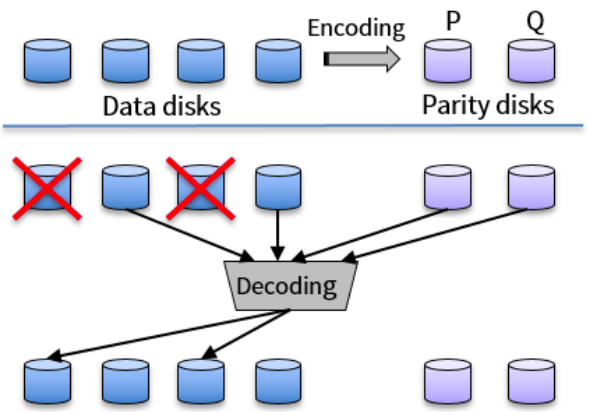


Figure 1
RAID-6 system that tolerates 2 disk failures

Disk P is encoded as XOR of all the data disks. On a single disk failure, its contents are decoded as the XOR of surviving disks.

Disk Q encoding is more complex, but it could be also done efficiently. Similarly, for two concurrent disk failures, failed disk contents are decoded based on the content of all surviving disks (including P and Q). Intel SSE/AVX instruction set extensions make computing P and Q and decoding extremely efficient.

Many flavors of RAID-6 and other more general erasure codes have been developed: Reed-Solomon, Array Codes, etc.[7] For the purposes of this paper, the details of specific erasure codes are not important. The techniques described here are general enough to apply to a vast majority of erasure codes.

The leading HCI vendors recently added distributed RAID-6 support to their product offerings. However, HCI RAID-6 remains a bolt-on non-default option applicable only to a small set of carefully selected workloads.

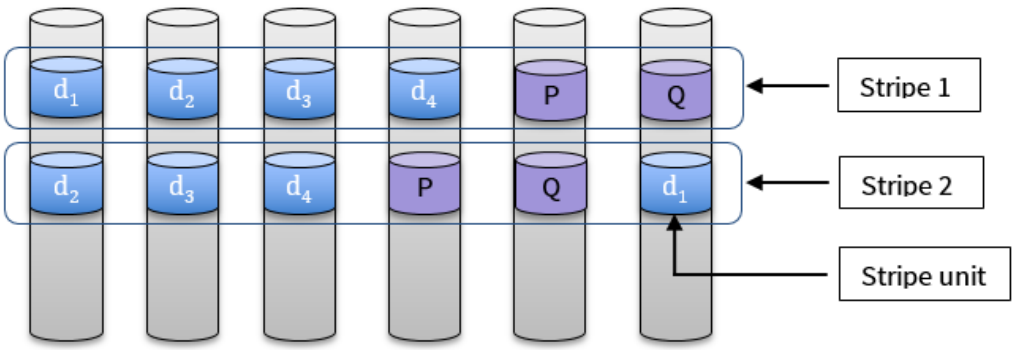


Figure 2
RAID stripes: stripe nits that encode and decode together are organized into stripes

6 https://www.snia.org/education/dictionary/r
7 James S. Plank, Erasure Codes for Storage Systems, https://www.usenix.org/system/files/login/articles/10_plank-online.pdf

In real storage systems, encoding does not apply directly to physical disks. Instead, disks are partitioned into stripe units (also known as striping units, strips or chunks) and sets of stripe units from each disk that encode and decode together are organized into stripes as depicted in Figure 2.

Treating each stripe independently for erasure coding has many advantages: it enables flexible mapping of stripes to disks for load balancing and to support a number of disks that is different from a stripe width. It also enables a flexible allocation of stripe units from each disk to stripes. Additional disks can be added seamlessly without taking a storage system offline. Writing to individual disks in relatively large stripe units improves performance of hard disks and SSDs alike.

Both storage arrays and HCI systems use stripes for their RAID implementations. While details vary, large stripes generally result in better sequential read and sequential write performance. However, unless the system is specifically designed for random I/O efficiency, large stripes can also hurt random write performance.

Stripe units are at least as large as the hard disk sector size. Writes smaller than the entire stripe size normally result in expensive read-modify-writes operations.

Read-Modify-Write Problem

Read requests that fit entirely in a stripe unit can be serviced by a single read IO to a disk hosting that unit. However, write requests smaller than an entire stripe result in multiple read and write IOs in classic RAID 6.[8]

Suppose a VM issues a small write that overwrites some contents of stripe unit d3 in Stripe 1 of Figure 3. A RAID 6 implementation cannot just overwrite the relevant part of d3 and complete the request. It must also update the contents of parity stripe units P and Q in Stripe 1 to reflect the new value in d3. Parity stripe units P and Q must be recalculated on each overwrite. For local RAID 6, this results in a number of additional read and write disk IOs. For distributed RAID 6 used in HCI, this also results in extra network transactions.

More specifically, completing a small write requires six disk IOs for classic RAID 6 as illustrated by Figure 3:

1. Read old overwritten data from stripe unit d3.
2. Read an old segment of stripe unit P.
3. Read an old segment of stripe unit Q.

Calculate new parities P and Q based on the newly written value and data read in steps 1 through 3.

4. Write new data to its target location in d3.
5. Write newly calculated segment of stripe unit P.
6. Write newly calculated segment of stripe unit Q.
7. For HCI, each of the additional IOs may be a remote network operation.

8 Peter M. Chen, et al., RAID: High-Performance, Reliable Secondary Storage, ACM Computing Surveys, Vol 26, No. 2, June 1994

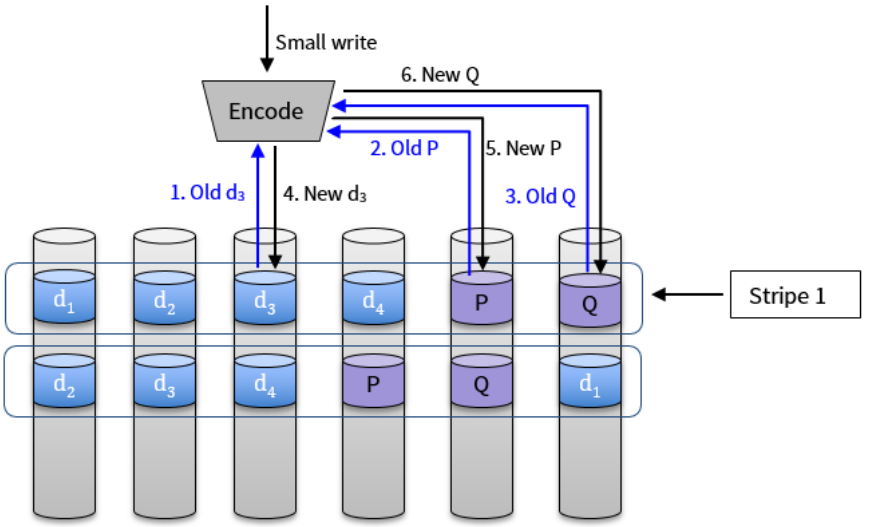


Figure 3
RAID 6: Small writes results in 6 IOs due to Read-Modify-Write

Raid Controllers

Design Goals

1. Address read-modify-write
2. Write-back caching in RAM
3. Write coalescing

Optimizing read-modify-write was a subject of active software research over the past two decades. More importantly, modern hardware RAID controllers apply aggressive forms of caching to avoid these additional IOs for local RAID improving RAID-6 performance and making RAID-6 commonplace in traditional storage appliances.

RAID controllers of conventional storage systems maintain sophisticated caches to address I/O amplification due to read-modify-write. Write-back caches are used to acknowledge the write as soon as the data is stored in high-speed protected memory without having to wait for all IOs and parity calculations to complete. The controller performs parity calculations and writes the data from the controller's write cache asynchronously hiding the read-modify-write latencies.

Subsequent read requests to the same disk location are satisfied from the cache often eliminating read IOs. Subsequent writes to the same disk location simply replace the in-memory cached content. Controllers use write-coalescing to combine small writes to adjacent logical blocks into a single large write in order to convert partial stripe writes into full stripe writes. Full stripe writes do not have read-modify-write penalty of the small writes. If all the data required for a full stripe are already cached, no extra disk reads need to be performed to calculate new parity. The RAID controller also ensures stripe consistency in the face of concurrent updates.[9]

A variety of techniques are used to guarantee data durability for write-back caches. Battery-backed caches automatically switch to a battery power source in the event of an unexpected power failure to protect cache DRAM. The cached data is written out to disk on the next power up.

Newer flash-backed write caches use flash devices to retain the cached data during power loss. Super-capacitors are employed to drain DRAM to the flash devices following a power failure.

Modern RAID controllers are sophisticated hardware systems with on-board DRAM and flash modules, FPGAs, and super-caps operated by complex firmware. They are specifically designed to address IO amplification due to read-modify-write. Hardware RAID controllers are generally reliable devices that undergo rigorous testing. However, the failure to drain write-back cache contents to disk may result in data inconsistencies. In subsequent sections we will present software file system techniques to address this problem.

While the read-modify-write problem is even more pronounced for HCI systems because of the network nature of extra IOs, unlike local RAID controllers, these systems were not designed for IO amplification avoidance leading to substantial performance penalties for RAID in HCI.

9 HP Smart Array controller technology, https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c00687518

EC in HCI Systems

Problems

1. Off by default
2. Causes IO amplification
3. Expensive post-processing
4. Requires careful analysis
5. Workload sensitive
6. Degrades performance

For years after inception, the leading HCI products supported only the most basic and space inefficient form of data protection – full mirroring.

Distributed 3-way mirroring that provides a level of data protection similar to RAID 6 is known as Replication Factor 3 (RF3) and Failures to Tolerate 2 (FTT=2). Because of its inherent space inefficiency, more recently leading HCI products were extended to also optionally support Erasure Coding in the form of distributed RAID 5 and RAID 6. Unlike conventional arrays, because of their distributed nature, HCI products do not have an option of simply using hardware RAID controllers for EC.

Nutanix was the first leading HCI supplier to offer optional Erasure Coding across nodes for increased space efficiency to loosely match the level of data protection offered by RF2 and RF3. For example, (4D+P+Q) stripe configuration reduces RF3 overhead from 200% to 50% while still protecting against two drive failures.[4]

Nutanix performs Erasure Coding via post-processing resulting in additional overhead for read and write IO after inline IO completes. To limit the impact of IO amplification due to read-modify-write, only "write-cold" data is encoded. The data extent (a 4MB logical span) is considered "write-cold" if it has not been written to in a long time. The original replicas are deleted following EC encoding.[4]

Since Erasure Coding in Nutanix implementation incurs additional overheads for overwrites, Nutanix places the burden of selecting the applicable workloads and enabling EC on the user (EC is disabled by default). The following environments are listed as not ideal for EC-X:[10]

- High amounts of re-writes or overwrites
- High number of small writes

Gartner's recent report similarly recommends avoiding EC for many important workloads classes for Nutanix:[11]

1. Use traditional three-way mirroring with RF3 when best performance, best availability and best re-protection time of data are all equally important.
2. Avoid large data ingests in combination with EC because this data will not immediately use EC capacity benefits and can fill up a node or system. Initial consumed capacity is according to mirrored protection mode storage requirements, which are 200% or 300% of the dataset.
3. Avoid EC for applications that strongly benefit from data locality, but have very cyclical batch workloads. Examples are databases with a monthly transaction update of their records, such as salary payment systems.

Other HCI vendors have followed suit and added Erasure Coding as an optional feature with narrow applicability, often limited to all-flash models because of IO amplification of distributed read-modify-write.[12] Similar to Nutanix, a careful analysis is recommended to decide whether the space savings due to Erasure Coding are worth the negative performance impact of write amplification. Enabling EC might even require upgrading network gear and storage controllers to better handle network IO amplification traffic and deeper storage IO queues.

The HCI administrator is burdened with the labor intensive and error prone guesswork of selecting the right level of data durability and capacity efficiency on a per VMDK or per datastore basis that also simultaneously takes into account dynamic workload characteristics and performance requirements.

10 https://invisibleinfra.com/2015/06/16/erasure-coding-in-nos-4-1-3/
11 Gartner, Jerry Rozeman: Key Differences Between Nutanix, SimpliVity and VxRail HCIS Appliances, Part 2: Data Protection, Capacity Optimization and Failure Analysis. Published: 26 July 2017, ID: G00334429
12 VMware vSAN 6.6 Technical Overview, July 28 2017, https://storagehub.vmware.com/t/vmware-vsan/vmware-vsan-6-6-technical-overview-4/

For this reason, Gartner general recommendation for Erasure Coding in HCI is:

Do not implement EC, deduplication and compression without understanding and analyzing the impact to system availability and reprotection time. [11]

To eliminate the labor-intense and error prone task of guess based policy management for individual workloads, Erasure Coding must have a fixed low performance overhead independent of the workload nature and it must be always on. To make this possible, Datrium fully solves the Read-Modify-Write Problem by a combination of techniques:

Always-on EC in Automatrix

Advantages

1. No write I/O amplification
2. Low space overhead
3. Survives 2 disk faults
4. No post-processing
5. Uses x86 server CPUs
6. No stroage controller bottlenecks
7. 1.8M random write IOPS

- **Full stripe writes.** Automatrix always writes full erasure-coded stripes without any partial stripe overwrites eliminating the extra read and write IOs for parity calculations.
- **Fast buffering of writes on a non-volatile medium.** To provide fast acknowledgments for small writes and improve write latencies, the incoming data is temporarily buffered on a protected durable medium.
- **Scale-out Log-structured file system (LFS).** Automatrix is based on a distributed version of LFS optimized for both capacity HDDs and SSDs. LFS works in cooperation with full stripe writes by treating sequences of stripes as a file system log.
- **Large server-side flash caches.** Automatrix engages massive server-side flash caches with always-on data reduction to fully absorb all read requests locally avoiding network congestion.

DVX tolerates concurrent failures of two disks under all conditions and for all workloads. This data protection level cannot be reduced which simplifies data management. Automatrix wide stripe geometry results in only 25% extra storage capacity overhead to protect against two concurrent disk failures. For comparison, RF3 provides a similar level of drive protection at 200% storage capacity overhead (3x space used).

Erasure Coding is always on and cannot be disabled. This is the only operating mode and the only way to persist data to durable storage. Stripes are erasure-coded inline without any expensive post-processing.

Automatrix relies on Erasure Coding during its normal operation and exposes no controls to disable EC or change the level of data protection. It does so without any performance sacrifices for workloads heavy on small random writes and/or overwrites. **DVX with built-in EC achieves 1.8M IOPS for 4K random writes for a system configured with 10 hard disk based Data Nodes** which exceeds write performance of most all flash arrays.

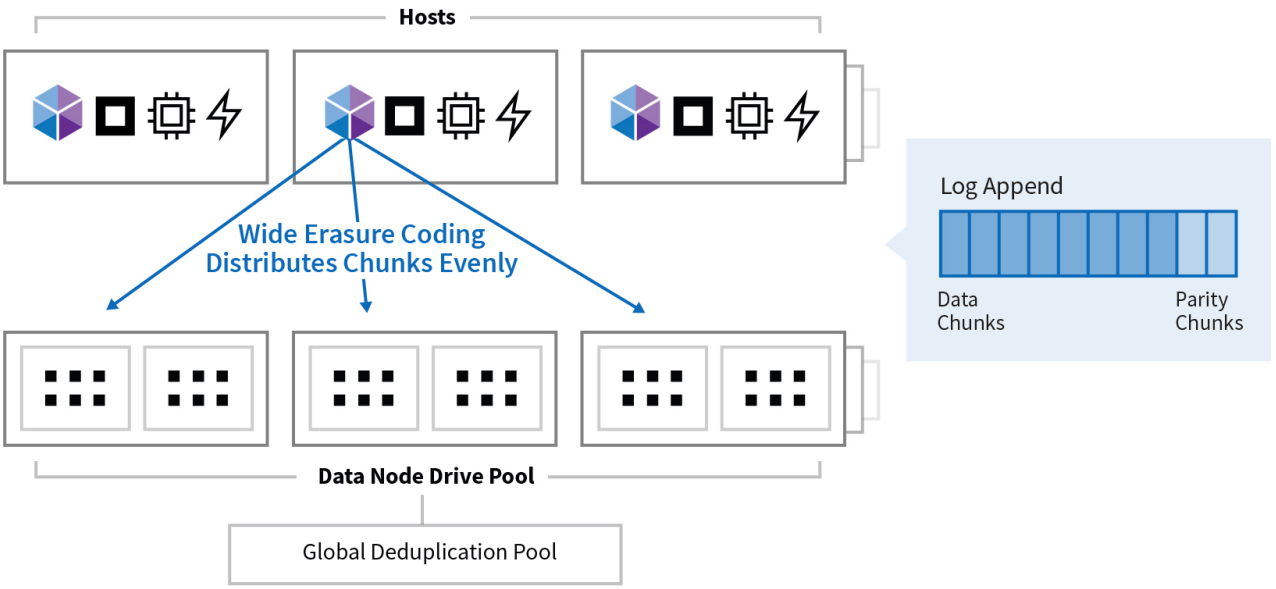


Figure 4
Datrium Automatrix: Architected for Full Stripe Writes

Full Stripe Writes

To benefit from the space efficiency of Erasure Coding without paying a significant performance penalty, a system must be designed from the ground up with full stripe writes in mind. Current HCI offerings simply added Erasure Coding as an afterthought to address the space inefficiency of data mirroring via RF3/FTT=2. These systems were not specifically designed to avoid the IO amplification of small writes.

Automatrix always writes full erasure-coded stripes eliminating the extra read and write IOs for parity calculations. Once written, the stripes become immutable and are never partially overwritten. Small writes incur no read-modify-write penalty. A distributed log-structured file system discussed in the subsequent sections is critical for enabling full stripe writes.

Hosts (also known as Compute Nodes) perform full network stripe writes to a set of Data Nodes with each Data Node maintaining its own local drives. DVX uses a wide erasure-coded stripe of (8D+2P) to increase write throughput and to lower the extra space overhead for encoding. Layout maps are maintained to map stripes to Data Nodes and disks. DVX uses declustered RAID layouts: data, parity and spare space are spread out across all disks of Data Nodes. The write bandwidth and disk rebuild speed increase with the addition of new Data Nodes.

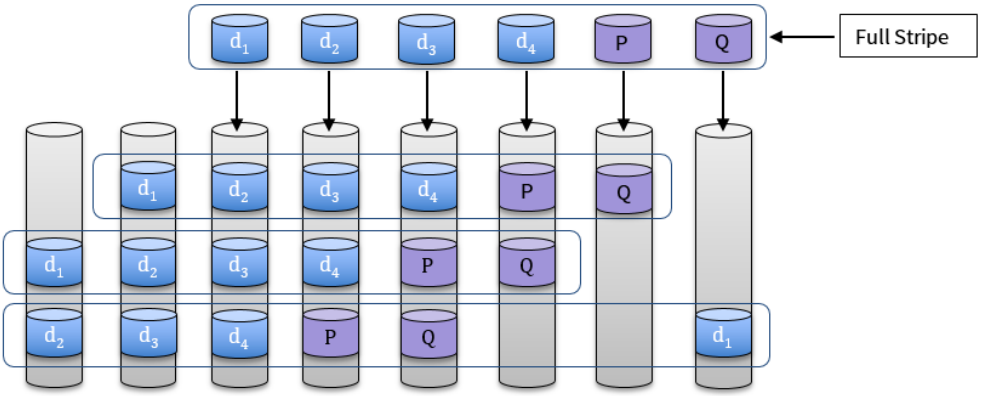


Figure 5
Full erasure-coded declustered stripe write in Datrium Automatrix

Because stripes are immutable, when the workload overwrites its content, new stripes are written and LFS data structures are updated to point to the new data location. Full parity information is always calculated from a full stripe buffered in memory without ever having to read any partial stripe information from disk. Like HCI, Datrium DVX is a distributed system and reading old data just to update parity information would otherwise result in expensive network reads.

To scale with the number of servers and VMs, DVX harvests the processing power of commodity x86 CPUs for Erasure Coding. Servers executing VMs that produce new stripe data also erasure code full stripes using x86 SSE4 and AVX vector instructions before full stripes are written to Data Nodes. In conventional arrays, storage controllers perform all RAID calculations and become a centralized bottleneck as the number and intensity of workloads increase. With DVX, a scale-out Software-Defined Storage Controller executes on all servers. Data Nodes are not involved in erasure coding.

One of the downsides of EC vs. mirroring is potentially lower rebuild speed on disk failure. DVX addresses this problem architecturally – the rebuild proceeds concurrently without centralized chokepoints and rebuild performance scales with the addition of new Compute and Data Nodes.

Buffering Writes

VMs issue IOs in a variety of sizes, generally at a much smaller granularity than a full stripe. Incoming data is buffered by Compute Nodes to accumulate a full stripe before the stripe is encoded and written out to Data Nodes.

DVX compresses the incoming data and stores it in the local RAM buffer of the Compute Node prior to striping. Simultaneously, it also synchronously writes the compressed (and optionally encrypted) block to a non-volatile log on the Data Node. This non-volatile log is also mirrored to protect against a physical controller failure(s). The DVX then acknowledges the write back to the hypervisor executing on the Compute Node that, in turn, acknowledges it to the VM.

When VM writes enough data for a full stripe (on the order of 10MB), Compute Node encodes the stripe and writes it out to Data Nodes over the network.

The copy of data present in two (or more) NVLogs is used to protect against Compute Node failures and is not used otherwise. After a full stripe is written, the corresponding NVLog entries are removed. This buffering architecture guarantees that written blocks are persisted to two or more durable locations during the entire write life cycle even though Compute Nodes perform Erasure Coding directly on their own (third) RAM copy of the buffered stripe.

NVlog is a well-known component of storage arrays that historically implement it via custom NVRAM PCIe cards. However, other more affordable and higher performance commodity alternatives have become available recently:

- **NVDIMMs:** non-volatile memory in the DIMM form factor that can coexist with memory DIMMs.[13] Several types of NVDIMMs are available on the market. Some, much like RAID controllers, use volatile DRAM during normal operation, but dump its content to on-board flash on power failure. NVDIMMs can operate at DRAM speed.
- **NVMe SSDs:** fast SSDs accessed via PCI Express bus with similar latencies to traditional NVRAM PCIe cards of storage arrays.
- **3D Xpoint:** Intel's new non-volatile memory technology delivered in both SSD and NVDIMM form factors.[14]
- **Battery-backed RAM:** This solution protects a portion of the RAM with a battery-equivalent power source. The protected memory operates at normal DRAM speeds and is vaulted to flash on power failure.

13 https://en.wikipedia.org/wiki/NVDIMM
14 https://en.wikipedia.org/wiki/3D_XPoint
15 John K. Ousterhout, Mendel Rosenblum. (1991), The Design and Implementation of a Log-Structured File System (PDF), University of California, Berkeley

Log Structured File System

Advantages

1. Sequential log organization
2. No overwrites
3. Good for full stripe writes
4. High write throughput
5. Friendly to HDDs and SSDs
6. Scales with nodes

Log Structured File System (LFS) is a file system in which all data and meta-data are written to a sequential log in order to avoid random overwrites. LFS design was based on the observation that, as DRAM continues to grow in capacity, reads would be increasingly satisfied by RAM caching and, therefore, the file system should focus on processing writes.

LFS was developed in the early 90s at UC Berkeley.[15] As it turned out, the hypothesis behind LFS was correct and DRAM continued to drop in price and increase in capacity over the last two decades. Perhaps even more importantly, SSDs that are both denser and cheaper than DRAM are now commonly used for file system caching. Interestingly enough, SSDs themselves are also internally managed via LFS based firmware.

Because LFS can batch log writes into large sequential runs, it provides an excellent foundation for always-on Erasure Coding – LFS log batches could be split into stripes and written to the log in their entirely. Writing large sequential runs also increases the write throughput of magnetic disks and the durability of SSDs.

In a DVX implementation, the log is distributed across Data Nodes. Erasure Coding is performed by Compute Nodes. The Datrium implementation of LFS scales in two different dimensions: Compute and Data Nodes can scale independently of each other.

As VMs overwrite locations within their virtual disks, previously written old blocks become obsolete. These old blocks are embedded in old stripes that make up the LFS log. In order to reclaim the space consumed by these obsolete and unreferenced blocks, DVX periodically runs a low-priority distributed Space Reclamation process. Space Reclamation tasks run on scalable Compute Nodes, utilizing abundant and cost-effective processing resources. Space Reclamation scales linearly as more Compute Nodes and Data Nodes are added to the system.

Because LFS never overwrites data in place, it does not exhibit failures resulting in inconsistent data. Traditional hardware RAID controllers have the worst-case failure modes of partially draining the write-back caches to disk. This may result in inconsistent stripes on disk. With LFS, even if stripe writes partially fail due to an extremely unlikely compound failure (such as a simultaneous failure of several NVRAM modules of different Data Nodes), the data already on disk remains fully consistent.

Server Side Flash

LFS is a write-optimized file system that requires read acceleration via a caching mechanism to achieve good read performance. DVX uses host DRAM and flash to accelerate reads. Compute Nodes maintain DRAM buffers and large local flash caches. Read requests are almost always satisfied from the local flash – SSDs are sized to fully accommodate all powered-on VMs.

This is economically feasible because DVX uses inexpensive commodity SSDs combined with inline flash deduplication and compression techniques. To accommodate commodity SSDs, flash itself is managed via a local LFS based file system (different from the distributed global LFS used for managing durable storage). Such an organization significantly increases SSD endurance making it possible to use a variety of inexpensive read-optimized flash SSDs for DVX read caches. Just like for durable storage, flash cache compression and deduplication are always-on and cannot be disabled.

Read requests are satisfied from local flash caches without ever issuing IO requests to the network. This is in contrast with traditional arrays that always require over-the-network IO.

This architecture virtually eliminates read requests to erasure-coded stripes letting LFS do what it does best – efficiently manage its distributed write log.

Conclusion

DVX Features

1. Always-on Erasure Coding
2. Two drive failure tolerance
3. 25% space overhead
4. 4K rand writes: 1.8M IOPS

To achieve better space efficiency, HCI products recently added Erasure Coding as a bolt-on feature. It is often applied as a post-processing step triggering unnecessary IO. More importantly, because Erasure Coding is not integrated with the rest of the file system, it results in 6x IO amplification due to read-modify-write for small writes. For this reason, both industry analysts and HCI vendors themselves recommend enabling Erasure Coding only for a carefully selected set of workloads.

In this paper, we showed how the Automatrix architectural focus on space efficiency and reducing management complexity can turn Erasure Coding into an always-on integrated part of software defined converged systems without any performance trade-offs. **This results in ~3x more efficient storage capacity utilization for all workloads with guaranteed capacity efficiency improvements** and eliminates the labor intensive and error prone task of guessing workload characteristics.

Datrium DVX relies on Erasure Coding during its normal operation and exposes no controls to disable EC or change the level of data protection. It does so without any performance sacrifices for workloads heavy on small random writes and overwrites. **DVX with built-in Erasure Coding achieves 1.8M IOPS for 4K random writes for a system configured with 10 hard disk based Data Nodes** which exceeds the performance of most all flash arrays.