Clairvoyant Prefetching for Distributed Machine Learning I/O

Nikoli Dryden
ndryden@ethz.ch

Tal Ben-Nun
tal.bennun@inf.ethz.ch

Roman Böhringer
romanboe@student.ethz.ch

Torsten Hoefler
torsten.hoefler@inf.ethz.ch

arXiv:2101.08734
High Performance Training
High Performance Training
High Performance Training
High Performance Training

Compute 😊

Communication 😊
High Performance Training

Compute 😊  Communication 😊  I/O 😞
High Performance Training

I/O overheads up to 85%!
High Performance Training

I/O overheads up to 85%!

Example: ResNet-50 on ImageNet-1k
- ImageNet-1k: ~150 GiB, ~1.3M images (average: 115 KiB, range: 508 B – 15 MiB)
- MLPerf on one A100: ~2.9K samples/s ➔ ~333 MiB/s random access
- ➔ 2 SSDs / GPU
- 2-4x for scientific problems like CosmoFlow
High Performance Training

NoPFS: Near-optimal Pre-Fetching System
Up to 5.4x end-to-end training improvements!
I/O for Machine Learning
I/O for Machine Learning

Randomly sample mini-batch
I/O for Machine Learning

Randomly sample mini-batch ➔ Epoch
I/O for Machine Learning

What makes a good I/O framework?
I/O for Machine Learning

What makes a good I/O framework?

System Scalability

...
I/O for Machine Learning

What makes a good I/O framework?

System Scalability

Dataset Scalability
I/O for Machine Learning

What makes a good I/O framework?

System Scalability

Dataset Scalability

Full Randomization
I/O for Machine Learning

What makes a good I/O framework?

- System Scalability
- Dataset Scalability
- Full Randomization
- Hardware Independence
I/O for Machine Learning

What makes a good I/O framework?

System Scalability

Dataset Scalability

Full Randomization

Hardware Independence
I/O for Machine Learning

What makes a good I/O framework?

- System Scalability
- Dataset Scalability
- Full Randomization
- Hardware Independence

Livermore’s El Capitan Supercomputer to Debut HPE Rabbit ‘Near-Node’ Storage

More distant storage
I/O for Machine Learning

What makes a good I/O framework?

- System Scalability
- Dataset Scalability
- Full Randomization
- Hardware Independence
- Ease of Use

```python
dataset = ImageFolder(data_dir, data_transforms)
dssampler = DistributedSampler(dataset, num_replicas=n, rank=rank)
dataloader = DataLoader(dataset, batch_size, sampler=dssampler)
```
Clairvoyant I/O

“Randomly sample mini-batch”

By “random”, we really mean pseudorandomly with a known seed!

We know the exact access pattern of every worker → We can exploit clairvoyance to optimize (distributed) I/O

NoPFS is a hierarchical, distributed cache and prefetcher that knows the future
Clairvoyant Prefetching and Caching

**Lemma 1.** If a worker accesses a sample \( \lceil (1 + \delta) \frac{E}{N} \rceil \) times (resp. \( \lceil (1 - \delta) \frac{E}{N} \rceil \) times), at least one other worker will access the sample at most \( \lceil \left( \frac{N-1-\delta}{N-1} \right) \frac{E}{N} \rceil \) (resp. at least \( \lceil \left( \frac{N-1+\delta}{N-1} \right) \frac{E}{N} \rceil \)) times.

**Single-process access distribution**

ImageNet-1k, 16 processes, 90 epochs

---

**Access stream**

\[ R = (\cdots, 7, 4, 5, 8, \cdots) \]

**Accesses for worker**

\( i \)

\[ \text{Cached in local storage} \]

\[ \text{Fetched from remote workers} \]

**Storage class 2**

**Storage class 1**

\[ \vdots \]

**Fetch sample** \( k \) from: \( \text{argmin} \) \( \text{fetch}_{i,(0,1,2),j}(k) \)

**Staging buffer**

\( \text{used} \quad \text{pending framework get} \)

---

**Some samples accessed 18 times!**
Loading ImageNet:

**PyTorch:**

```python
dataset = ImageFolder(data_dir, data_transforms)
dataloader = DataLoader(dataset, batch_size, sampler=d_sampler)
```

**NoPFS:**

```python
job = Job(data_dir, batch_size, num_epochs, 'uniform', drop_last)
dataset = NoPFSImageFolder(data_dir, job, data_transforms)
dataloader = NoPFSDataLoader(dataset)
```
Performance

Runtime per epoch

Up to 2.2x faster!

ImageNet-1k / ResNet-50

Up to 5.4x faster!
Performance

Runtime per batch

ImageNet-1k / ResNet-50

- Piz Daint
- Lassen

Runtime per batch:

- >100x
- >150x
Performance

ImageNet-1k / ResNet-50
Performance

NoPFS improves performance and reduces noise across systems and scales

ImageNet-1k / ResNet-50
Performance: Going Bigger

ResNet-50 / ImageNet-22k (1.5 TB)

- Up to 2.4x faster!

Lassen

CosmoFlow (4 TB)

- Up to 2.1x faster!
Conclusions

NoPFS is a hierarchical, distributed cache and prefetcher that knows the future.

System Scalability

Dataset Scalability

Full Randomization

Hardware Independence

Ease of Use

https://github.com/spcl/nopfs