Excelero

LINEARLY SCALABLE, "LOCAL" BURST BUFFER

USE CASE

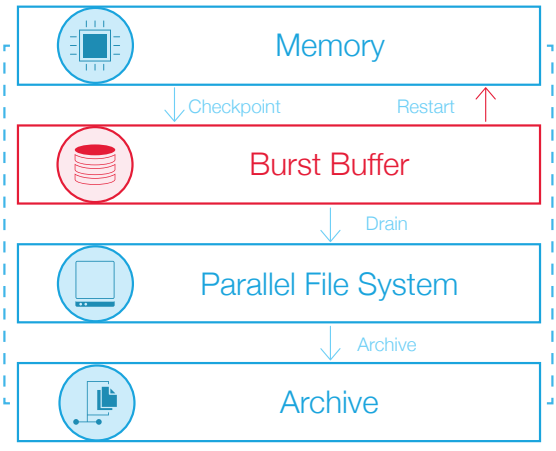
INTRODUCTION

High-performance computing applications consist of complex sets of processes that sometimes run for weeks on massive supercomputers. When any of these processes is interrupted, this could destroy the results of the entire compute job. This problem becomes worse as supercomputers become more powerful. Therefore, parallel computing applications use the concept of check-point-restart. This technique allows compute jobs to be restarted from the most recently saved checkpoint.

Checkpoints are typically saved in a shared, parallel file system. As clusters become larger and the amount of memory per node increases, each individual checkpoint becomes larger and either takes more time to complete or requires a higher-performance file system. When a system is checkpointing, it's not computing – and often these large supercomputers have limited windows they are allowed to checkpoint in.

This limited amount of checkpoint time has traditionally forced administrators to choose between two "evils": to either perform less frequent checkpoints (meaning losing more time on a restart) or sizing the performance of the file system to a very high write performance number that is not needed for general usage.

In recent years, another practice has emerged and is generally referred to as a "busrt buffer". Instead of making the entire file system meet a very large write performance requirement, a portion of the file system (or in some cases, a separate file system) is configured to take a burst of write IO at a very high rate.



Once the burst (checkpoint) is complete, the written data is "drained" to the larger, slower pool of storage making up the majority of the file system. This allows checkpoints to finish rapidly so that systems meet SLAs and the time between checkpoints is used to move the stored checkpoints to longer-term storage pools.

When flash storage is used as the burst buffer pool it has the added advantage of facilitating a faster restart (when needed) as checkpoint restarts often impose a very large random read load on the underlying storage. Thus, many bust buffer configurations are sized to hold at least two checkpoints so the most recent completed checkpoint is available for restart.

BURST BUFFER METHODS

There are many ways to build burst buffers for checkpoints but the goal is always the same: to complete the checkpoint as fast as possible to get the cluster back to its main purpose, which is computing. The different approaches largely fall into two categories:

BUILD THE FASTEST, CENTRALIZED STORAGE BASED, PARALLEL FILE SYSTEM YOU CAN, UP TO THE SPEED YOU CAN AFFORD.

BURST TO A TEMPORARY LOCATION, PERHAPS LOCAL, AND MOVE THE INDIVIDUAL HOST BURSTS TO A CENTRALIZED LOCATION LATER.

BOTH METHODS HAVE MAJOR WEAKNESSES.

The centralized approach suffers in many ways:

- **Cost** - making a very large parallel file system from multiple redundant (proprietary) controller pairs is very expensive. Additionally, you must build the entire system for peak bandwidth when most time it is not being pushed very hard.
- **Power (and cooling)** - when using disk drives, this uses a lot of power.
- **Bottleneck** - if you use flash to speed things up, the controllers become the bottleneck.

On the positive side, any written checkpoint is immediately preserved, useable.

Bursting to a temporary location seems attractive in that you can scale the burst speed in unison with compute hosts. You then drain the checkpoint from all the individual hosts to a centralized location at a reasonable bandwidth. This allows the parallel file system to be built economically for durability and capacity with reasonable performance. The biggest problem with this approach is that while the checkpoint remains on the compute host's local media, it is subject to failure with the host itself – making the checkpoint potentially useless.

THE NVMesh® "LOCAL" BURST BUFFER

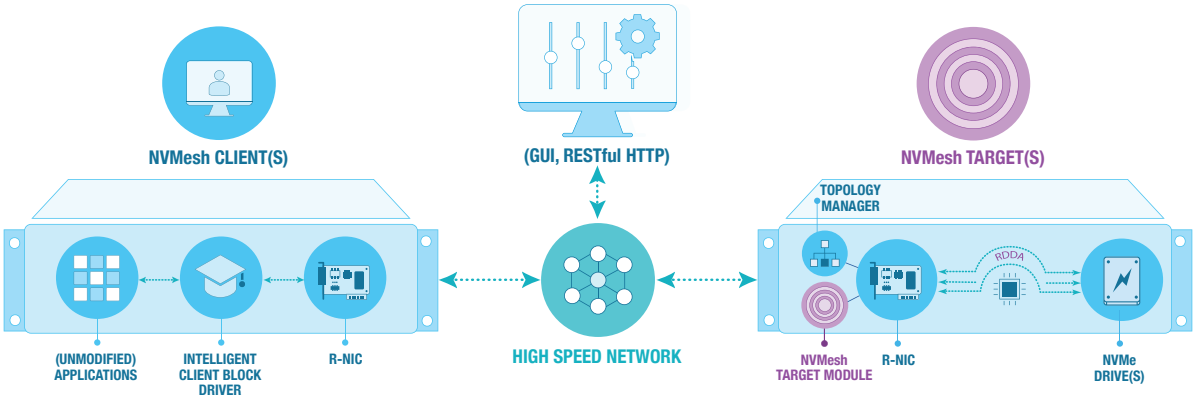
Excelero NVMesh offers unmatched deployment flexibility on standard servers and components without the need for additional storage servers or proprietary hardware - unlike other solutions.

Some solutions that integrate with IBM SpectrumScale (GPFS) and require IBM appliances – or at least dedicated hardware. For other solutions integrated as hardware from some HPC vendors, you must buy from that vendor, with proprietary HW and SW tied to the supercomputer you purchase.

With NVMesh for burst buffer, you can source standard NVMe drives and can completely obviate the need for proprietary hardware, and even dedicated storage appliances. It builds on the local burst buffer methodology but with a unique advantage: it adds redundancy with centralized management while at the same time preserving all compute resources for the applications themselves.

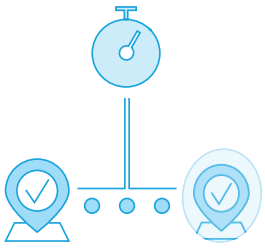
NVMesh and patented Remote Direct Drive Access (RDDA) technology allow you to logically disaggregate NVMe drives in the compute nodes away from CPU resources. That is, though the local NVMe drive may be used by remote compute nodes, that usage does not consume local CPU.

NVMesh® Software Components



Thus, every compute node can have a local NVMe SSD (or multiple drives) and all the drives are pooled for use by the cluster. In the most simplistic form, with something like Lustre - 1/2 of each drive is used as a local burst buffer and 1/2 is reserved for the redundant copy of a peer. As a result, when a node fails, it's scratch is preserved and accessible by an alternate node - any node on the fabric.

Each node has a local scratch space (let's say **/scratch**) mounted on one of these mirrors. When a checkpoint is initiated, contents of the application memory are written to the locally mounted file system **(/scratch)** that writes to the local drive, and mirrors to a peer. This happens for every node, simultaneously. If using 100Gb/s fabric, in full duplex, this means each node would be bottlenecked at about 12GB/s BW. Thus, the bottleneck will likely be your choice of NVMe drives, and how many. The main benefit is that you scale linearly with every node. Depending on your choice of device, each device writes between 500MB/s and 2GB/s.



With a single a single high-performance NVMe device per compute node, at 2GB/s - a host with 512GB of RAM would checkpoint in about 8-9 minutes. This is because each drive will be taking in the local checkpoint plus the mirror of another node's checkpoint. All nodes can write simultaneously, so effectively, the entire cluster can checkpoint in 8 minutes regardless of the number of nodes - as long as "pairs" have at least 2GB/s of network bandwidth to each other. With 2 NVMe devices per host, this time is cut in 1/2

Once all nodes have completed, the local checkpoints can be transferred to a centralized (spinning) repository, likely a parallel file system, at a much slower rate. For a 5000-node cluster, you could checkpoint in 8 minutes. This would be at an effective rate of 5TB/s of bandwidth. With each node at 512GB, the total of all checkpoint files would be 2.5PB. If your spinning disk Lustre file system was capable of 500GB/s, it would take about 90 minutes to copy the "local" (but protected) checkpoints to the Lustre file system. With 2TB NVMe drives, you could store 2 checkpoints within the "local" burst buffer before you would have to drain one.

NVMesh® BENEFITS FOR BURST BUFFER

- Petabyte-scale unified pool of high-performance flash retaining the speeds and latencies of directly-attached media.
- Supports large-scale modeling, simulation, analysis and visualization.
- Visualizes supercomputer simulation data on 100s of compute nodes.
- Finish check pointing faster and start running the job.
- Achieve highest performance at the lowest price.
- Leverage the full performance of your NVMe SSD's at scale, over the network.
- Scale your performance and capacity linearly.
- Easy to manage & monitor, reduces the maintenance TCO.
- Utilize hardware from any server, storage and network vendor, no vendor lock-in.

CONCLUSION

NVMesh gives you an extremely cost effective method to achieve unheard of burst buffer bandwidth by adding commodity flash drives and NVMesh software to compute nodes and low latency network fabric that you were going to buy for the supercomputer itself. It provides redundancy without impacting target CPUs. There is no need for additional dedicated hardware or proprietary file system integrations as NVMesh looks like a simple block device.