Optimization Strategies for the Deep Learning storage stack

DDN EXAScaler

The Deep Learning Data Lifecycle

01 Acquire

02 Process

03 Train

04 Production

05 Store

Let's look at this bit

But remember there are more challenges in the other stages

So What is Different about Deep Learning?

And how can we create a storage environment that keeps things efficient and fast?

IO Data Path	Backtesting	Deep Learning
Application	Time Series DataBase	AI Framework
Application IO	Read, Wide and Random Big mmap operations	Read (and Write!), often mmap. Many re-reads
Data Volumes	Typically 100s TB to PBs	PBs --> 100s of PBs,
Compute System Processor	CPU	CPU and many GPU
Compute System Network	Single connection to 100G/IB	Many connections to 200G/IB

The Deep Learning Data Lifecycle

Let’s focus on three parts:

03 Train

Optimizations for….

1) The Containerized, GPU-centric Client
2) The RDMA, multi-pathed Network
3) The Deep Learning Data Patterns

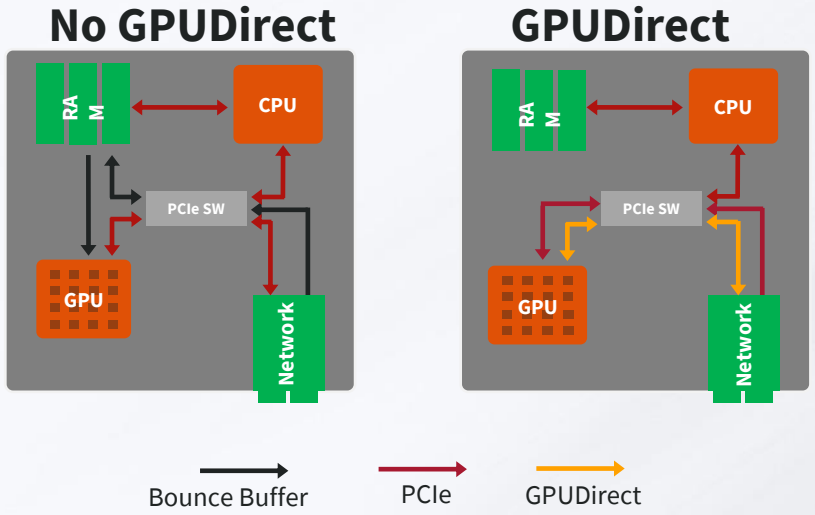
Optimizations for the Containerized, GPU-centric Client

An Example GPU Platform

- More complex that conventional compute!
- Two single port NICs (cluster) and two GPUs per PCIe switch, up to 24 GB/s per NIC, optimal path for GPUDirect Storage.
- Dual port NICs (storage) close to CPU, up to 28 GB/s per NIC.
- Eight GPUs interconnected within the system through NVSwitches (NVIDIA proprietary interconnect, 10x higher bandwidth than PCIe Gen4, 600 GB/s GPU-to-GPU).
- GPUs from multiple systems can communicate through a cluster network, ideally via single port NICs on same PCIe Switch as GPU (ports 1-8).

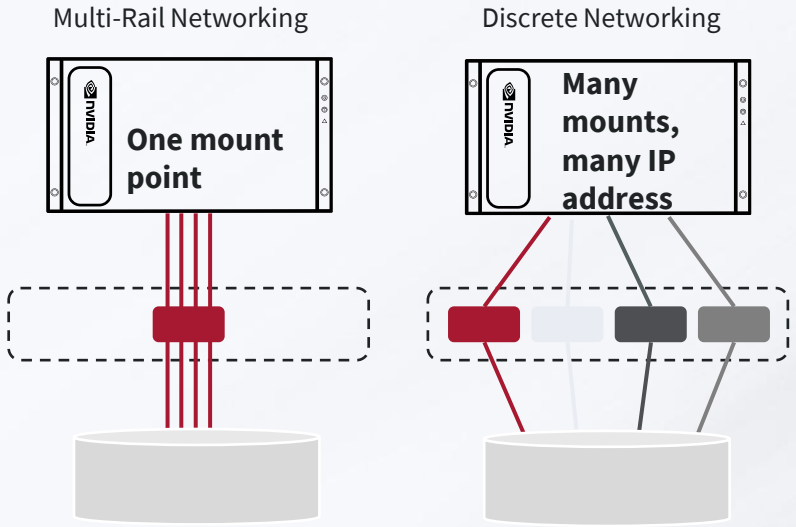
Optimizing the DataPath at the CPU/GPU level

- GPUDirect* enables Direct Remote Memory Access from the filesystem to GPU memory
- 2x-8x higher BW data transfers between Storage and GPU.
- 3.8x lower latency with no faulting and bounce buffers
- Stable and flat latencies as GPU concurrency increases.
- Lower consumption of host CPU or memory subsystem
- The GPU is the computing element with the highest IO bandwidth, e.g. 215 GB/s vs. the CPU's 50 GB/s.
- Very fast access to petabytes of remote storage faster than even the page cache in CPU memory.



Optimizing the DataPath at the Network level

- In GPU systems we usually have access to multiple interfaces. How can we use them all, without making our lives complicated?
- Some filesystems enable the grouping of network interfaces to achieve full aggregate throughput capabilities on a Client
- This doesn't need networking changes – it all happens with the filesystem software
- Improved peak performance with **single mount point and no complicated setup**



Optimizations for Deep Learning Data Patterns

Deep Learning Training Workflow

- Most Deep Learning workloads follow the same paradigm:
  - **Read Data** into GPU memory, compute and all-reduce
  - **Read Same Data** into GPU memory, compute and all-reduce
  - **Checkpoint** ← save the state for later use!
    - Maybe the job crashes and needs restarting
    - Maybe we want to change parameters (increasingly common)
  - Read Same Data into GPU memory, compute and all-reduce

Large Transformer Models create big challenges

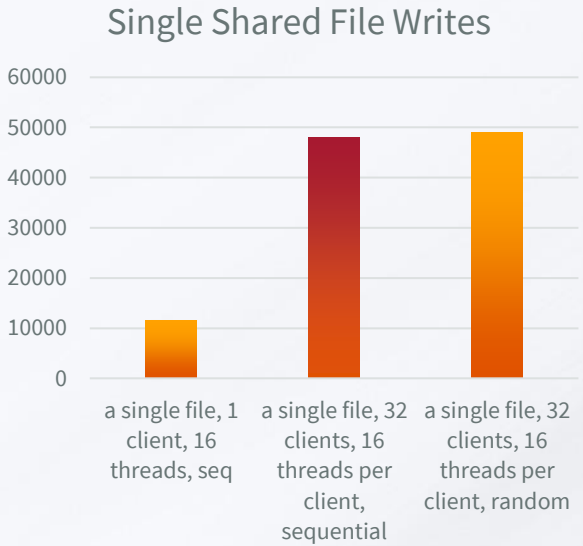
- GPT-3 scale from 1B to 1T parameters in size
  - parameters are the number of layers, the number of neurons per layer, the number of training iterations, etc. – these are getting bigger and bigger every year by multiple factors
- This is a tough use case for storage because large models require large checkpoints to hold their state
- **Example:** A GPT-3 13B parameter model: (this is quite a small model today!)
  - 4-way tensor parallel, 2-way pipeline parallel, distributed workload
  - Size of checkpoint file is **172GB** across 8 files ← a small model…
- Every minute spent waiting for IO is wasted time

Read and Write Performance of Storage

Usually there is ***some*** discrepancy between read and write performance for a storage system, since

- the write path needs to have data protection applied (erasure coding) and be written safely to persistent media or protected media
- Flash is faster for reads vs writes
- Concurrent writers to a single file need to be handled with locks

But for some storage systems the write is many factors lower than reads. To handle large scale checkpoints you want the storage system write performance to be more than 50% of the read performance.



Example of good write performance (2RU storage system)

* Not a STAC benchmark

The Multi-Epoch Training Problem

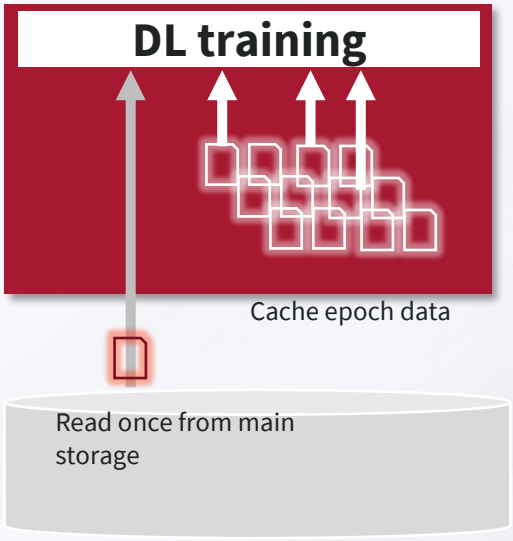
- Deep Learning Workloads need to read the same datasets over and over and over again – it's called multi-epoch training
- Gradient decent based learning algorithms can get it wrong! They need many passes over the data – i.e. Multiple epochs improve the chances of getting it right…

But

- Manually copying datsets to compute-local Flash is a tiresome process for admins
- Users are not necessarily familiar with data transfer strategies, cost and time,

So….

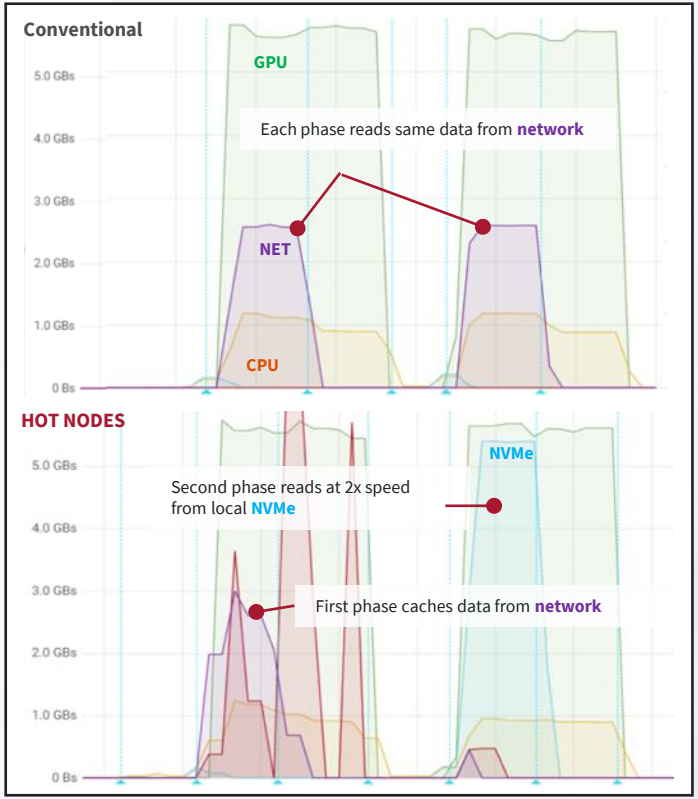
- Each compute node can have a local directory that can be used as a cache
- All necessary datasets are prepopulated into the cache
- All re-reads do not require any network traffic and keeps the whole network/storage environment less loaded for other concurrent workloads



Compute Cacheing for Multi-Epoch Training

- ResNet50 benchmark on DGX-A100 + AI400 without cacheing
- Each phase reads same data from network (purple)
- Compute runs in parallel with IO (CPU orange, GPU green)

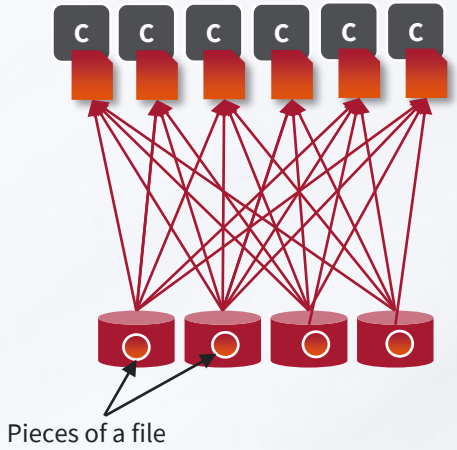
- ResNet50 with chacheing on internal NVMe devices
- First phase also reads from network (purple)
  - Total data read volume is similar, second read from RAM
- Computation also reads from network/RAM while files copied
- Write to cache storage on NVMe (red)
- Second phase reads from NVMe at double bandwidth (cyan)
- GPU usage (green) the same, CPU usage (orange) lower
  - No network/server load on second and later runs



Shared File Operations

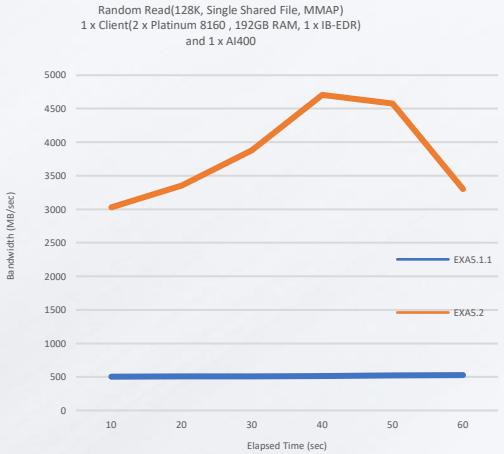
Many processes on a distributed system often want to read to a single file.

- This can often be a problem for filesystems that aren't truly parallel.
  - the pieces of the file reside upon a limited set of physical devices
  - Even if each individual process is reading in a nice sequential fashion, because there are many competing readers, the IO pattern seen by the storage system will be random
  - If a storage system is not really parallel, the devices can become an area for contention

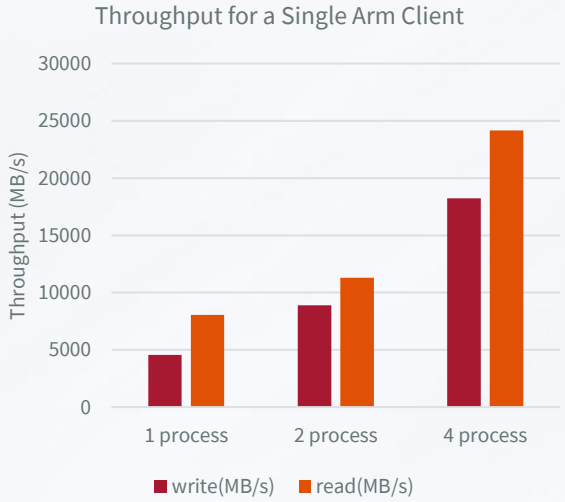


How to Make Shared File Operations FAST

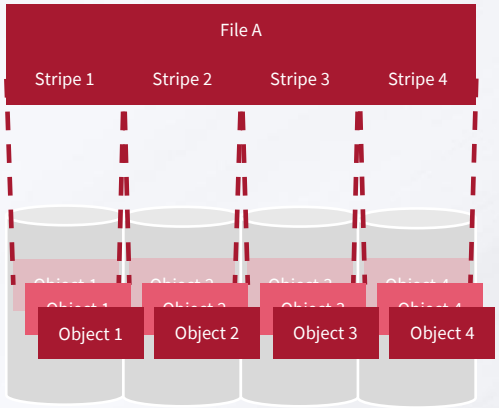
1. Let's ensure MMAP is Fast



2. Let's ensure 1,2,4 threads is fast

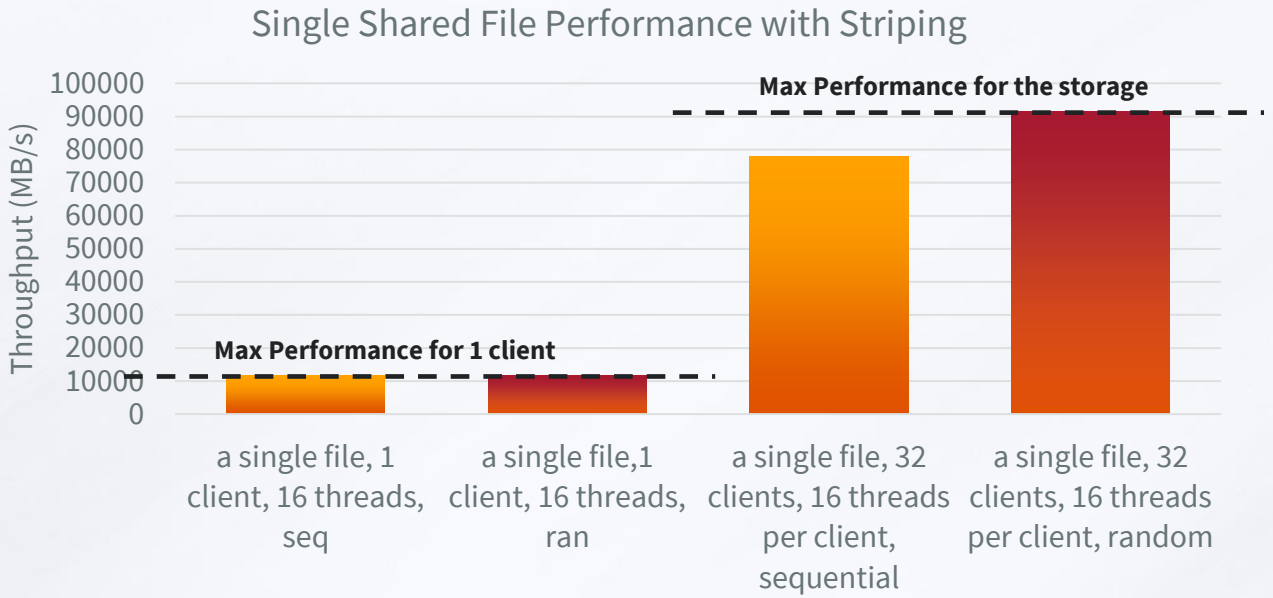


3. Let's scale



* Not a STAC benchmark

Parallel Filesystems can max-out the hardware even with just one file



* Not a STAC benchmark

Optimization Strategies for the Deep Learning Storage

- Deep Learning can involve very large amounts of data subject to:
  - Complicated compute-side environments
  - Checkpointing (writes) that can stop productive output
  - Large scale re-reads of data
  - Highly concurrent access to one file
- To find a suitable storage solution one needs to know:
  - Throughput/IOPs to a single thread!
  - Throughput/IOPs to a single client
  - Concurrency Results for shared file access
  - Write Performance
  - Cost and Complexity of the storage system to get the best numbers!

Latest STAC Benchmark KDB221014

vs All-Flash NAS solution*

STAC-M3 Antuco Suite

was faster in 13 of 17 mean response time Antuco benchmarks, including:

- 6x speedup in 50-**multi-user intervalized stats** (STAC-M3.β1.50T.STATS-UI.TIME)
- 5x speedup in **10-user aggregate stats** (STAC-M3.β1.10T.STATS-AGG.TIME)
- 4.9x speedup in s**ingle-user intervalized stats** (STAC-M3.β1.1T.STATS-UI.TIME)

STAC-M3 Kanaga Suite

Was faster in 21 out of 24 Kanaga mean-response time benchmarks, including:

2.1 – 4.4x speedup in **single-user high-bid** (STAC-M3.β1.1T.{2,3,4,5}YRHIBID.TIME)

***KDB220506**


ddn