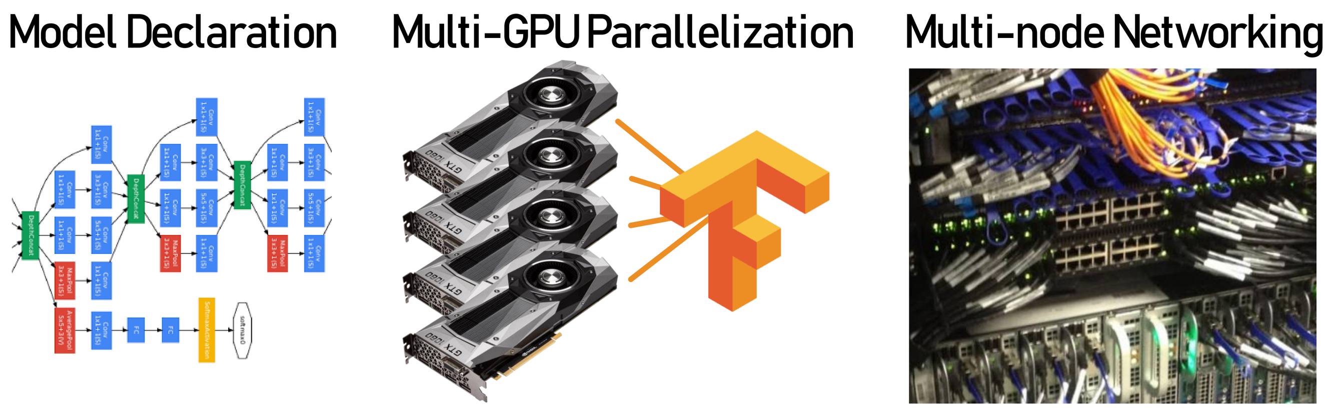
Efficient Resource Sharing for Distributed Deep Learning

Changho Hwang, Taehyun Kim, Kyuho Son, Jinwoo Shin, and KyoungSoo Park
School of Electrical Engineering, KAIST

Distributed Deep Learning over Shared GPU Cluster

- Distributed Deep Learning (DL) Job
  - Data-parallel training with multiple networked GPUs



- Legacy Cloud Resource Managers for DL Jobs

  MESOS · hadoop YARN · kubernetes

  - Static resource allocation: prioritize first-coming jobs
  - Badly affect avg. job completion time (JCT) & makespan especially for DL jobs! (reasons follow)

✓ Why Dynamic Resource Allocation?

Sublinear scaling of DL job throughput
- Static allocation incurs cluster-wise inefficiency

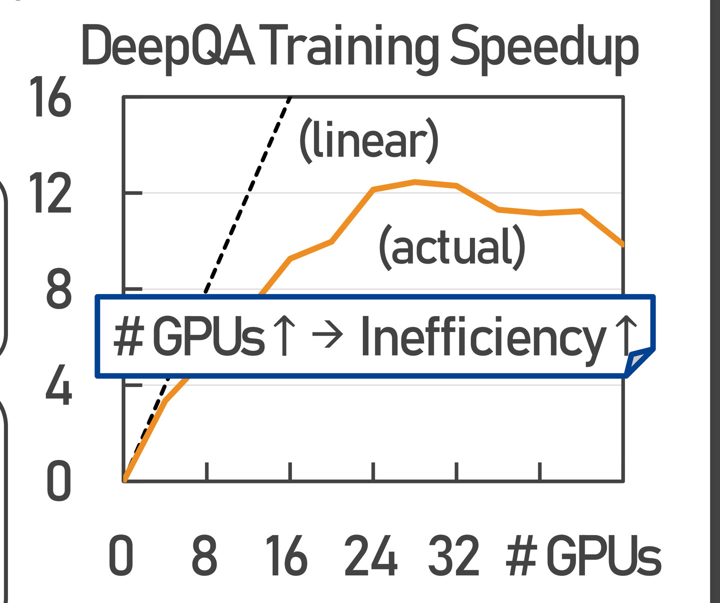
DL job typically runs for a long time
- Suboptimal resource utilization is detrimental to overall throughput

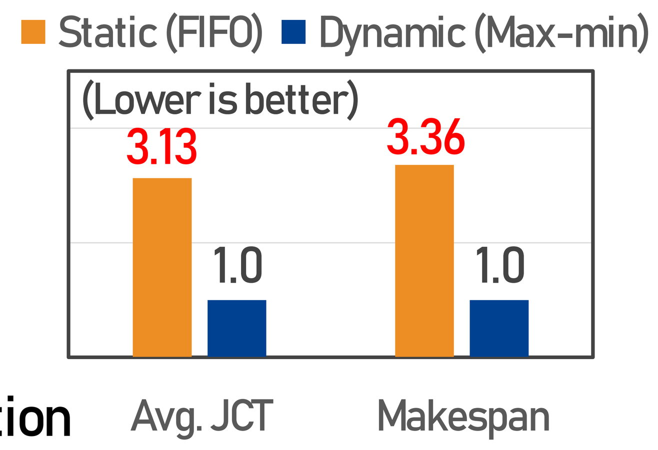
⇨ Dynamic resource re-distribution helps avg. JCT & makespan

See also: Optimus (Y. Peng et al., EuroSys '18)

- Experiment: online job submission simulation
  - Submit 512 randomly selected jobs among 8 different DL models including CNN, RNN, & GAN in 512-GPU cluster
  - Compare avg. JCT & makespan with static or dynamic allocation

2 x Intel Xeon E5-2630 v4 + 4 x NVIDIA GTX 1080 per node
40 Gbps RoCEv2 w/ Mellanox ConnectX-4 NIC, batch size 1K



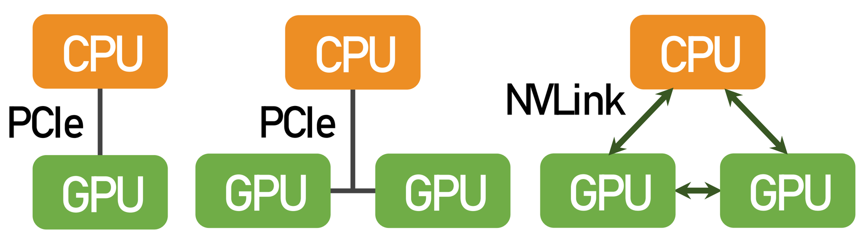


GOAL: Efficient Design of Dynamic Resource Management System for DL Training Jobs

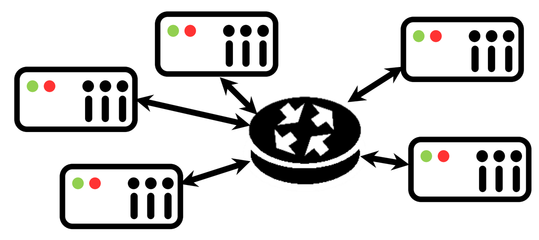
Switch from Static to Dynamic Allocation: Challenges

Limitations of Legacy Systems	User's Perspective: Complex & Hard to Optimize	System's Perspective: Poor Controllability
User app manages its own resources	Need to manage dynamic & distributed devices	System analysis & optimization rely on users

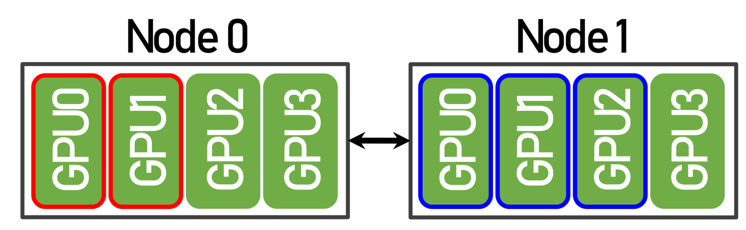
✓ Our Approach: separate dynamic resource management from user apps & automate it by the system



Multi-GPU HW Topology Consideration
- Stronger CPU-GPU interconnection → update more params in GPU
- Stronger GPU-GPU interconnection → merge more gradients in GPU



Parameter Update Load Balancing
- Exact 1/N division: re-division overhead when # of nodes changes
- Determine a proper unit size of params according to the HW link capacity

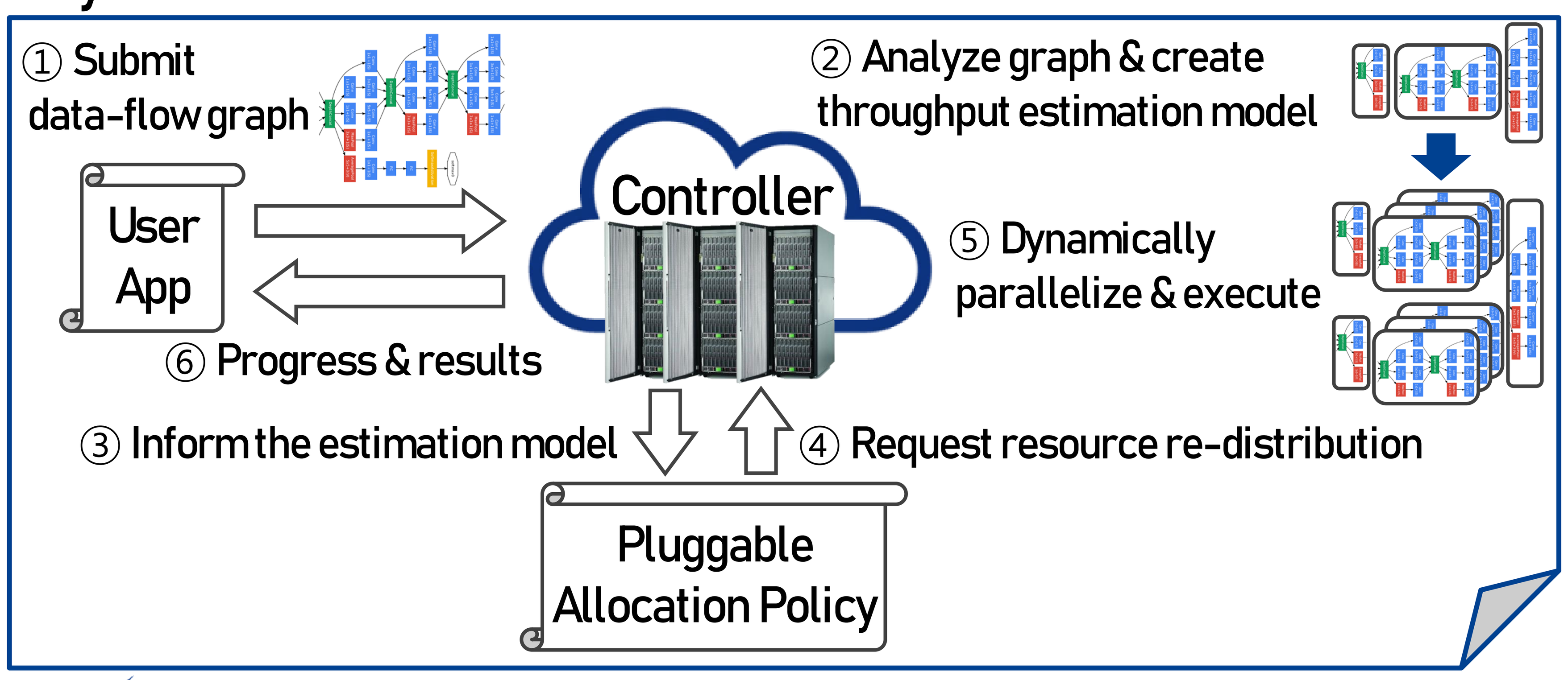


Topology-aware Resource Allocation
- E.g., 2 GPUs in a node vs. 3 GPUs across 2 nodes: which one performs better?
- Important to optimize overall throughput
- Requires inspection on both model & HW

Heterogeneity Consideration
- New lineup of GPU comes out almost once every year
- Load balancing btw different GPU types
- CPU type or # of CPUs may also vary

System-side Auto-parallelization

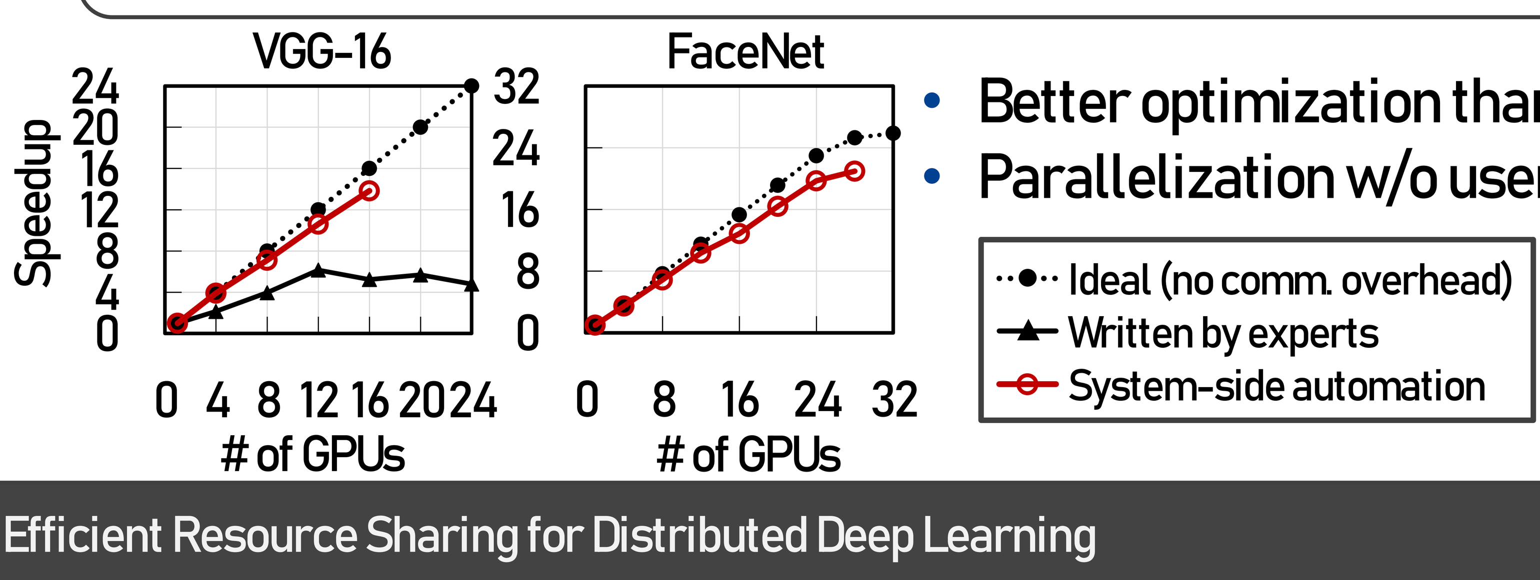
System Overview



✓ Specialized for Dynamic & Heterogeneous Resources

Dynamic data-flow graph management
- Change resources with low overhead by reusing the graph

Dynamic load balancing across heterogeneous devices
- Balance batch size & params to update depending on HW spec



- Better optimization than experts
- Parallelization w/o user efforts

Topology-aware Throughput Estimation

- The system should continuously find a better allocation
  - Simple trial-and-error is too wasteful
- Throughput estimation largely reduces trial-and-errors

✓ Accurate Throughput Estimation Using a Single GPU

Step 1: Offline System Performance Inspection
- Link bandwidth & overhead, parameter update performance

⬇

Step 2: Online Graph Inspection & Create Estimation Model
- Estimates timeline of operations w/ arbitrary resources given

⬇

Step 3: Online Adjustment of Estimation Model
- Update the model using the actual end-to-end throughput

- Preliminary results
  - Throughput estimation (1k samples/sec)