CSC290/420 Machine Learning Systems for Efficient AI
Training - I

Sreepathi Pai

November 2, 2026

URCS

Outline

Training Overview

Auto-differentiation

Gradient Descent

Distributed Training

Outline

Training Overview

Auto-differentiation

Gradient Descent

Distributed Training

Training

- A mode of executing machine learning models distinct from inference mode
  - Used to build models
- Recall: a neural network consists of weights and biases
- Training "learns" their values for a particular task
  - Loss function measures how different the model is from an expected output
  - The value of the loss function dictates how the weights and biases get updated
- Steps
  - Initialization [random values]
  - Forward [mostly the same as inference]
  - Backward [compute the gradient and update the weights]
  - Repeat until Convergence

Initialization

- Initialization sets all weights and biases to random numbers
- Can't initialize to all zeroes
- Not considered a performance or efficiency bottleneck
- But high performance random number generation is sometimes a bottleneck
    - in Markov Chain Monte Carlo simulations, for example

Forward Pass

- Essentially, the same computation as inference
  - but need to calculate gradients as well
- Operates on a training set
  - Large, with significant storage demands (hundreds of TB for latest LLMs)
  - Inference requires less storage
- But can be optimized similar to inference

Backward Pass (Backpropagation)

- Only in training
- Propagate loss "backwards" through the neural network
  - i.e. update weights and biases so that the next forward pass will have less error (or loss)
- Two conceptual steps form this "Optimization step"
  - Gradient calculation
  - Gradient descent

Convergence

- Training is iterative
  - Repeat forward and backward passes until desired loss is achieved
- Convergence is not guaranteed

Outline

Training Overview

Auto-differentiation

Gradient Descent

Distributed Training

Gradient Calculation using Differentiation

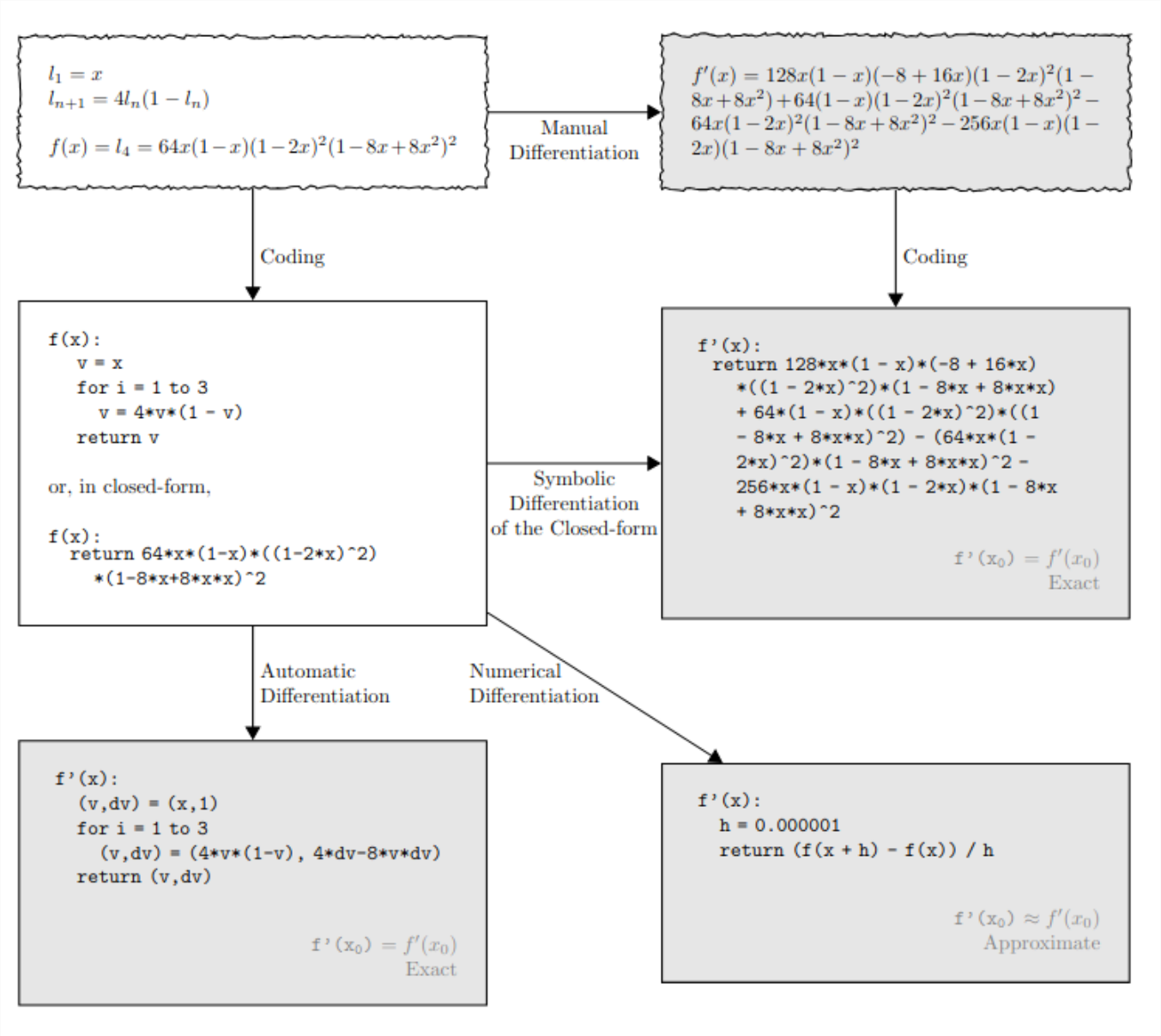
$$W_{i+1} \leftarrow W_i - \frac{\eta}{n}(\Delta E_i)$$

where ΔE_i is the gradient of the error wrt weight i

The gradient can be calculated by differentiating the computation.

Numerical, Symbolic, and Auto-Differentiation

From Baydin et al., Automatic Differentiation in Machine Learning: a Survey



The Chain Rule

$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

$$w_0 = x$$

$$w_1 = h(w_0)$$

$$w_2 = g(w_1)$$

$$w_3 = f(w_2) = y$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial f(w_2)}{\partial w_2}\frac{\partial g(w_1)}{\partial w_1}\frac{\partial h(w_0)}{\partial x}$$

From Wikipedia, Auto-Differentiation

Forward AD

Forward Primal Trace			Forward Tangent (Derivative) Trace		
v_{-1}	$= x_1$	$= 2$	$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
v_0	$= x_2$	$= 5$	$\dot{v}_0$	$= \dot{x}_2$	$= 0$
v_1	$= \ln v_{-1}$	$= \ln 2$	$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$	$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
v_3	$= \sin v_0$	$= \sin 5$	$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$	$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$	$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
y	$= v_5$	$= 11.652$	$\dot{y}$	$= \dot{v}_5$	$= \mathbf{5.5}$

Table 2 from Baydin et al., Automatic Differentiation in Machine Learning: a Survey

Forward AD using Duals

- Forward AD is implemented using a special number system
- A pair of numbers known as duals (x, x')
  - Think of x as the actual value, and x' representing the derivative
- The normal arithmetic operations are extended to handle these dual numbers
- Every value in the program is extended to track its x'
  - Extra storage space
- Forward AD yields gradients with respect to one neural network input x
  - Must be repeated n times if there multiple NN inputs $x_1, x_2, x_3, ..., x_n$

Reverse AD

Forward Primal Trace			Reverse Adjoint (Derivative) Trace			
v_{-1}	$= x_1$	$= 2$	$\bar{x}_1$	$= \bar{v}_{-1}$		$= \mathbf{5.5}$
v_0	$= x_2$	$= 5$	$\bar{x}_2$	$= \bar{v}_0$		$= \mathbf{1.716}$
v_1	$= \ln v_{-1}$	$= \ln 2$	$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$	$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
			$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
v_3	$= \sin v_0$	$= \sin 5$	$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
v_4	$= v_1 + v_2$	$= 0.693 + 10$	$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
			$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$	$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
			$\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
y	$= v_5$	$= 11.652$	$\bar{v}_5$	$= \bar{y}$	$= 1$	

Table 3 from Baydin et al., Automatic Differentiation in Machine Learning: a Survey

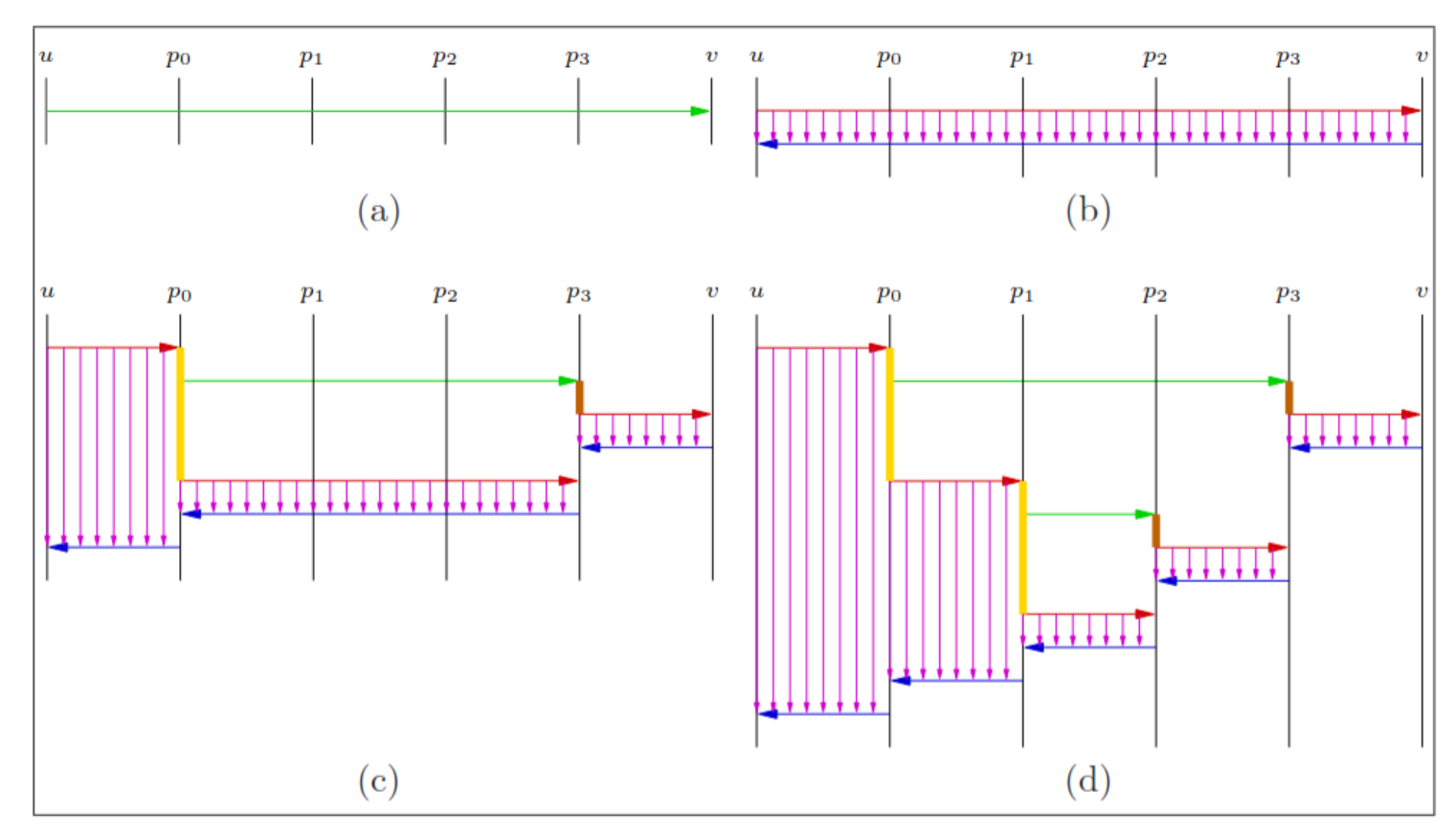
Reverse AD Implementation

- Store computed values to a "tape"
  - until end of computation
  - required to compute gradients
- However, only one pass required even for multiple NN inputs
  - Storage vs compute overhead

Overcoming Storage overheads

- Active area of research
- Recomputation
  - sometimes called rematerialization
  - recompute values instead of storing them in memory
- Checkpoint
  - usual meaning: store state of program to restart from this point
  - In ML, similar meaning but more intertwined with AD

Checkpointing, Recomputation, and AD



From Siskind and Pearlmutter, Divide-and-Conquer Checkpointing for Arbitrary Programs with No User Annotation

Outline

Training Overview

Auto-differentiation

Gradient Descent

Distributed Training

Stochastic Gradient Descent

- Gradient descent requires computing gradients for all inputs
- And averaging them before updating weights
- Stochastic gradient descent only uses one input picked randomly

Parallel Stochastic Gradient Descent

- Compute gradients as usual on one input
- However, compute multiple updates in parallel
- Aggregate those updates at the same time
    - Use mutual exclusion to aggregate in parallel

Hogwild!

Stochastic Gradient Descent (SGD) is a popular algorithm that can achieve state-of-the-art performance on a variety of machine learning tasks. Several researchers have recently proposed schemes to parallelize SGD, but all require performance-destroying memory locking and synchronization. This work aims to show using novel theoretical analysis, algorithms, and implementation that SGD can be implemented without any locking. We present an update scheme called Hogwild which allows processors access to shared memory with the possibility of overwriting each other's work. We show that when the associated optimization problem is sparse, meaning most gradient updates only modify small parts of the decision variable, then Hogwild achieves a nearly optimal rate of convergence. We demonstrate experimentally that Hogwild outperforms alternative schemes that use locking by an order of magnitude.

Abstract of Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent

Outline

Training Overview

Auto-differentiation

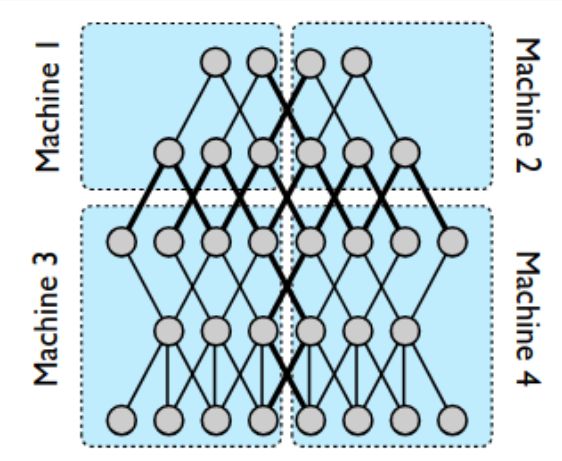
Gradient Descent

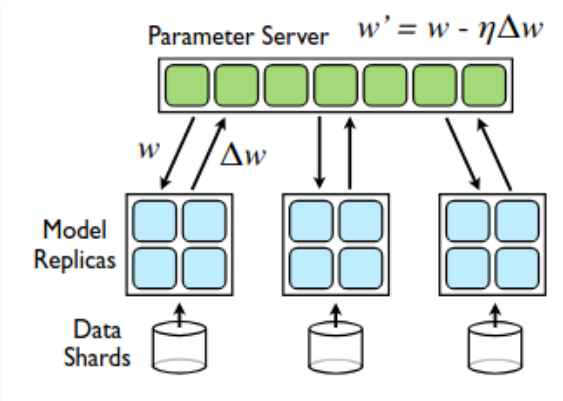
Distributed Training

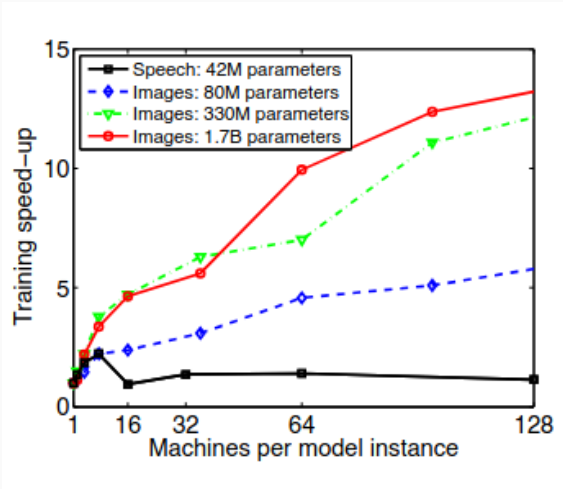
Machines

- AlexNet - two NVIDIA GTX 580 GPUs
  - ImageNet size - 130GB to 1.31TB
  - one machine?
- ChatGPT 3.5 - 1024 A100 GPUs (?)
  - Text size upto hundreds of TB
  - definitely not one machine

DistBelief (2013)

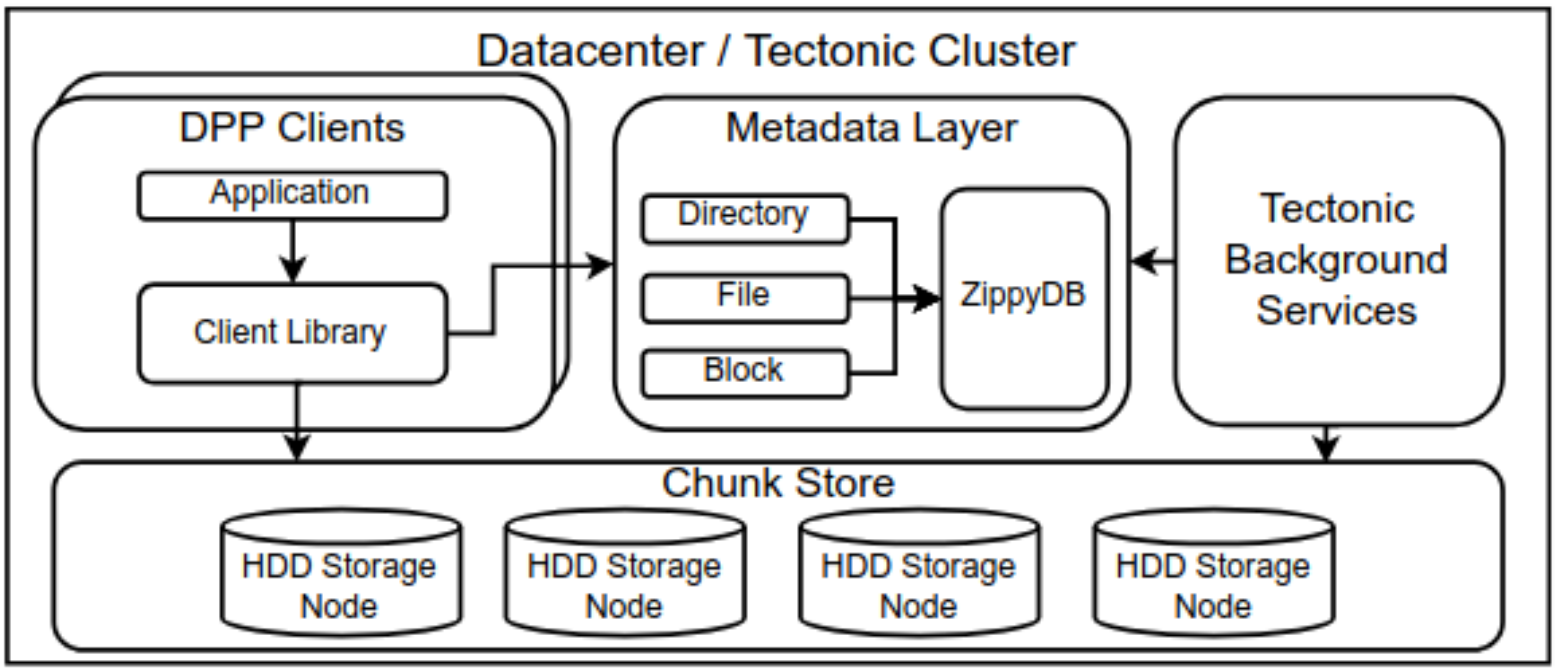






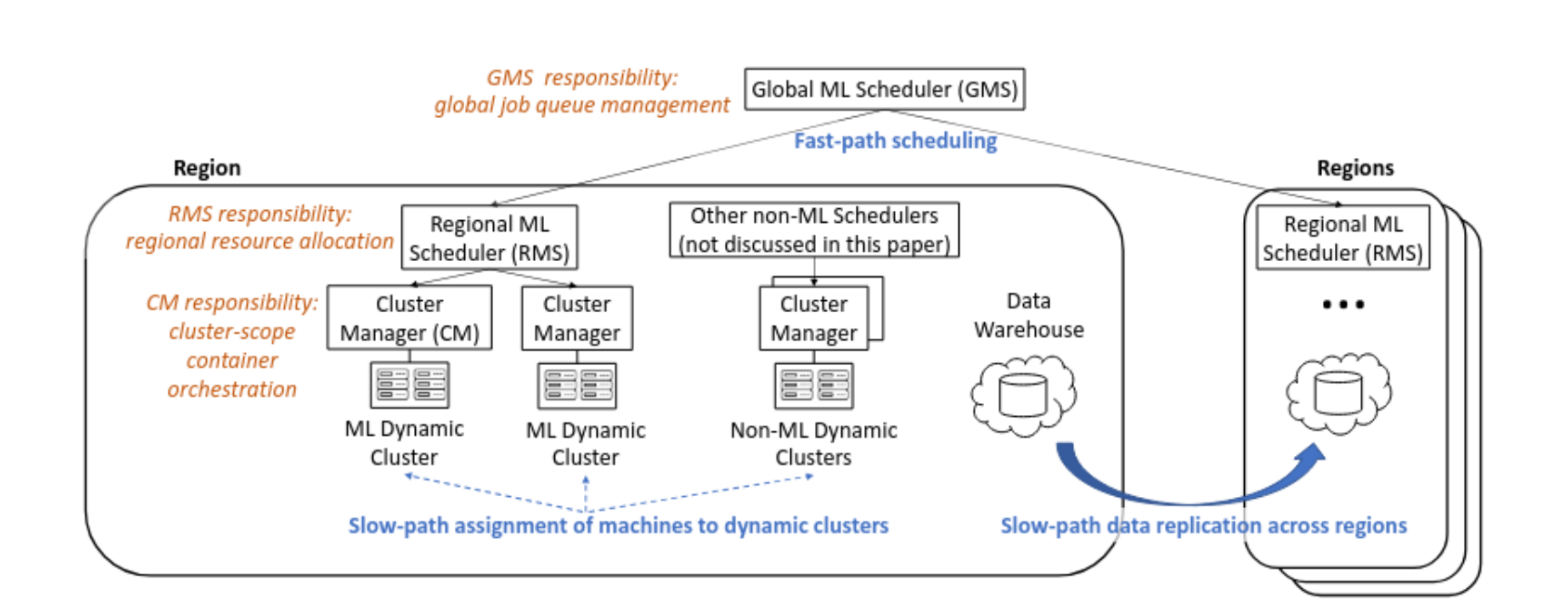
Dean et al., Large Scale Distributed Deep Networks

Storage (Meta's Tectonic)



Zhao et al., Tectonic-Shift: A Composite Storage Fabric for Large-Scale ML Training, USENIX ATC 23

Scheduling (Meta's MAST)



Choudhary et al., MAST: Global Scheduling of ML Training across Geo-Distributed Datacenters at Hyperscale, OSDI 2024