CS 425 / ECE 428
Distributed Systems
Fall 2026

Aishwarya Ganesan & Ram Kesavan

Lecture 26: Graph Processing, Machine Learning

Ack: Indranil Gupta

All slides © IG

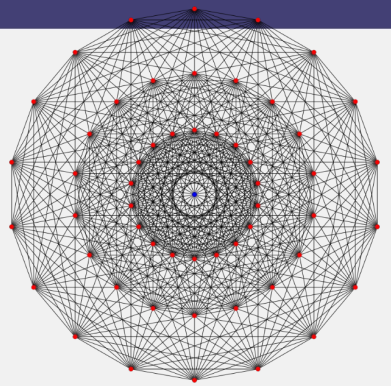
Graph Processing: What We'll Cover

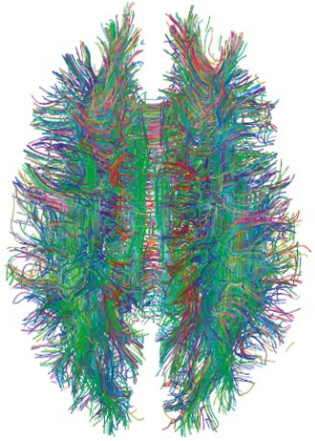
- Distributed Graph Processing
- Google's Pregel system
  - Inspiration for many newer graph processing systems: Piccolo, Giraph, GraphLab, PowerGraph, LFGraph, X-Stream, etc.



Lots of Graphs

- Large graphs are all around us
  - Internet Graph: vertices are routers/switches and edges are links
  - World Wide Web: vertices are webpages, and edges are URL links on a webpage pointing to another webpage
    - Called “Directed” graph as edges are uni-directional
  - Social graphs: Facebook, Twitter, LinkedIn
  - Biological graphs: Brain neurons, DNA interaction graphs, ecosystem graphs, etc.





Source: Wikimedia Commons, Wikipedia

Graph Processing Operations

- Need to derive properties from these graphs
- Need to summarize these graphs into statistics
- E.g., find shortest paths between pairs of vertices
  - Internet (for routing)
  - LinkedIn (degrees of separation)
- E.g., do matching
  - Dating graphs in match.com (for better dates)
- PageRank
  - Web Graphs
  - Google search, Bing search, Yahoo search: all rely on this
- And many (many) other examples!



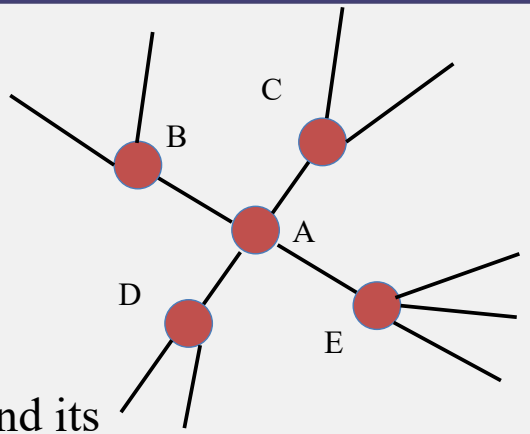
Why Hard?

- Because these graphs are large!
  - Human social network has 100s Millions of vertices and Billions of edges
  - WWW has Millions of vertices and edges
- Hard to store the entire graph on one server and process it
  - On one beefy server: may be slow, or may be very expensive (performance to cost ratio very low)
- Use distributed cluster/cloud!
  - But walking a graph requires message passing



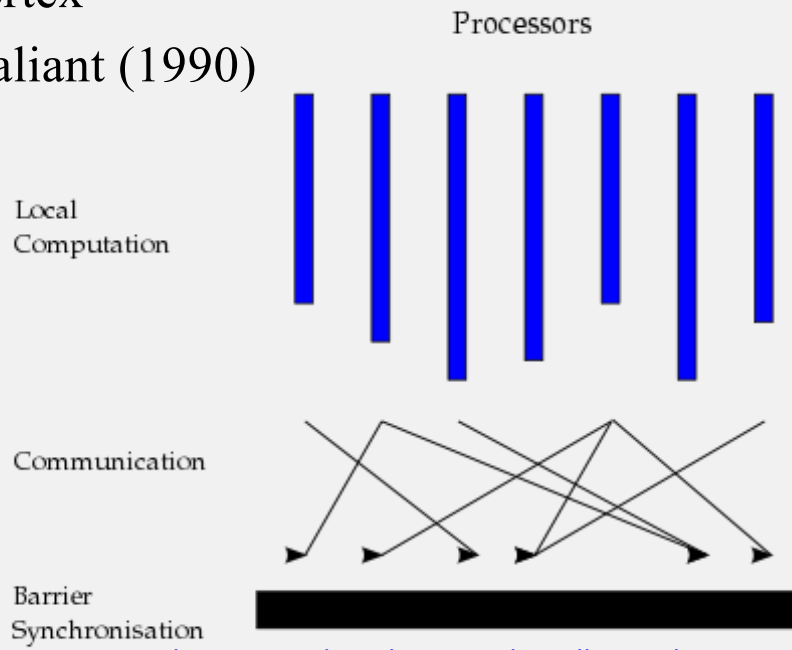
Typical Graph Processing Application

- Works in *iterations*
- Each vertex assigned a *value*
- In each iteration, each vertex:
  1. **Gather**: Gathers values from its immediate neighbors (vertices who join it directly with an edge). E.g., @A: B→A, C→A, D→A,…
  2. **Apply**: Does some computation using its own value and its neighbors' values.
  3. **Scatter**: Updates its new value and sends it out to its neighboring vertices. E.g., A→B, C, D, E
- Graph processing terminates after: i) fixed iterations, or ii) vertices stop changing values



Bulk Synchronous Parallel Model

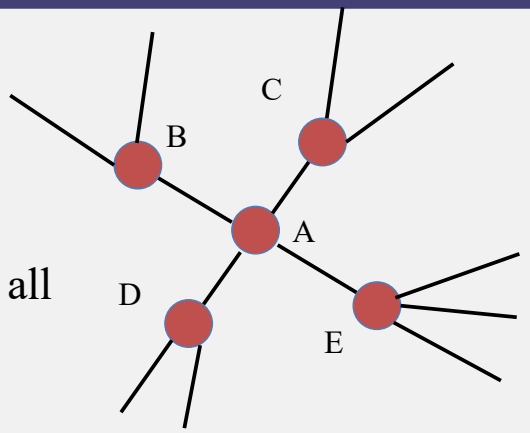
- "Think like a vertex"
- Originally by Valiant (1990)



Source: http://en.wikipedia.org/wiki/Bulk_synchronous_parallel

Basic Distributed Graph Processing

- "Think like a vertex"
- Assign each vertex to one server
- Each server thus gets a subset of vertices
- In each iteration, each server performs **Gather-Apply-Scatter** for all its assigned vertices
  - Gather: get all neighboring vertices' values
  - Apply: compute own new value from own old value and gathered neighbors' values
  - Scatter: send own new value to neighboring vertices
- Races? Not, unless graph structure is modified



Assigning Vertices

- How to decide which server a given vertex is assigned to?
- Different options
  - Hash-based: Hash(vertex id) modulo number of servers
    - Remember consistent hashing from P2P systems?!
  - Locality-based: Assign vertices with more neighbors to the same server as its neighbors
    - Reduces server to server communication volume after each iteration
    - Need to be careful: some "intelligent" locality-based schemes may take up a lot of upfront time and may not give sufficient benefits!



Pregel System By Google

- Pregel uses the leader/worker model
  - Leader (one server)
    - Maintains list of worker servers
    - Monitors workers; restarts them on failure
    - Provides Web-UI monitoring tool of job progress
  - Worker (rest of the servers)
    - Processes its vertices
    - Communicates with the other workers
- Persistent data is stored as files on a distributed storage system (such as GFS or BigTable)
- Temporary data is stored on local disk

Pregel Execution

1. Many copies of the program begin executing on a cluster
2. The leader ("Master" originally) assigns a partition of input (vertices) to each worker
   - Each worker loads the vertices and marks them as *active*
3. The leader instructs each worker to perform an iteration
   - Each worker loops through its active vertices & computes for each vertex
   - Messages can be sent whenever, but need to be delivered before the end of the iteration (i.e., the barrier)
   - Worker informs leader after completing an iteration; sends #active vertices for next iteration, and any aggregator values
   - When all workers reach iteration barrier, leader starts next iteration
4. Computation halts when, in some iteration: no vertices are active and when no messages are in transit



Fault-Tolerance in Pregel

- **Checkpointing**
  - Periodically, at start of an iteration, leader instructs the workers to save state of their partitions to persistent storage
    - e.g., Vertex values, edge values, incoming messages
- **Failure detection**
  - Using periodic "ping" messages from leader → worker
- **Recovery**
  - The leader reassigns graph partitions to the currently available workers
  - The workers all reload their partition state from most recent available checkpoint



How Fast Is It?

- Shortest paths from one vertex to all vertices
  - SSSP: “Single Source Shortest Path”
- On 1 Billion vertex graph (tree)
  - 50 workers: 180 seconds
  - 800 workers: 20 seconds
- 50 B vertices on 800 workers: 700 seconds (~12 minutes)
- Pretty Fast!

Summary: Graph Processing

- Lots of (large) graphs around us
- Need to process these
- MapReduce not a good match
- Distributed Graph Processing systems: Pregel by Google
- Many follow-up systems
  - Piccolo, Giraph: Pregel-like
  - GraphLab, PowerGraph, LFGraph, X-Stream: more advanced

Polls



Basic ML: SGD

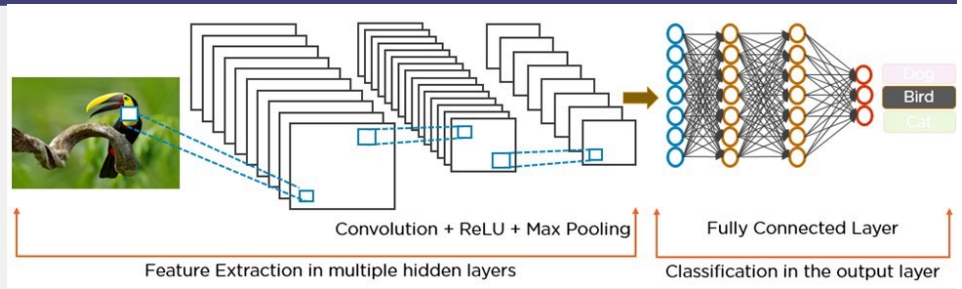
"Machine learning is nothing but damn statistics." – A lot of people

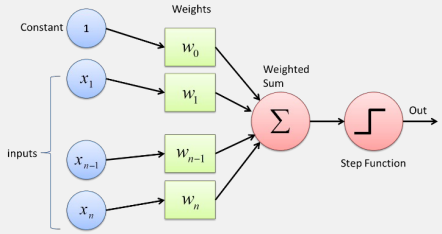
- Machine learning trains "models" (computer representations)
- Training vs. Inference (latter called Prediction, or Model Serving)
- Learning: Supervised vs. Unsupervised vs Reinforcement
- SGD = Stochastic Gradient Descent
  - Minimize an objective function that is smooth and differentiable
  - Popular variants: AdaGrad (adaptive gradient), Adam (adaptive moment), RMSProp
  - A common strawman application for many distributed ML papers!



Basic Neural Networks

- ANN = Artificial Neural Network
  - Another kind of "model"
  - A common form of representation learning
  - Used widely in vision, NLP, …
- Graphs of operators
- Operators can be computationally heavy
- Graph can be in "stages" or "layers"





A Perceptron layer

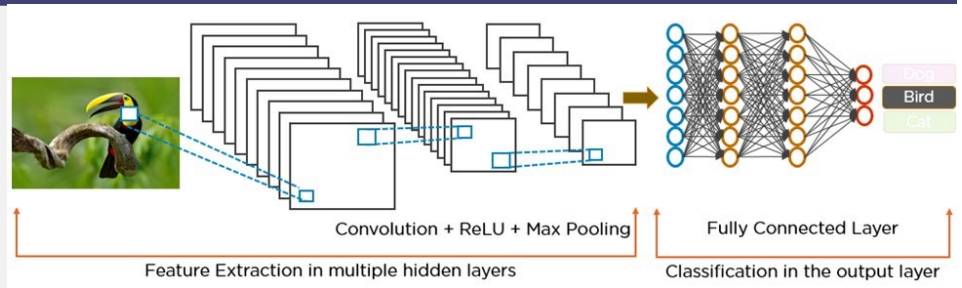
- Common: All to all communication between operators in consecutive stages
- Most edges have a set of associated weights (parameters). (Exceptions: activation funcs (ReLU), dropout). The collection of these weights IS the model.

Basic Neural Networks (2)

- Training data passed "forward", followed by calculating error against known result (supervised learning)
  - followed by "backward" pass that adjusts/updates the weights at operators (so that for an incorrect training pass, if the same input were to be passed through, the result would be correct)
- Data between operators: tensors (multi-dimensional vectors)
- Default: train on one item (forward + backward), followed by next item
  - Extension: train on a mini-batch of items
- After model is trained, Prediction/Inference needs only forward pass
- Emerging area: Online training : do both training and inference together (aka Continual training)
- Hyperparameter: configuration parameter for your model (not to be confused with a model "weight" == parameter), e.g., batch size



ANN Types: FFNN, CNNs, RNNs



- FFNN: feed forward neural net (no loops)
- DNN: Deep NN
  - More than one stage
  - "Hidden" layers between input and output
- CNN: Convolutional NN
  - DNN + additional layers for convolutions
  - Transform data, e.g., sequence of filters that result in an activation/detection, e.g., image.
  - E.g., Facebook photo captioning
- RNN: Recurrent NN
  - DNN + loops within layer (e.g., time aspect (e.g., GIF), sequential aspects)
  - E.g., auto-correction on your phone
- Other types: GNN (graph neural net), Transformer, …
- A few neural nets used in evaluation of distributed machine learning
  - Inception(v3), (G)NMT, Resnet,…

Distributed Machine Learning

Training data – collection of input and output labels x_i, y_i of size N

Learn model weights w

Model size and N can be huge

Centralized Machine Learning

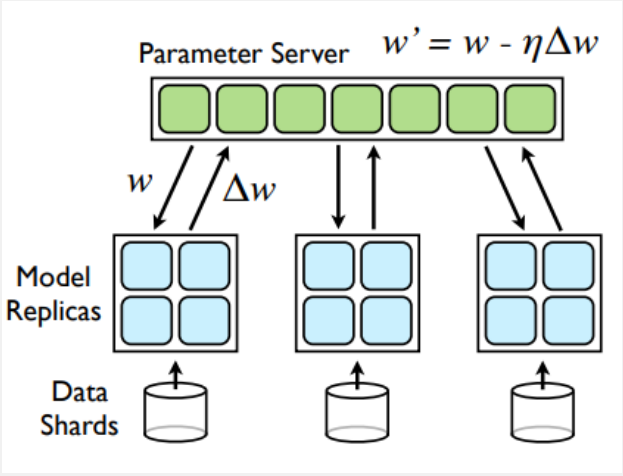
Slow and often infeasible

Distributed Machine Learning

Parallelize via Multiple workers

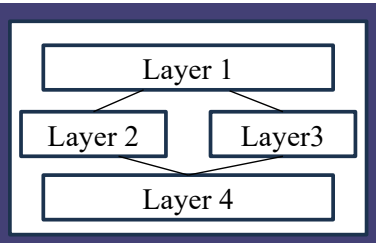
"Parameter Server" to aggregate data from workers

from current iteration and start next iteration at workers.



Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in Neural Information Processing Systems (NeurIPS) 25 (2012).

Kinds of Parallelism

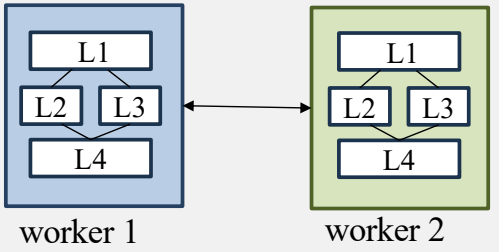


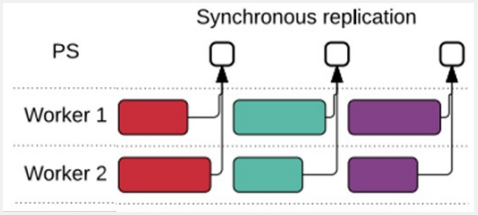
Data parallelism: multiple workers run same model, different data is sent to each device, data "batched" into mini-batches

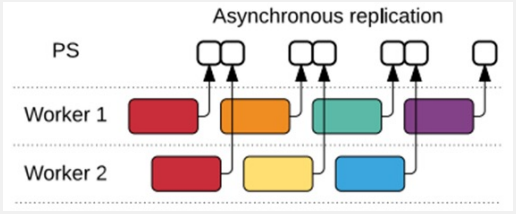
Workers synchronize after each mini-batch

1. Parameter Server approach
2. All-Reduce approach: each worker multicasts weights to others

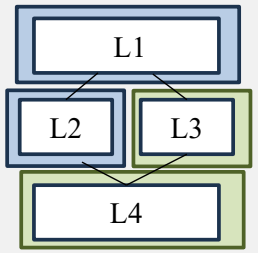
Variant: Asynchronous training







Model parallelism: same model (DNN graph) is split across multiple devices, one input passed through collection of devices at a time.



Abadi, Martín, et al. "TensorFlow: a system for Large-Scale machine learning." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.

TensorFlow

Built by Google

Framework for large-scale training and inference

Hides details of distribution

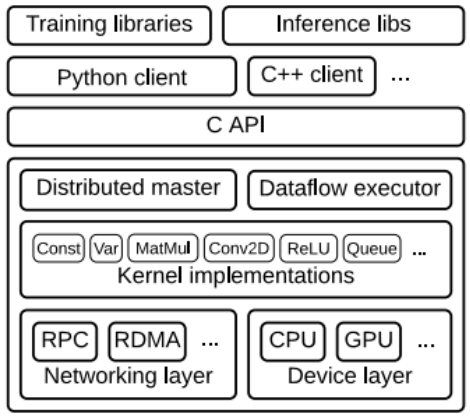
Uses dataflow graphs to represent computation, shared state, and operations that mutate state

Dataflow captures structure of computation in ML

Extensible, runtime contains over 200 standard operations

Support for CPUs, GPUs, and TPUs (Tensor Processing Units)

Different communication protocols



Abadi, Martín, et al. "TensorFlow: a system for Large-Scale machine learning." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.

Image classifier using TensorFlow API

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784]) # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10]) # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100])) # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100])) # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2) # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10])) # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10])) # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2 # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess: # Connect to the TF runtime.
     sess.run(tf.initialize_all_variables()) # Randomly initialize weights.
     for step in range(NUM_STEPS): # Train iteratively for NUM_STEPS.
            x_data, y_data = ... # Load one batch of input data.
            sess.run(train_op, {x: x_data, y: y_data}) # Perform one training step.
```

Abadi, Martín, et al. "TensorFlow: a system for Large-Scale machine learning." 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016.



PyTorch

Another popular ML framework originally developed by Meta
Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers
Dynamic computation graphs
Automatic computation of gradients
Implements many algorithms and components
Domain-specific – TorchText, TorchVision, and TorchAudio – include datasets
Tensors – n-dimensional arrays
Module – define what makes up the model and a forward member function.
E.g., Linear Module
    weight and bias as parameters
    forward function generates output as input * weight + bias



Example with DistributedDataParallel

```
dist.init_process_group("gloo", rank=rank, world_size=world_size) # create default process group

model = nn.Linear(10, 10).to(rank) # create local model
ddp_model = DDP(model, device_ids=[rank]) # construct DDP model

# define loss function and optimizer
loss_fn = nn.MSELoss()
optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)

# forward pass
outputs = ddp_model(torch.randn(20, 10).to(rank))
labels = torch.randn(20, 10).to(rank)

# backward pass
loss_fn(outputs, labels).backward()

# update parameters
optimizer.step()
```



Jax

- ML framework from Google
- Provides a familiar NumPy-style API
- Multiple backends, including CPU, GPU, & TPU
- grad: automatic differentiation
- jit: compilation
- vmap: auto-vectorization
- pmap: automatic parallelization



Other ML

Distributed ML

assumes homogenous data across workers

Federated Machine Learning

allows heterogeneous data across workers

workers may be datacenters or mobile devices

failures, possibly privacy issues.

E.g., Google's Federated learning to predict keystrokes from mobile devices



Summary of this Lecture, and one more video to watch!

Emerging topics in Distributed Computing

- Nov6th: Stream Processing (in syllabus)
- Graph Processing (in syllabus)
- Machine Learning (in syllabus)

- **Additional Video on Course website (In Syllabus, NOT optional): Spark**



Announcements

- HW4 due this Thursday 12/4 at 2 pm US Central
- MP4 due this Sunday (12/7), demos next Monday (12/8)

