facebook

INFRASTRUCTURE

Distributed Training @ Facebook

Mohamed Fawzy, Kutta Srinivasan
AI Infrastructure

Agenda

- ML @ Facebook scale
- The role of Distributed Training
- Challenges & Solutions

ML @ FB Scale
(Mohamed’s slides go in this section)

Why Distributed Training?

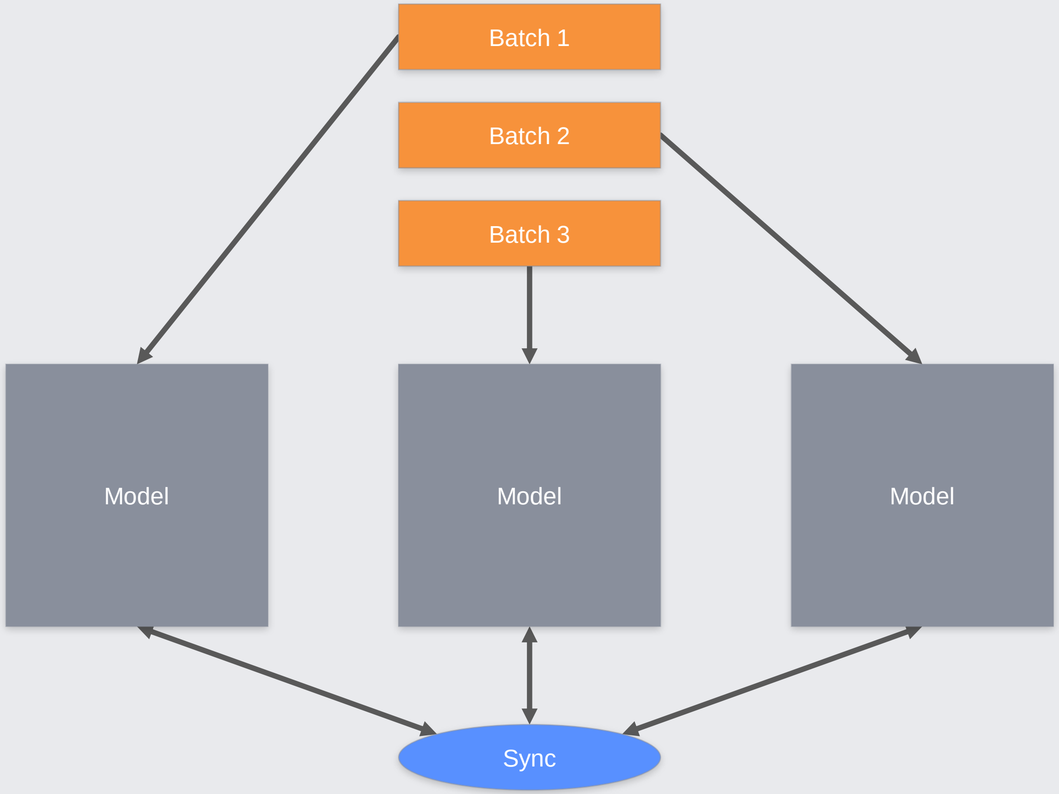
Improve ML Productivity

- Complex models train on **multi-PB** datasets
- Would take **years** to run on single machine
- Data-parallelism to the rescue

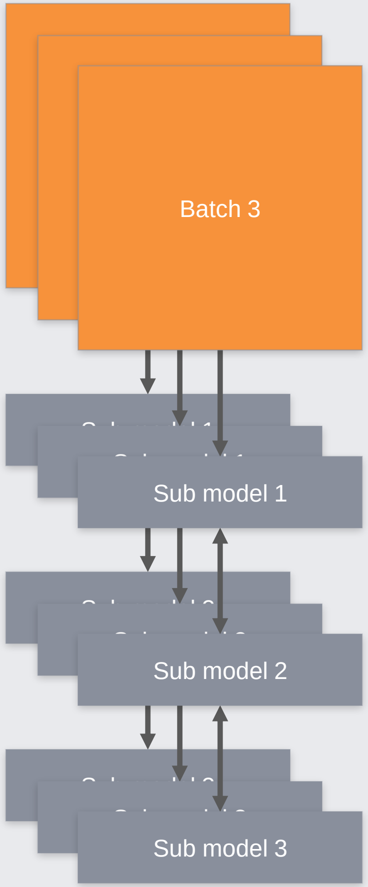
Support Huge Scale

- Sparse architectures for ranking, personalization, language
- Range from **100s of GB** → **TBs** per model
- **Both** model- and data-parallelism required

Data Parallelism



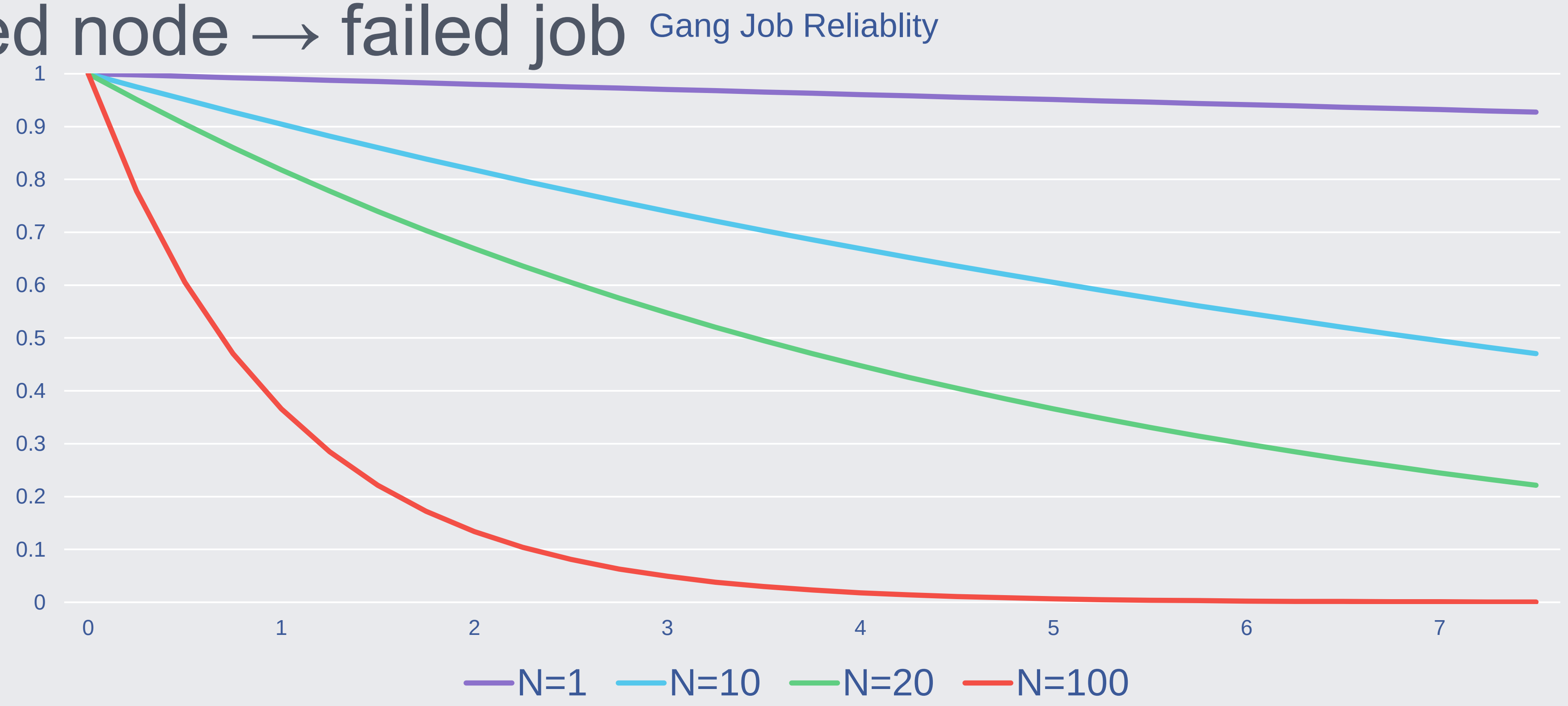
Model Parallelism
+ Data Parallelism



Distributed Training is HARD

Inherently less reliable

- **Gang scheduling** means resources are required **all-or-nothing**
- Failed node → failed job



Heterogeneous Hardware

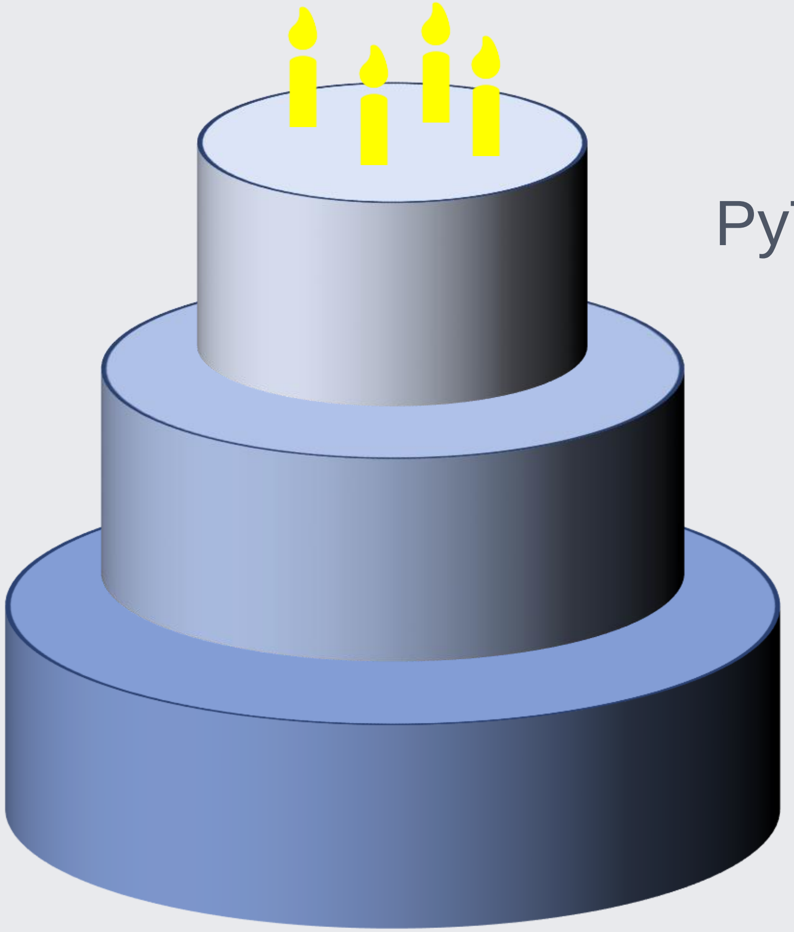
- HPC workloads sensitive to hardware types, generations
  - Scheduling more complex than just {x GB RAM, y CPU} per task
  - Multiple gen GPUs, CPUs, ASICs

Expensive & Experimental

- Unlike data pipelines, majority of ML jobs are ad hoc
    - Long running jobs complicate demand prediction & control
- Cost & efficiency
    - ROI for jobs hard to estimate
    - Sub-linear scaling + huge scale = easy to waste resources

Affordable Productivity is the Goal

A Layered Solution



PyTorch Elastic Distributed Training

ML-Aware Cluster Scheduling

Elastic Compute (Spot Instances)

PyTorch Elastic Distributed Training

- Fault tolerance for failed nodes
  - For transient errors, re-sync workers and keep going
  - **Jobs don't need baby-sitting**
- Auto-scaling
  - Start fewer nodes under resource contention, adjust hyper params[1][2]
  - Eliminate bottlenecks, improve utilization

[1] https://arxiv.org/abs/1706.02677 [2] https://openreview.net/pdf?id=B1Yy1BxCZ

Elastic Training Pseudocode

```
while not finished:
  # discover peers, use rank and size to update model hyperparams
  rank, size = rendezvous(min_nodes, max_nodes)
  sync_model(rank, size) # most tenured worker broadcasts state
  while not finished:
    try:
      train_step() # forward/backward pass + allreduce
    except TransientError:
      break # allreduce will raise if any worker fails
    if detect_new_workers():
      break # allow job to scale up if new workers arrive
```

ML-Aware Cluster Scheduling

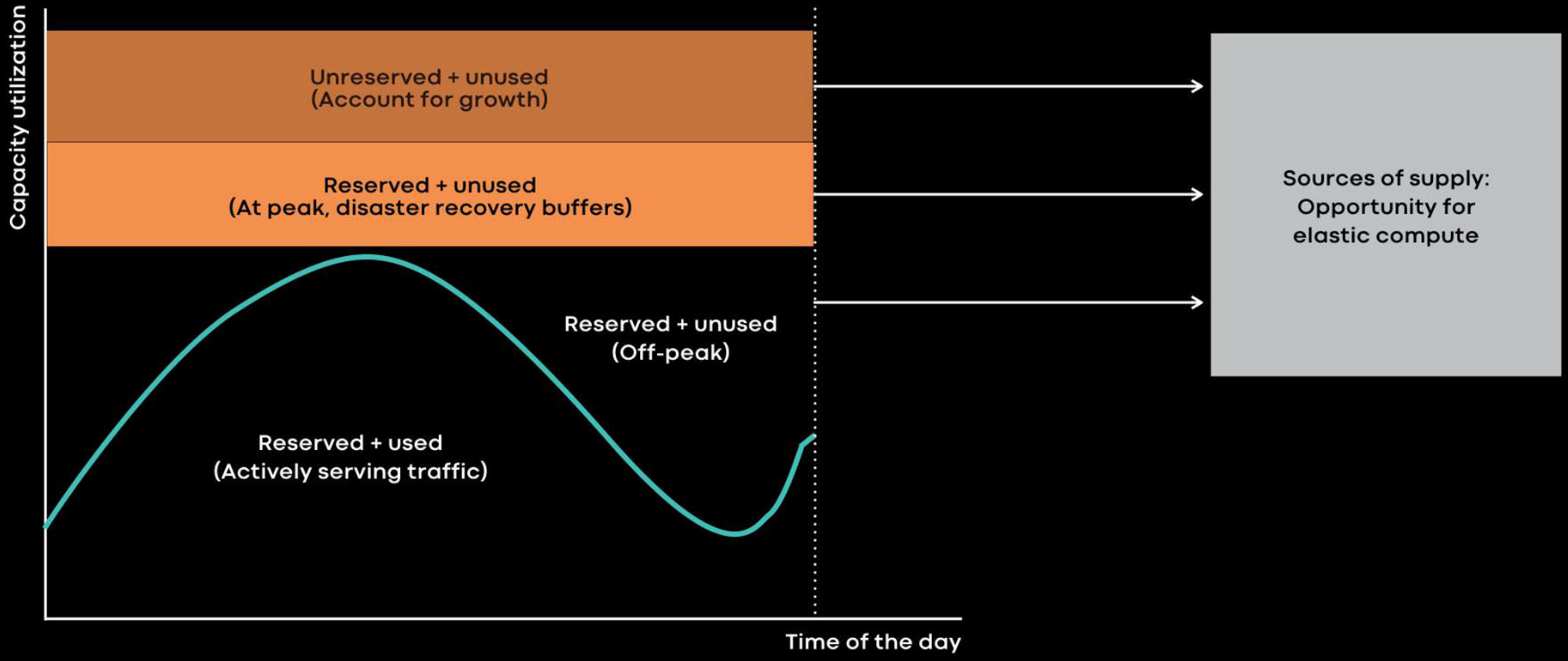
- Maximize **throughput** & **utilization**
  - Subject to quota & priority constraints
  - Allow users to **borrow** unused resources; **evict** to reclaim
- Ongoing work
  - Gang-awareness for draining, preemption
  - **Time-slicing** jobs for improved fairness
  - Globally **federated** scheduling for efficiency & DR

Elastic Compute

- FB uses power-efficient servers w/autoscaling[3]
  - Significant capacity available off-peak
- Idle machines for growth & DR buffers
- Think “Spot Instances”

[3] https://engineering.fb.com/data-center-engineering/tupperware/

Sources for Elastic Compute



Sources of Elastic Capacity [OPTIONAL]

Buffer capacity

ML goes here!

Off-peak idle capacity

Actively used capacity

Time →

Elastic Compute Challenges

- Machines reclaimed at anytime by donor service
- End-to-end job latency higher on off-peak
- Primarily useful for CPU workloads

Conclusion

- Distributed Training key to pushing state-of-the-art ML modeling
- “Affordable Productivity” a key focus of AI Infrastructure @ FB
- Investments required across entire stack

facebook

Thank you

mfawzy@fb.com
kutta@fb.com