File Direct – System Description

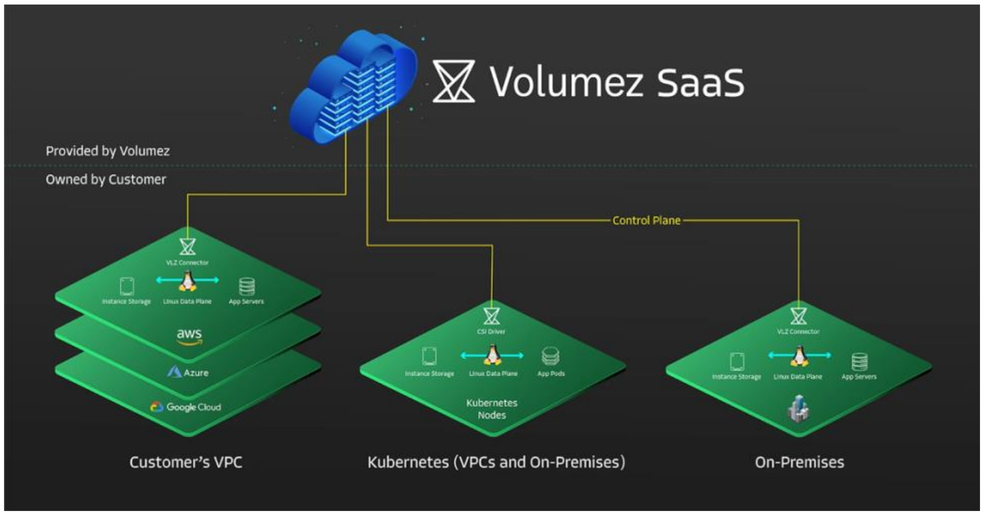
Volumez

August 18, 2024

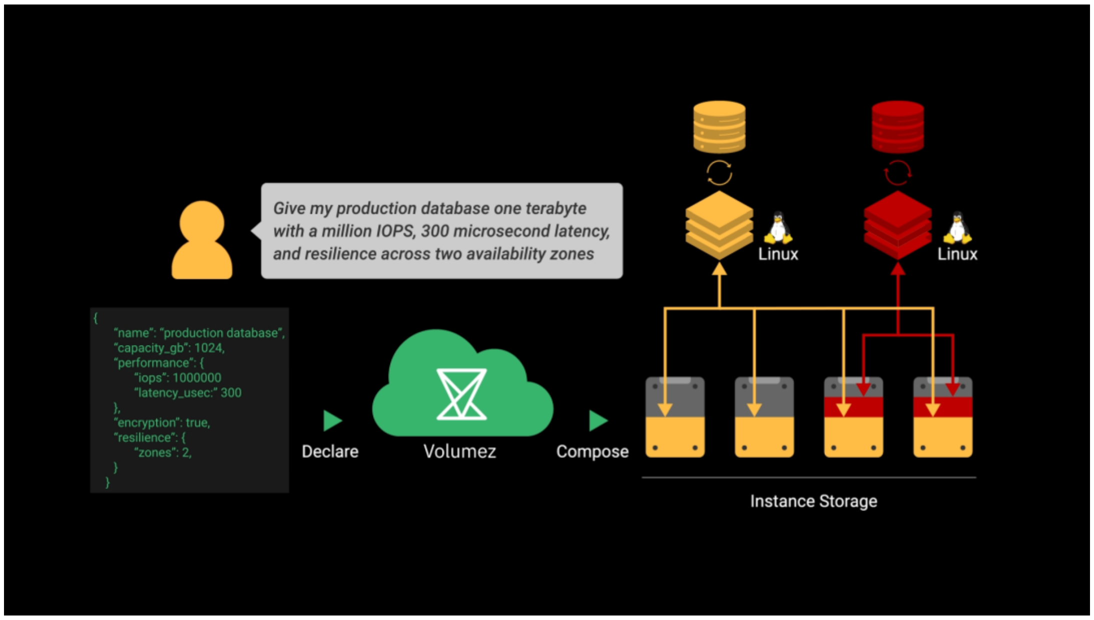
Table of Contents

About Volumez

Volumez is a new architecture for block and file storage in the cloud. Volumez separates the storage control plane, hosted in the Volumez cloud, from the storage data plane, which runs in customer virtual private clouds and data centers.



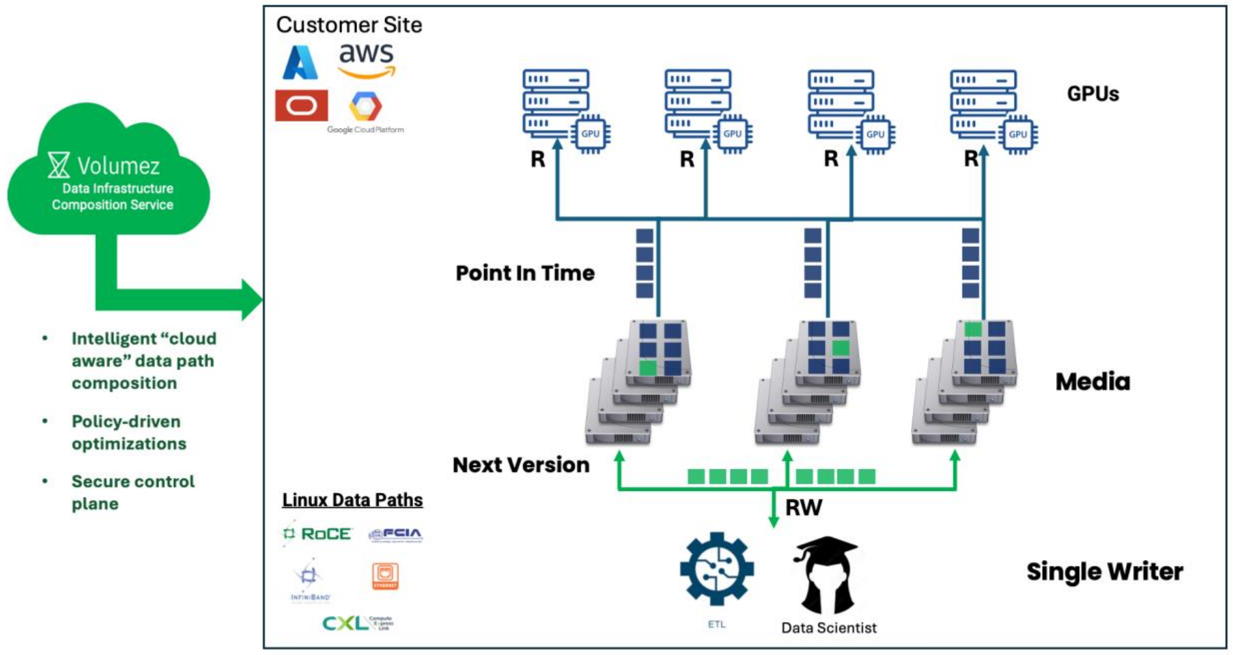
Volumez is not a scale-up or scale-out storage architecture. It is not even a storage system in the conventional sense, although it replaces cloud storage systems in practice.
At its core, Volumez is a composable infrastructure software that makes it easy for developers to request storage resources, similar to the way they request CPU and Memory resources in Kubernetes.


Give my production database one terabyte with a million IOPS, 300 microsecond latency, and resilience across two availability zones
Declare
Volumez
Compose
Linux
Linux
Instance Storage

```
{
    “name”: “production database”,
    “capacity_gb”: 1024,
    “performance”: {
        “iops”: 1000000
        “latency_usec:” 300
    },
    “encryption”: true,
    “resilience”: {
        “zones”: 2,
    }
  }
```

About *File Direct*

File Direct is our AI composable data infrastructure solution designed to support large scale training jobs.



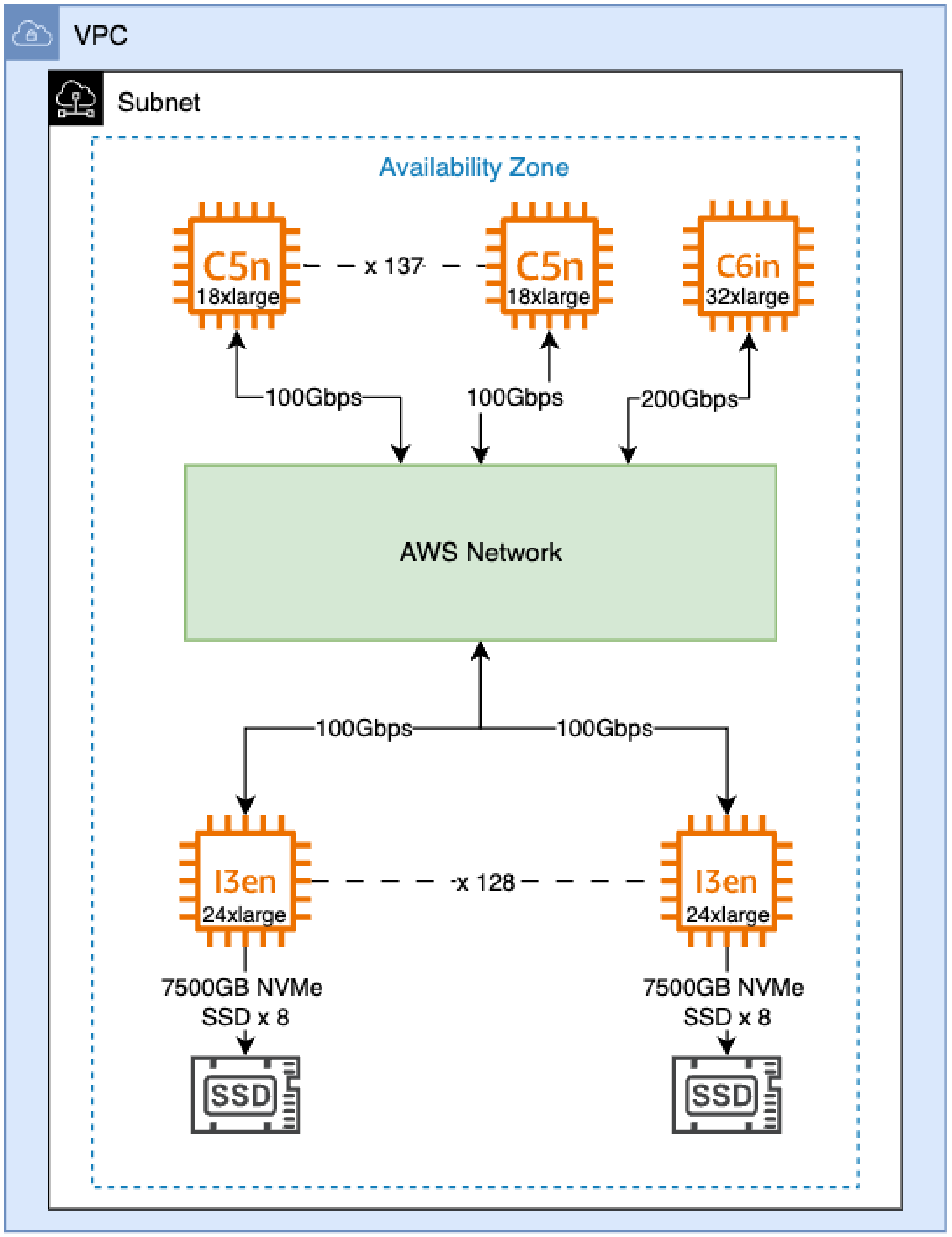
File Direct allows hundreds of GPUs to read simultaneously from the same volume and the same dataset.

It is also possible for a single writer to prepare the next generation of the dataset or prepare for the next experiment while the training job is running with no effect.

The entire data pipeline is Linux native and resides within the customer's environment. Volumez SaaS is only managing the control plane.

Benchmark Configuration

Benchmark Environment



Benchmark Configuration

- 137 host nodes (c5n.18xlarge), running 3 h100 accelerators for unet3d workload.
- 128 media nodes (i3en.24xlarge) orchestrated by the Volumez control plane.
- 1 c6in.32xlrage for data generation only.
- 2 Volumes were created:
    - Data volume
        - Size = 350 TiB
        - Policy = "Performance Optimized"
    - Checkpoint volume
        - Size 100GiB
        - Policy = "Performance Optimized"

Open Changes

- Custom Data loader. -- see description below

- NPY data format. -- see description below

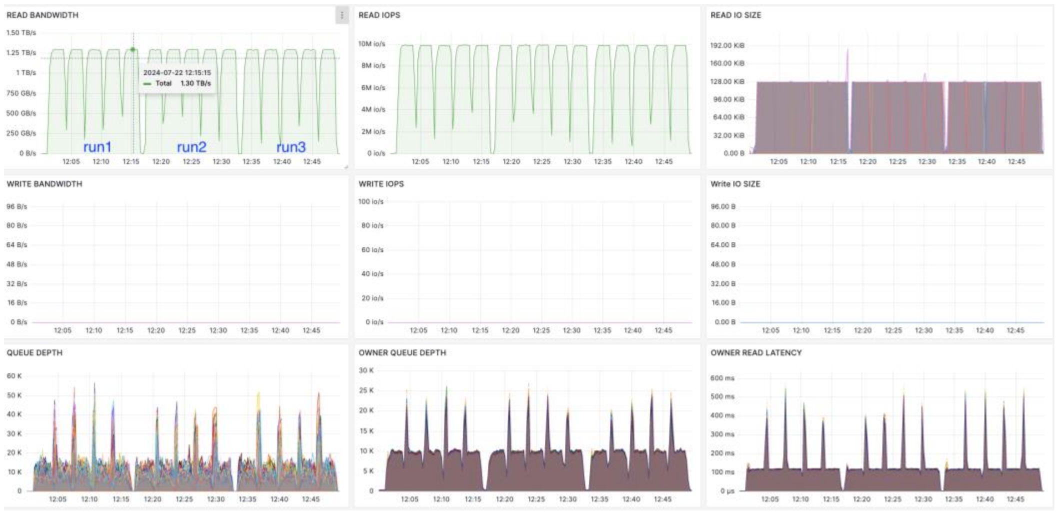
- Barrier frequency change – The barrier synchronization per batch at *dlio_benchmark/dlio_benchmark/main.py:277* is replaced by a per epoch barrier. See description below.

Results

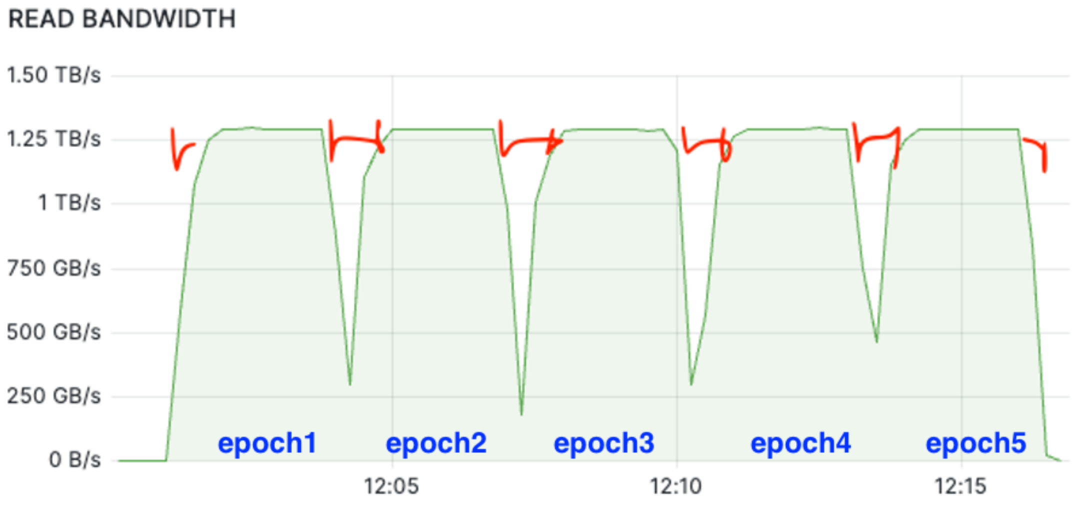
Metric	Average (5 runs)	Std
Throughput	1.14TB/s	0.8GB/s
Samples/s	8159	5.8
IOps	9.9M	-
AU Utilization	97.82%	0.08%

Our platform was able to achieve an outstanding aggregated throughput of 1.14TB/s, while maintaining an average 97.82% AU utilization as measured by the benchmark.

Our monitors are measuring a consistent peak performance rate of 1.3TB/s, as seen in the following image:



The source of the difference is in the benchmark, measuring the throughput over the entire time, including time between epochs that is not yet utilizing the storage.



Discussion

Custom Data loader

Our custom data loader issues O_DIRECT IOs to the storage system. This allows the IOs to be issued to storage directly without passing through the OS page-cache which creates redundant cpu bottleneck and memory pressure.

Data Format

The data was generated in .npy format instead of .npz format. The difference between the format types is that .npz represents compressed tensors, while .npy does not compress the data.

The decompression of the .npz files is compute intensive, preventing the benchmark from scaling and achieving the required AU utilization although the storage can keep up.

The necessity to separate samples online pre-processing (decompression in this scenario) from the compute nodes (e.g GPU nodes) in high scale is discussed in [1].

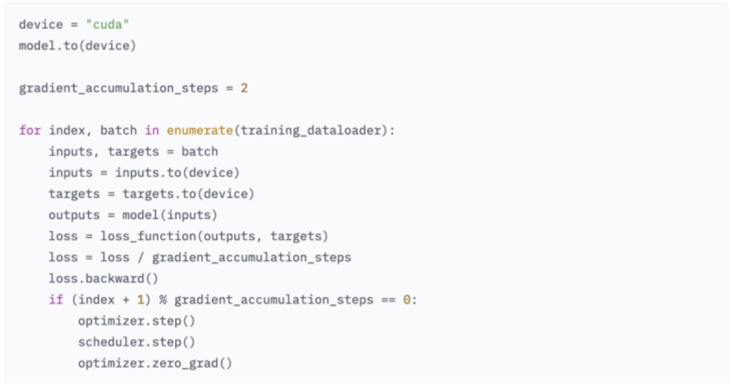
Weight Synchronization Frequency

Distributed learning involves training machine learning models across multiple GPUs and nodes, allowing for the handling of large datasets and complex models that exceed the capacity of a single machine. Weights synchronization is the process of updating the model weights across all GPUs during training. In the context of the benchmark, this is emulated by MPI's communication barrier. A significant challenge in distributed training is the communication bottleneck, where the overhead of exchanging data and model updates (weights synchronization) between nodes can hinder performance.

Over the last decade, several methods have been developed to address this bottleneck and to enhance efficiency and performance. These methods include asynchronous training, decentralized training, gradient compression, and advanced optimization techniques.

Asynchronous training allows nodes to update model parameters independently, reducing idle time and minimizing synchronization delays, but requires careful handling of stale gradients. A prevalent approach is to execute multiple optimization steps locally, keep the aggregation of the updates, and synchronize them every communication period τ . This approach was introduced by Zhang et al. [2]. They proposed the Elastic Averaging SGD algorithm, which enables the local workers to perform more exploration by reducing the amount of communication (τ) between local workers and the master. They demonstrated that this exploration can lead to improved performance. Stich [3] proved and demonstrated that Local SGD improves distributed training efficiency by reducing communication frequency while maintaining linear speedup and convergence rates comparable to mini-batch SGD. However, the significant communication reduction of Local SGD comes with a cost, as a larger number of local updates requires less communication but typically leads to a higher error at convergence. There is an interesting tradeoff between the error-convergence and communication efficiency. Wang et al. [4] proposed the Overlap Local-SGD algorithm that further improves the communication efficiency of Local SGD and achieves a better balance in the error-runtime tradeoff.

Notice that Local SGD can be easily implemented as described in this guide, https://huggingface.co/docs/accelerate/en/usage_guides/local_sgd, by Hugging Face.

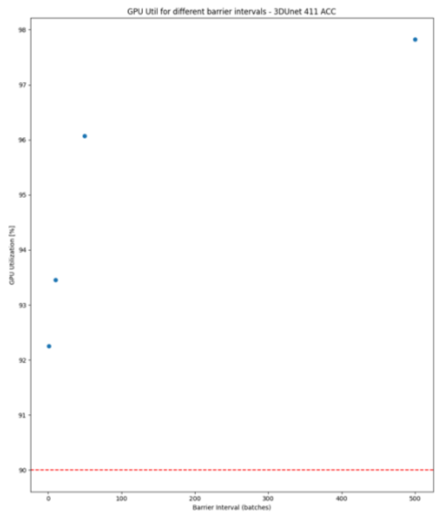


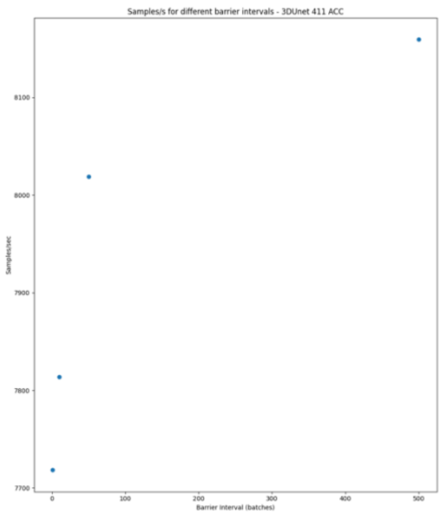
```python
device = "cuda"
model.to(device)

gradient_accumulation_steps = 2

for index, batch in enumerate(training_dataloader):
    inputs, targets = batch
    inputs = inputs.to(device)
    targets = targets.to(device)
    outputs = model(inputs)
    loss = loss_function(outputs, targets)
    loss = loss / gradient_accumulation_steps
    loss.backward()
    if (index + 1) % gradient_accumulation_steps == 0:
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
```

We ran the same experiment while changing the barrier interval. In our results, we can observe that the mean percentage AU increases as the barrier interval increases, meaning that the communication frequency decreases. We can also observe a similar growth in the storage samples/s, implying that the benchmark was not requesting data.





Furthermore, in real training applications, GPUs are using RDMA networks to communicate weights synchronization. This happens much faster over RDMA than over ethernet networks. This also gives an artificial advantage to participants with better networks. As we acknowledge the importance of this procedure in real life, this is driving the GPU utilization down as the benchmark scales, effectively preventing the benchmark from accurately measuring storage performance on high scale (compared to real life scenarios).

It has been shown above that there are methods to reduce communication frequency while maintaining linear speedup and convergence rates comparable to mini-batch SGD. Therefore, changing the barrier interval in the benchmark can represent a real-life scenario of workload of distributed training that uses Local SGD. Deciding on the weight synchronization strategy and frequency greatly varies for real life applications, and depends on the specific task, setting, objectives, and infrastructure constraints, e.g. network bandwidth.

Works Cited

[1] M. Zhao, N. Agarwal, A. Basant, B. Gedik, S. Pan, M. Ozdal, R. Komuravelli, J. Pan, T. Bao and H. a. N. Lu, "Understanding data storage and ingestion for large-scale deep recommendation model training," *In Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA),* pp. 1042-1057., 2022.

[2] S. Zhang, A. Choromanska and Y. LeCun, "Deep learning with elastic averaging sgd.," *Advances in neural information processing systems,* p. 685–693 , 2015.

[3] S. U. Stich, "Local SGD converges fast and communicates little.," 2018. [Online]. Available: arXiv preprint arXiv:1805.09767.

[4] J. Wang, H. Liang and a. G. Joshi, "Overlap local-SGD: An algorithmic approach to hide communication delays in distributed SGD.," *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2020.