The Case for Unifying Data Loading in Machine Learning Clusters

Aarati Kakaraparthy*†, Abhay Venkatesh*, Amar Phanishayee†, Shivaram Venkataraman*
University of Wisconsin, Madison* & Microsoft Research†

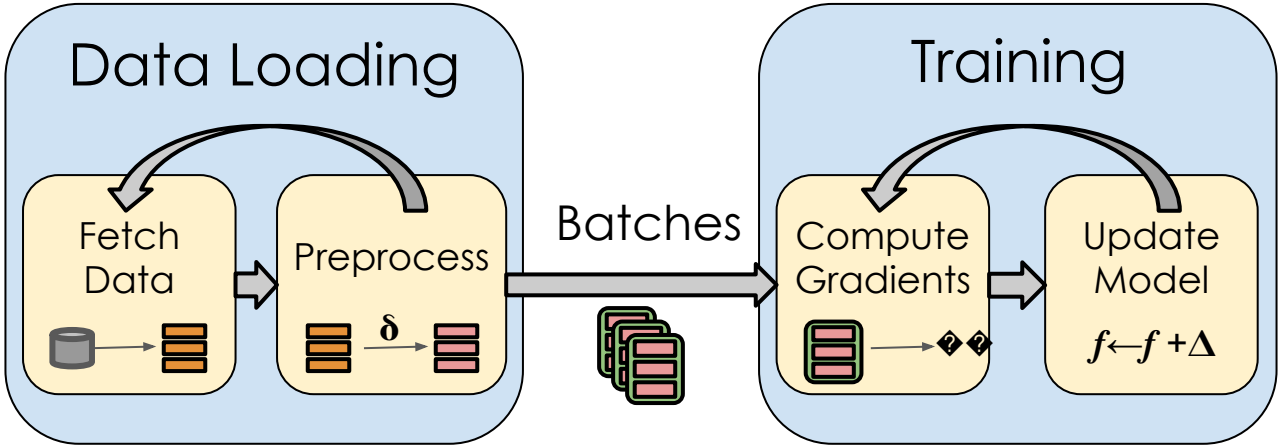
THE UNIVERSITY of WISCONSIN MADISON

Microsoft Research

Machine Learning Frameworks – Two Parts

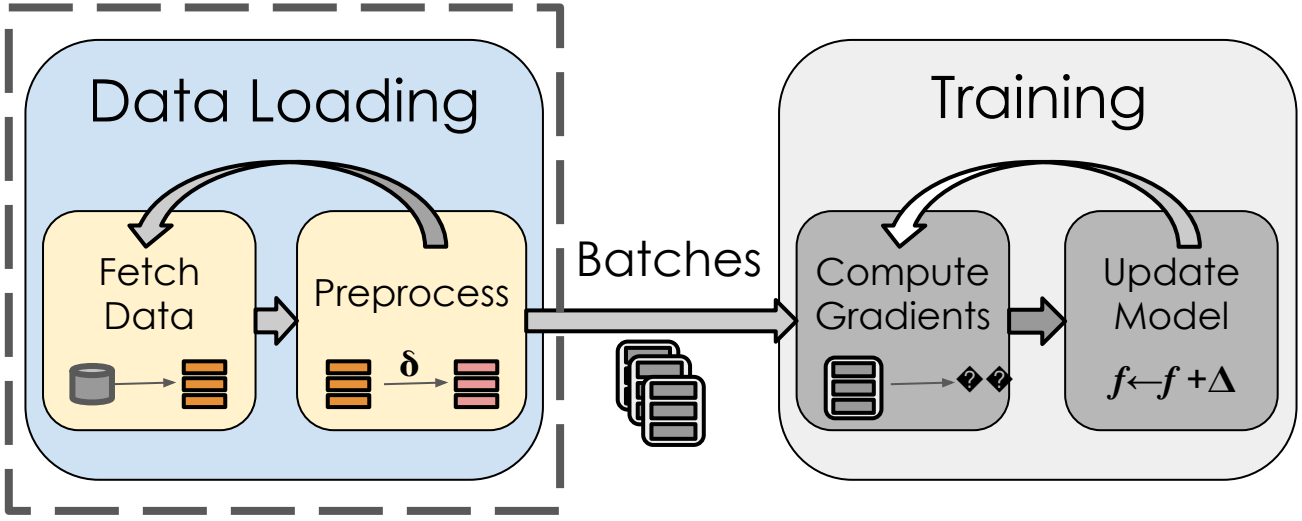
Machine Learning (ML) frameworks like PyTorch, Tensorflow, etc. typically consist of two sub-systems:

- ***Data Loading***, to generate batches of data, and
- ***Training***, to compute gradients and update the model

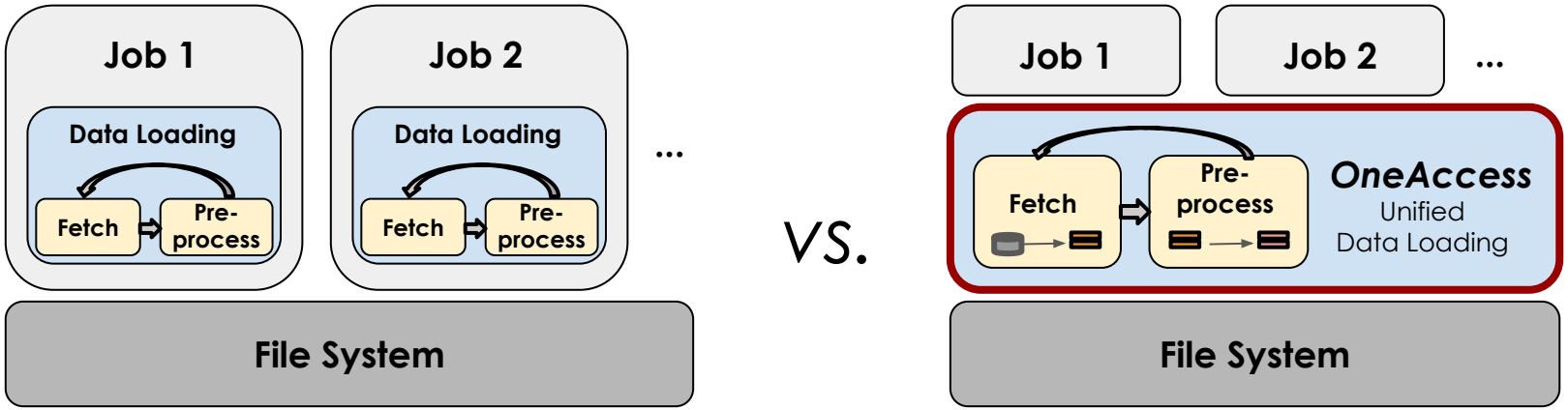


Data Loading for Machine Learning

- Data Loading can be multi-threaded or multi-process
- Runs asynchronously from training
- Requires random accesses to generate batches
  - Randomness leads to faster convergence of the training process
  - Undesirable for HDDs, SSDs



The Case for Unifying Data Loading



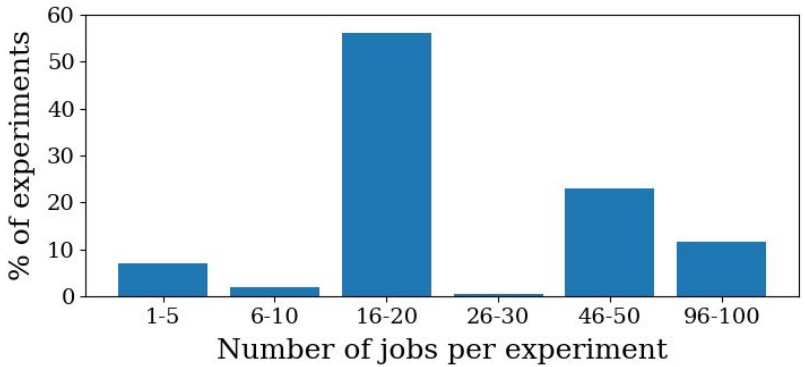
- Typically, each ML job performs data loading independently
- Inefficient in the cloud
  - Multiple concurrent jobs could be accessing the same dataset
- We propose unifying data loading in a single system: OneAccess

Case Study: Potential of Unified Data Loading

- We study Hyperparameter Tuning at Microsoft for one month
  - Demonstrates the potential of a unified data loading system in the cloud
- What are Hyperparameters?
  - Hyperparameters determine the configuration of a model
  - The number of hidden layers in a neural network, learning rate for SGD, etc.
- What is Hyperparameter Tuning?
  - Training multiple configurations of a model with different hyperparameters
  - Choose the best configuration

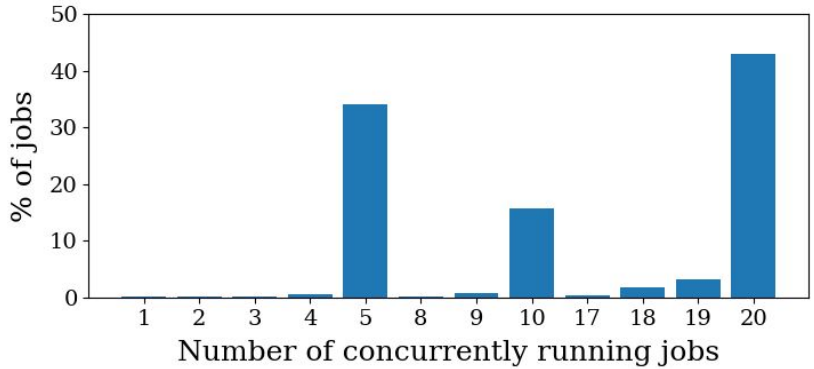
Case Study: Hyperparameter Tuning

- Hyperparameter Tuning using HyperDrive
  - Each launched experiment consists of multiple jobs
  - Each job trains a different configuration of the model on the same dataset
- On an average, each experiment has 35 jobs

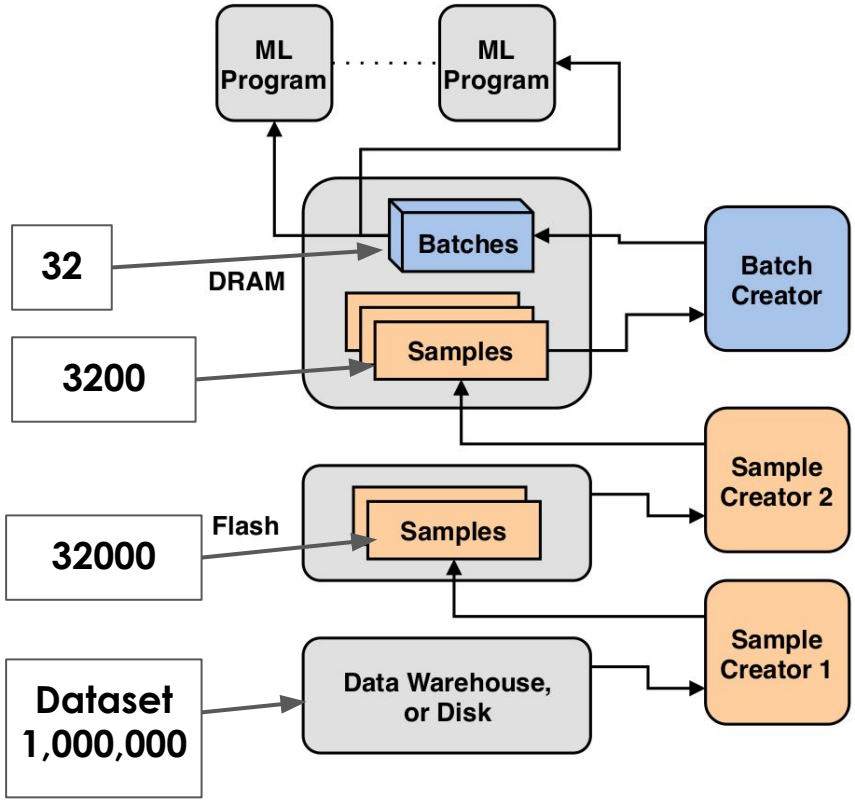


Case Study: Concurrent Tuning Jobs

- Not all of the jobs of an experiment are launched concurrently
- We observe that HyperDrive frequently launches 5, 10, or 20 jobs simultaneously, and 9 concurrent jobs on an average
- I/O time to fetch data can be reduced by (1-□) = 89% if concurrent jobs share batches



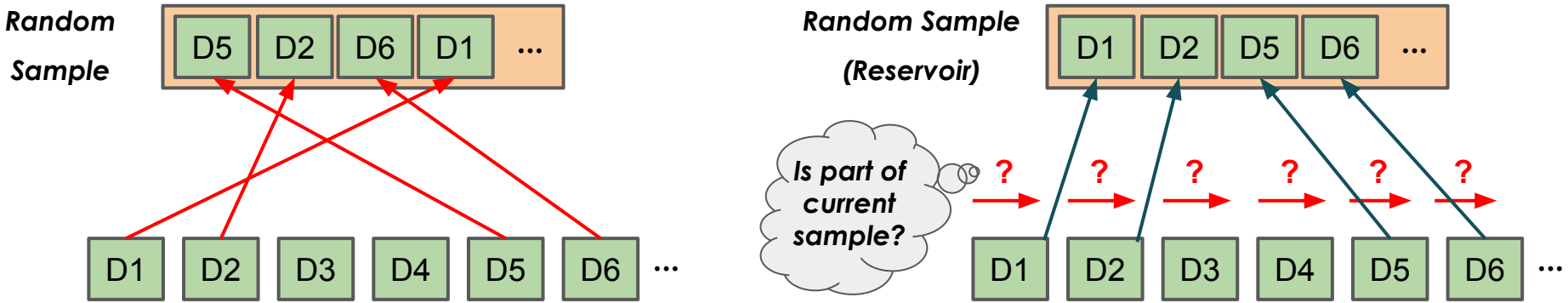
Unified Data Loading Using **OneAccess**



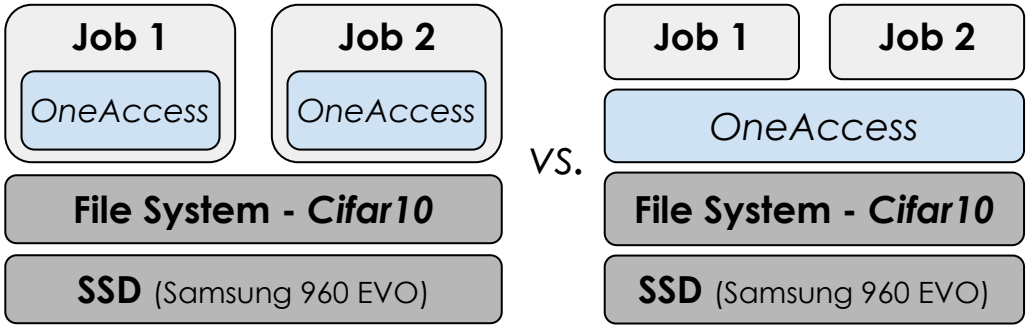
- Two types of processes for data loading:
  - Sample Creator(s), for generating reservoir samples
  - Batch Creator, for generating batches for training
  - More details in the paper
- The distinguishing aspects of OneAccess are:
  - Unified Data Loading
  - Sequential Accesses through Reservoir Sampling

Sequential Accesses with Reservoir Sampling

Random Sampling without replacement VS. **Reservoir Sampling**



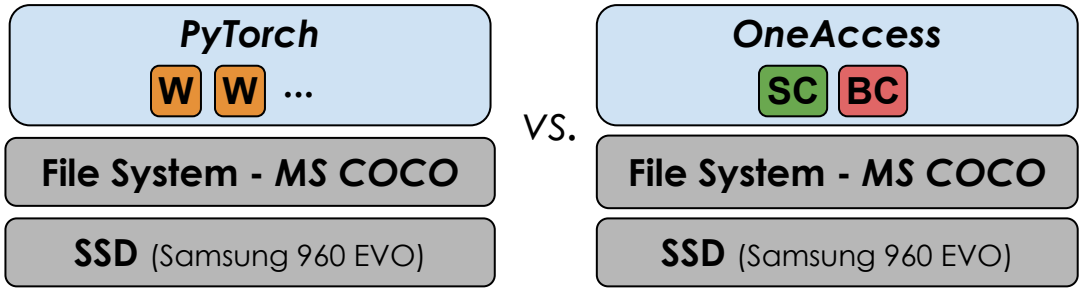
Evaluation: The Benefit of Unified Data Loading



Training Jobs/ Configuration	**Total IOPS (1 epoch)**	**I/O Time (seconds)**	
		Job 1	*Job 2*
Separate Data Access	50k × 2	14.3	15.1
Unified Data Access	50k	15.4	

- We compare the conventional approach of separate data loading, to unified data loading with OneAccess.
- As fetching data is amortized between the two jobs, we find that the total I/O time reduces by 47.3%.

Evaluation: The Benefit of Sequential Access



Framework	Total Time (min)
PyTorch (1 I/O worker)	23
PyTorch (2 I/O workers)	12
PyTorch (4 I/O workers)	6.8
OneAccess (1 Sample Creator and 1 Batch Creator)	6.4

- We compare the batch creation time of PyTorch against OneAccess for the MS-COCO dataset during one epoch.
- We find that OneAccess is 3.6x and 1.9x faster compared to PyTorch with 1 and 2 workers respectively.

Open Challenges

- Unifying data pre-processing across frameworks
  - Sharing pre-processed data across frameworks will require a standard format
- Synchronizing data access across jobs
  - The training speed of different training jobs can be different
  - Jobs can start at different times
  - Jobs can have different batch sizes
- Importance of locality and parallelism
  - It remains to be seen how effective is reservoir sampling across machines in a cluster
  - Extend reservoir sampling to use multiple workers

Conclusions

- We study data loading in popular ML frameworks
- Make a case for unifying data loading for machine learning in clusters
- We present a case study on Hyperparameter Tuning to demonstrate the potential of unified data loading in the cloud
- We develop a prototype system called OneAccess, which has
  - Unified data loading
  - Sequential accesses through reservoir sampling

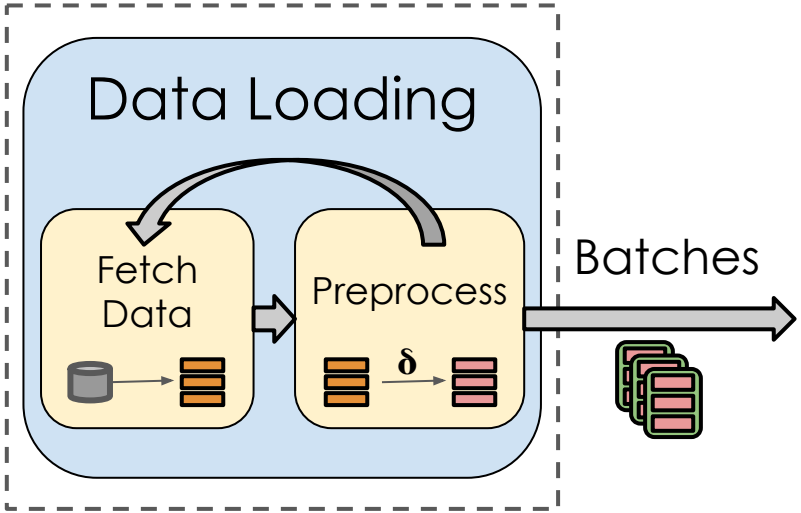
Thank You!

Aarati Kakaraparthy | aaratik@cs.wisc.edu

Machine Learning Frameworks – Two Parts

Machine Learning (ML) frameworks like PyTorch, Tensorflow, etc. typically consist of two sub-systems:

- ***Data Loading***, to generate batches of data, and
- ***Training***, to compute gradients and update the model



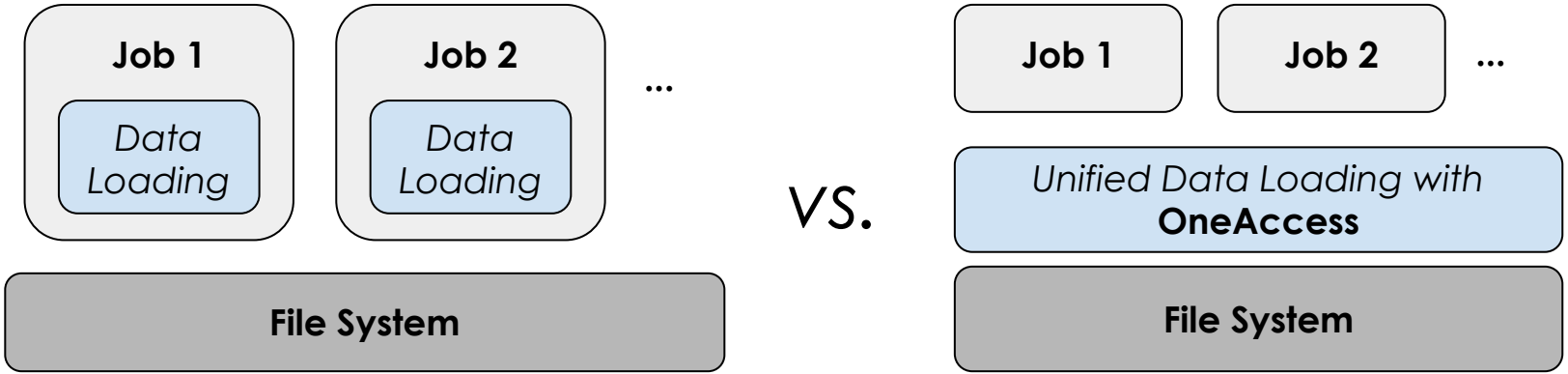
The Cost of Data Loading

- To demonstrate the cost of data loading, we train Resnet-18 on MS-COCO Dataset using PyTorch on an Nvidia GTX 1050 GPU
- Increasing the number of workers alleviates the bottleneck of data loading

Number of PyTorch Workers	Training Time	Data Loading Time	Total Time
1	416s	443s	521s
2	309s	250s, 248s	310s
4	309s	124.7s, 125.2s, 125.3s, 125.2	309.7s

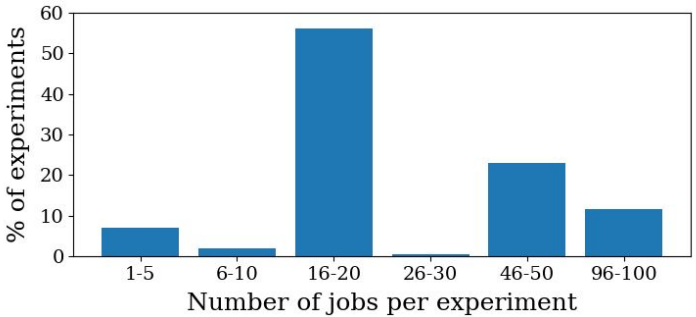
The Case for Unifying Data Loading

- Typically, each ML job performs data loading independently
- Inefficient in the cloud
  - Multiple concurrent jobs could be accessing the same dataset
- We propose unifying data access in a single system: OneAccess

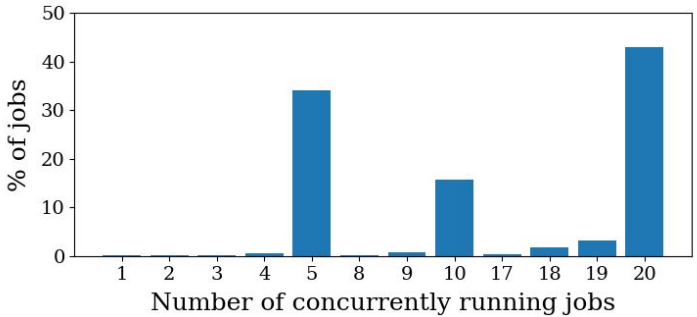


Case Study: Hyperparameter Tuning Jobs

- The potential benefit of unified data loading
  - If concurrent jobs reuse batches, the I/O cost of fetching data can be reduced by (1-□) = 89%
  - If pre-processed data is persisted, the pre-processing cost can be reduced by (1-1/35) = 97%



35 jobs per experiment on an average



9 jobs running concurrently on an average