Excelero

# ACHIEVING EFFICIENT AND SCALABLE DISTRIBUTED ERASURE CODING WITHOUT THE PERFORMANCE PENALTY

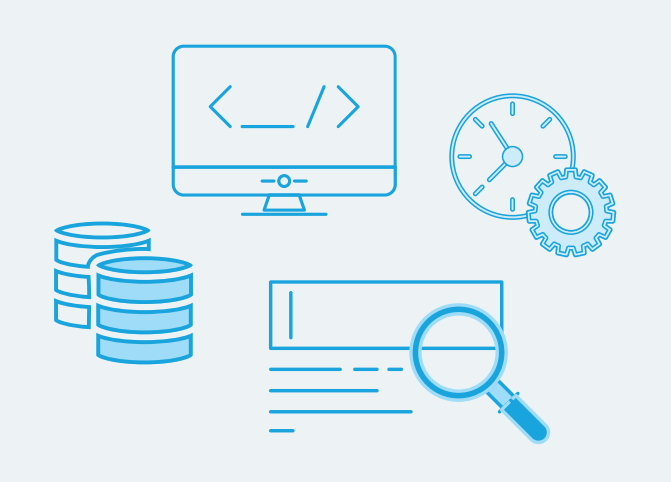
WHITE PAPER

## ERASURE CODING FOR SCALE-OUT, SHARED STORAGE INFRASTRUCTURES IN CONVERGED DATA CENTERS: ACHIEVING HIGHER DATA PROTECTION LEVELS AND RESOURCE UTILIZATION WITH REPLICATION-LIKE PERFORMANCE

### SETTING THE STAGE: MIRRORING VS. ERASURE CODING IN SCALABLE STORAGE INFRASTRUCTURES

Storage capacity and performance requirements are higher than ever. The flash revolution enables meeting both requirements without increasing, in most cases even decreasing, the storage footprint. This makes individual drives even more valuable as they store more and more data, which requires protection. Data protection can be achieved through replication or erasure coding. Both methods have their benefits and costs: mirroring has virtually no impact on performance but there is a higher cost as each copy requires more capacity. The cost/capacity calculation is very straightforward: you choose the required level of protection (2 or 3 copies are most common) and you know exactly how much additional capacity will be required for this protection. Erasure coding drastically reduces the capacity requirement, for similar or better data protection levels, but you have a performance trade-off. In this paper, we will show you how to pay less for the same level of protection, without performance trade-off.

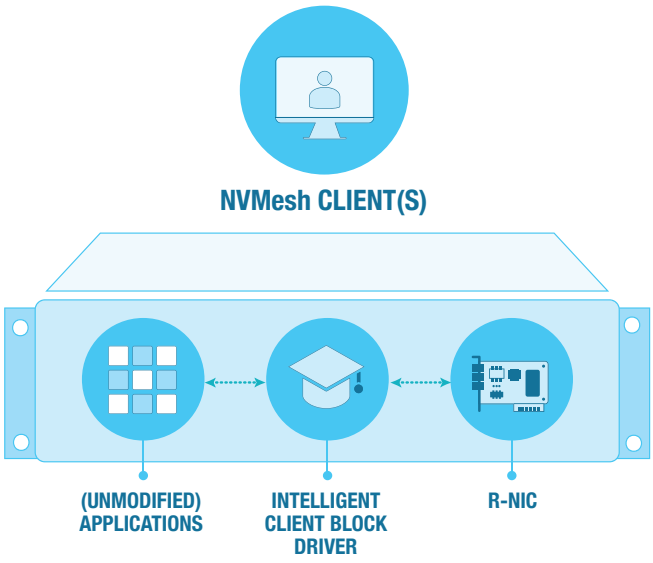
We do not claim solving this long-standing issue for all cases. Instead we concentrate on large, scalable storage infrastructure deployment and distributed, scalable applications for which the mirroring tax is especially glaring.



## THE BENEFITS OF CLIENT-SIDE ARCHITECTURES

**The NVMe protocol specifically benefits flash media** since it is naturally lockless and reduces the software stack overhead to a minimum. An additional benefit for scale out storage deployments is the protocol's extension to media access over a network - NVMe-over-Fabrics (NVMeoF) which is naturally mapped onto RDMA fabric protocols like Infiniband or RoCE.

For the longest time, storage architectures have been using intermediaries between the actual storage media and the clients: "storage controllers". These controllers implement a variety of data services, cached and controlled access to the data. The cost of this additional layer was outweighed by the additional features and data efficiency they provided. However, with the new NVMe-based flash drives, the sheer amount of performance one can get from even a handful of such drives is so high any intermediary CPU becomes an unacceptable bottleneck. This is especially the case for flexible scale-out storage infrastructures.




**The solution to this problem usually follows one of two directions:**

- Building HW accelerators and/or embedded solutions to offload storage controller functionality from CPUs and bring it closer to the media. The overall package, consisting of several drives and FPGAs/HW accelerators/embedded CPUs, is then connected to the datacenter network.

- Removing the intermediary altogether by moving the functionality provided by the storage controllers to the client side and using the much larger client compute resource pool to implement the same functionality in software on the client side

*In this paper we focus on the latter approach.*

## TRENDS TOWARDS LARGER WRITES

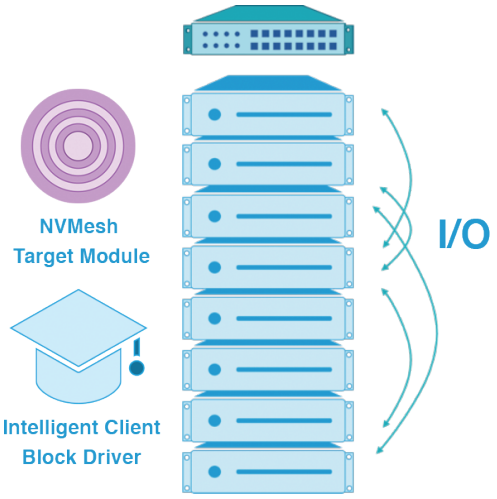
There are many existing applications that treat the block storage layer as a persistence layer for memory cache. In such case the storage access optimization is for bandwidth and not for latency which brings the transaction size to be in the range of hundreds of kilobytes to single megabytes. The popular Lustre filesystem is one such example - the default size for storage access is 4MB.

In recent years, in-memory processing has gained popularity across the industry with applications in scientific computing such as rendering and analytics. These solutions require persistence of in-memory information. Here, once again, the bandwidth of storage access has the highest priority and therefore storage access sizes tend to be at least 128KB. Burst buffer based on IBM Spectrum Scale is a nice example of such case.

## MODELS FOR COMPARISON AND ANALYSIS

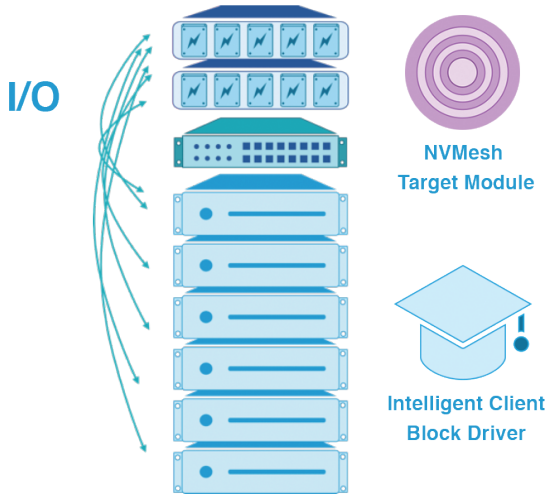
We will consider two widely used models for scale-out storage deployment in modern data centers: **converged and disaggregated architectures.** In both cases we assume mixed-use NVMe drives since they are frequently written to: around 2GBps in a sequential write case and around 200K 4KB IOPS in a random write case. These drives also provide ~800K random 4KB read IOPS.

### LOCAL STORAGE IN APPLICATION SERVER




**In a converged model,** all compute nodes double as storage nodes. typically, all nodes are standard servers that have several NVMe drives and at least two 25Gbps RNICs. The number of NVMe drives is determined by the ability of the RNICs to use the full performance provided by the drives, so we assume 4 drives per server (3 drives can drive 3*2GB/s = 6GB/s = 48Gb/s so almost 50Gb/s but this is a stretch so 4-drive configuration makes more sense). From a PCIe lane utilization perspective, having 4 drives make sense as well as they use 16 PCIe lanes which, together with 8 lanes for the RNIC, can be accommodated by most of the servers. A second RNIC is typically added for redundancy purposes.

### STORAGE IS CENTRALIZED




**In a disaggregated model,** specific "target" storage nodes supply storage to the "client" compute nodes. Target nodes can be based on JBOFs (Just a Bunch of Flash) or on dedicated servers with a large (usually at least 24) number of drives. To provide access to this many drives, target nodes usually have RNICs with up to eight 100Gb/s ports. Client nodes will have local storage only for the operating system and use RNICs of up to two 25Gb/s ports.

***We will concentrate on the converged model however the presented methods and conclusions are applicable to the disaggregated case as well.***

## REPLICATION-BASED RELIABILITY (MIRRORING)

For the highest write bandwidth, mirroring is an obvious choice. In large deployments the probability of a single drive failure becomes a significant one, making 3-way mirroring a better choice from the reliability perspective. However cost considerations may drive the decision to deploy 2-way mirroring. For the baseline we will consider both cases:

### 2-WAY MIRRORING:

- Maximum client write bandwidth is approximately 3GB/s
- Maximum client read bandwidth is approximately 6GB/s
- Usable capacity is 50%

### 3-WAY MIRRORING:

- Maximum client write bandwidth is approximately 2GB/s
- Maximum client read bandwidth is approximately 6GB/s
- Useable capacity is 33%

## ERASURE CODING-BASED RELIABILITY (RAID6)

**Erasure coding is used to provide the same reliability as mirroring (replication) but with much higher usable capacity from the same number of drives.** The technique uses erasure codes to recreate the lost data blocks in case of a drive failure. Since we assume a shared, client side storage infrastructure, the possibilities of client side write caching are limited so in our model we assume row-based erasure codes like Reed-Solomon (RS). These codes get more attention lately since there are both Intel x86 CPU instruction set additions and RNIC-based accelerators reducing CPU tax of implementing RS-like erasure codes in the datapath. In row-based erasure codes, there are two configuration parameters per “stripe”: N - the number of data blocks under protection and P - the number of parity blocks. From a write operation perspective, when one writes N data blocks it actually writes N+P so the larger N (given the same P), the more space-efficient the configuration is. On the other hand, for write operations, it is important to note that if a subset of N pages are re-written, the operation requires reading of at most N/2 pages which usually means much higher latency of such write operation. So to increase write bandwidth, the configuration needs to be chosen so that in most cases the full stripes of N blocks will be written. So we will analyze a very popular 8+2 configuration, which means that given a block size of 4KB any write operation containing multiples of 8*4KB=32KB won’t require any read operation. Dual drive protection for every 8 pages is enough even by the most rigorous standards.

From a reliability perspective it’s recommended to configure the system so that every drive in the RAID will reside on a different server. It is not a problem to have several different RAID groups sharing the same servers as the reconstruction of failed drives is done independently per RAID group. This requirement can be relaxed in the disaggregated scenario as JBOFs are usually built with higher reliability requirements, for

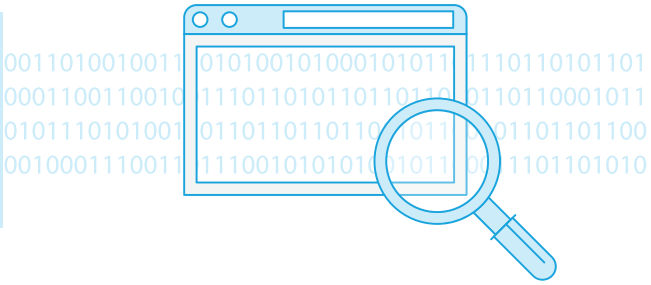
## ERASURE CODING - BASELINE

**As a baseline, we consider a client-based RAID6 system based on the NVMeoF protocol to access the remote drives.** We consider only write operations spanning multiples of 32KB in 8+2 configuration so each drive resides on a different server for maximum availability. We may relax the last bit for disaggregated model but it won't change basic principles of operation. The main problem with RAID based on erasure coding is that the write transaction is completed only when all 10 drives completed their operation. Since any interruption can leave the data in an inconsistent state (a "write hole" situation when the parity blocks don't reflect the data) the widely used solution is to keep a "journal" - a place where either the old or the new data is persisted until the transaction is completed. Then, in a case of disk failure or any other transaction interruption, a recovery operation is possible. Such recovery will either roll the transaction forward or backward in a consistent manner ('consistent' in this context means that read operations following the write will consistently bring the same data). As the journal writes need to be persistent, we can allocate a journal area on every drive and use NVMeoF writes to write to journal in the same manner the write operations are performed. In this case there is no need for remotely accessible persistent memory devices like NVRAMs, NVDIMMs or battery-backed RAM.

In such setup **each 32KB write operation is performed in 2 phases, each one consisting of 8+2 writes.** Overall 80KB are written, meaning that a client can push up to 6GB/s * (32/80) = 2.4GB/s effective write bandwidth from network perspective. From the target side, each drive is capable of 200K * 4KB = 800MB/s write bandwidth but since we write twice, the effective write bandwidth of a RAID group becomes 800MB/s * ½ * 8 = 3.2GB/s. Don't forget that the directions of write operation from client and target side don't interfere with each other.

In a converged scenario, if we consider 10 servers with 4 drives each as described above, we will have 4 RAID groups so the overall write bandwidth on the target side will be 4 * 3.2GB/s = 12.8GB/s. On the client side, these 10 servers could consume 10 * 2.4GB/s = 24GB/s. So the overall system write bandwidth is unbalanced. **However, the unused network bandwidth can be used to read from the drives:** each server receives 800MB/s * 4 = 3.2GB/s data while sending the same amount of network bandwidth to support this level of write performance. This means that all clients can use 10 * (6-3.2)GB/s = 28GB/s read bandwidth while the 40 drives are able to provide at least 120GB/s in random read performance.

**Summary (10 server configuration):**
Combined 12.8GB/s Write bandwidth
Combined 28GB/s Read bandwidth



## HOW NVMESH MAKES THIS CONFIGURATION MUCH MORE BALANCED AND EFFICIENT?

As NVMesh controls where the information is written on the target side, it can use the same bounce buffer when writing to both journal and data blocks. This means that a client needs only half the network bandwidth for the same effective write bandwidth. This implies that for 1.28GB/s write bandwidth a client will pay with only 1.28GB/s * (40/32) = 1.6GB/s network bandwidth, effectively rising the consumable read bandwidth to 10 * (6-1.6)GB/s = 44GB/s.

**Summary (10 server configuration):**
Combined 12.8GB/s Write bandwidth
Combined 44GB/s Read bandwidth

If we want to have an even higher write bandwidth we can use NVMe drives supporting Persistent Memory Region (PMR), which is already previewed by some drive vendors and will be part of NVMe standard v1.4. This is a flash-backed part of a Controller Memory Buffer - effectively a persistent memory on each drive which can be used for RDMA. If a drive supports PMR, NVMesh can use it as bounce buffers for write transactions transforming PMR into journal area per drive and eliminate the need for an additional write to a drive. In this case a RAID Group will be able to support twice the bandwidth - 6.4GB/s. So 4 such groups will be able to provide 25.6GB/s write bandwidth. Each client will need to supply 2.56GB/s * (40/32) = 3.2GB/s network bandwidth to support this write bandwidth. This leaves 2.8GB/s for the read bandwidth. Together with the single network traffic this means the configuration is able to support:

**Summary (10 server configuration):**
Combined 25.6GB/s Write bandwidth
Combined 28GB/s Read bandwidth
Usable capacity for all cases approaches 80%

## MIRRORING COMPARISON

To achieve the same level of protection, one needs 3-way mirroring which allows to reach only 20GB/s combined write bandwidth without leaving anything for read bandwidth since it consumes the full 6GB/s available by the RNIC.
With reduced protection and 2-way mirroring we can get 30GB/s combined write bandwidth but without leaving any space for additional read bandwidth.

## CONCLUSION

**For a converged, scale-out data center, leveraging shared software-defined storage, cost-efficient standard servers, RNICs and NVMe drives, we can reach mirror-level write bandwidth with better protection and more balanced performance profile.**