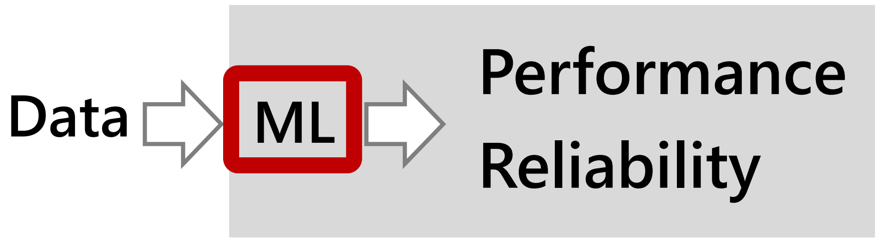
Large-Scale Shared GPU Clusters for DL Training Workloads

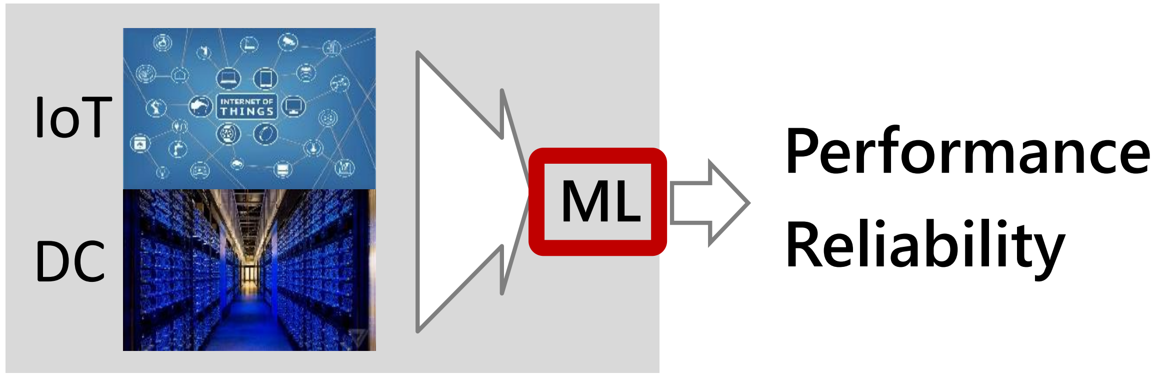
Myeongjae Jeon

UNIST CSE

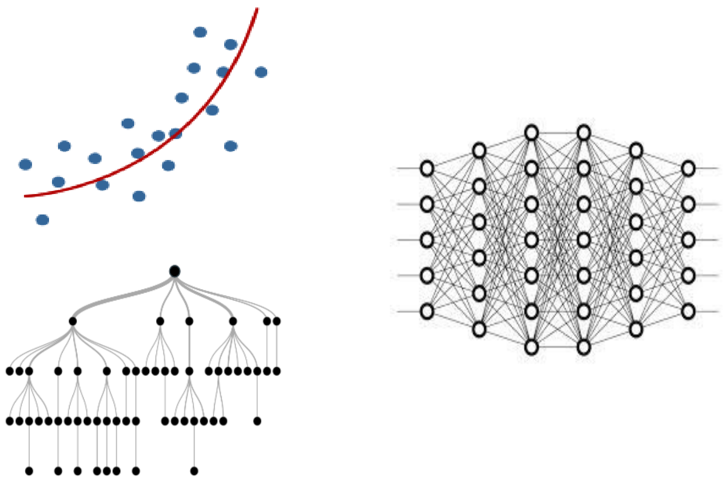
ML for Systems



Large-scale Stream Processing

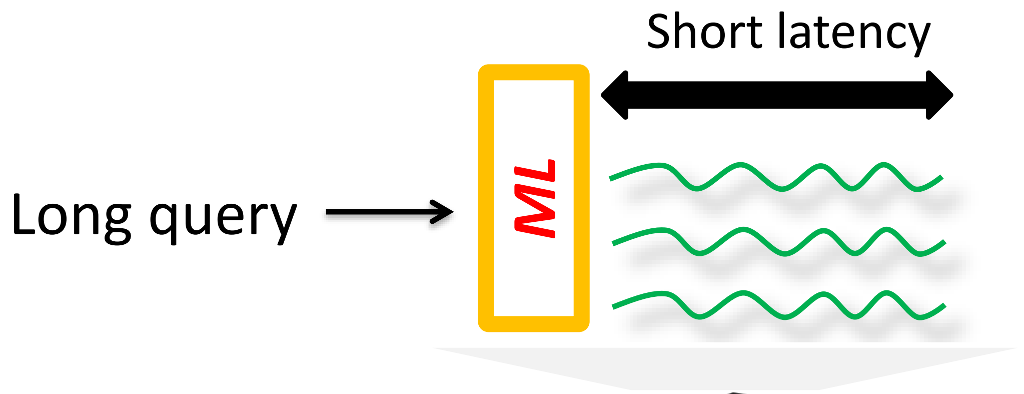


Systems for ML



Resource scheduling
Failure handing
Data/model sharing
Multi-tenancy
....

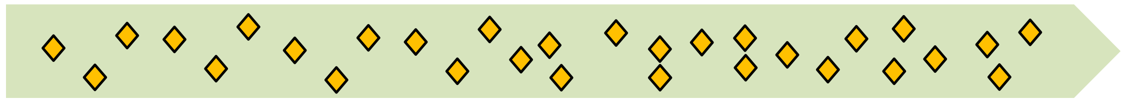
ML for Systems



[EuroSys'13, SIGIR'14, ASPLOS'16]

Large-scale Stream Processing

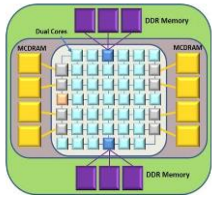
Compression [ATC'18], Approximation [arxiv]



Data management

Resource management

Manycore [ATC'17]



Fast memory [ASPLOS'19]

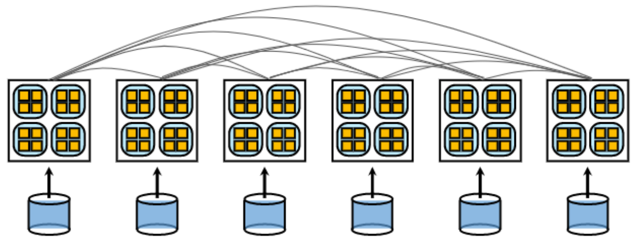
Systems for ML

Focus today

GPU cluster manager

[NSDI'19, ATC'19]

Efficient distributed training



[arxiv]

Deep Learning at Enterprise

Deep learning (DL) is popular

– Speech, Image, Ads, NLP, Web Search …

– ***10.5×*** increase of DL training jobs in Microsoft

DL training jobs require GPU

– ***5×*** increase of GPU cluster scale in Microsoft[1]

[1]. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. USENIC ATC 2019

Demands for Systems Supports

Resource scheduling and mgnt

– Training on few-to-many GPUs

– High-speed network

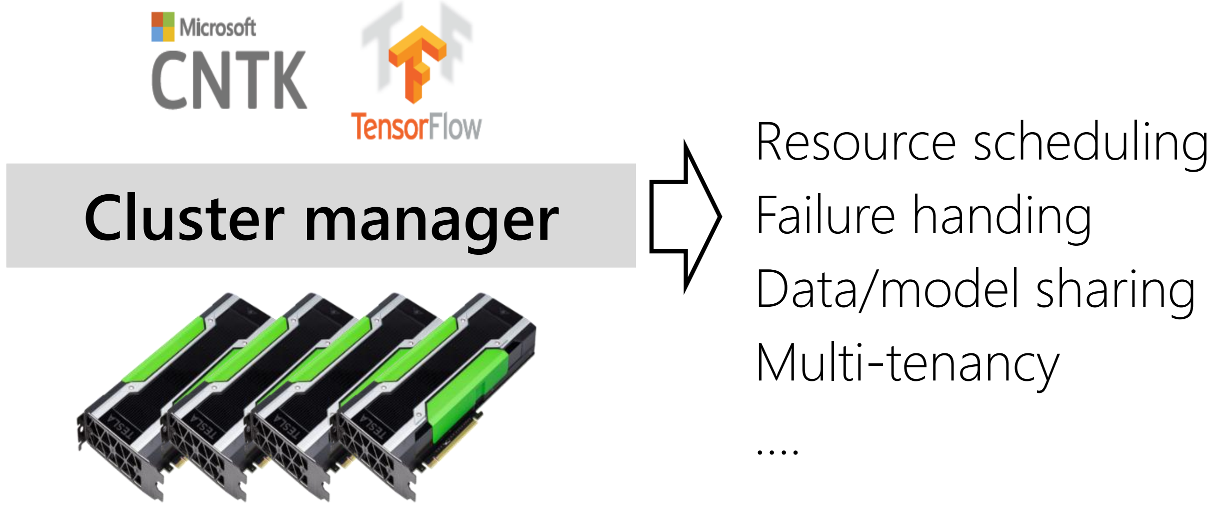
Failure handling

– Days to weeks of job run time

Storage and data handling

– Identical training data

– Reusing checkpointed models



How to efficiently manage a GPU cluster for DL training jobs?

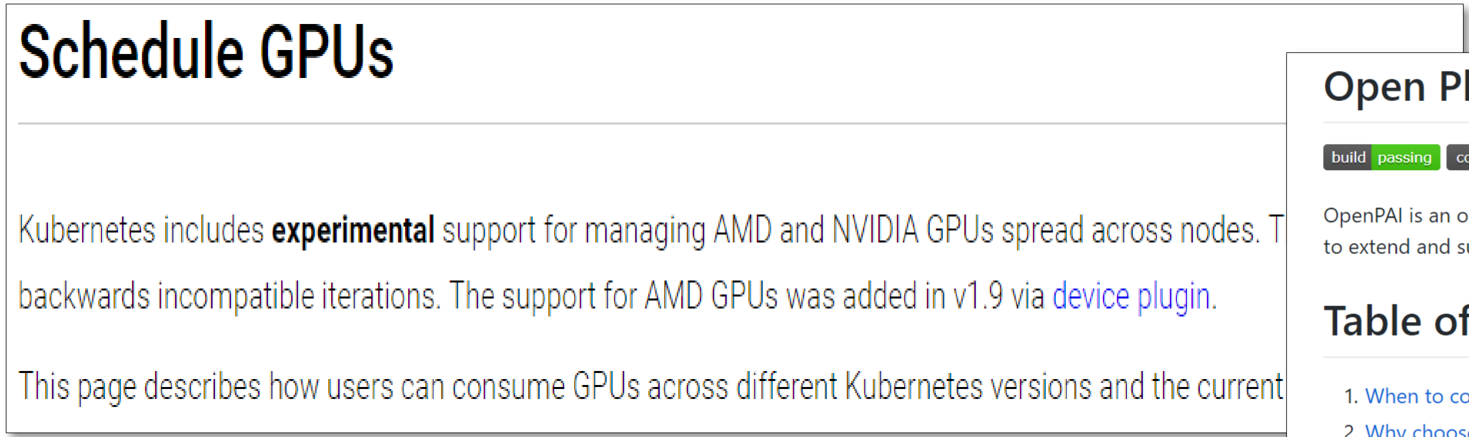
State-of-the-art DL Cluster Managers

	Gandiva [OSDI'18]	Philly [ATC'19]	Optimus [EuroSys'18]	Tiresias [NSDI'19]
Objective	Consolidation	Consolidation	Average JCT	Average JCT
Job Property	Any	Any	Converging	Any
Scheduling Algorithm	Time-sharing	Locality-based	SRTF	Gittins Index
Input	N/A	Arrival time	Remaining time	Attained service

Most used Microsoft trace[1], will be open for public soon! ☺

[1]. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. USENIC ATC 2019

Widespread Support by Open Source

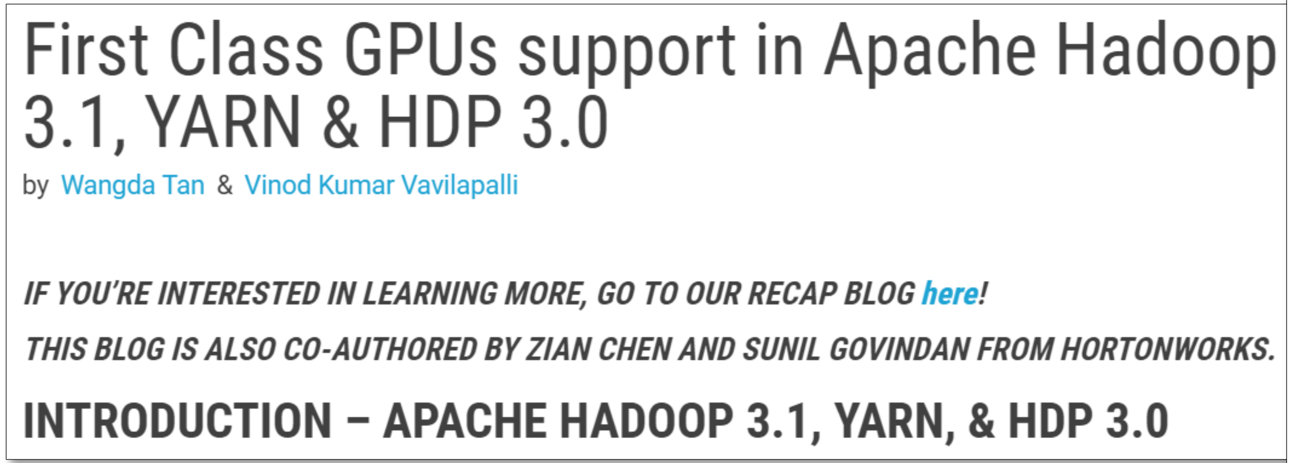


Schedule GPUs

Kubernetes includes **experimental** support for managing AMD and NVIDIA GPUs spread across nodes. T

backwards incompatible iterations. The support for AMD GPUs was added in v1.9 via device plugin.

This page describes how users can consume GPUs across different Kubernetes versions and the current



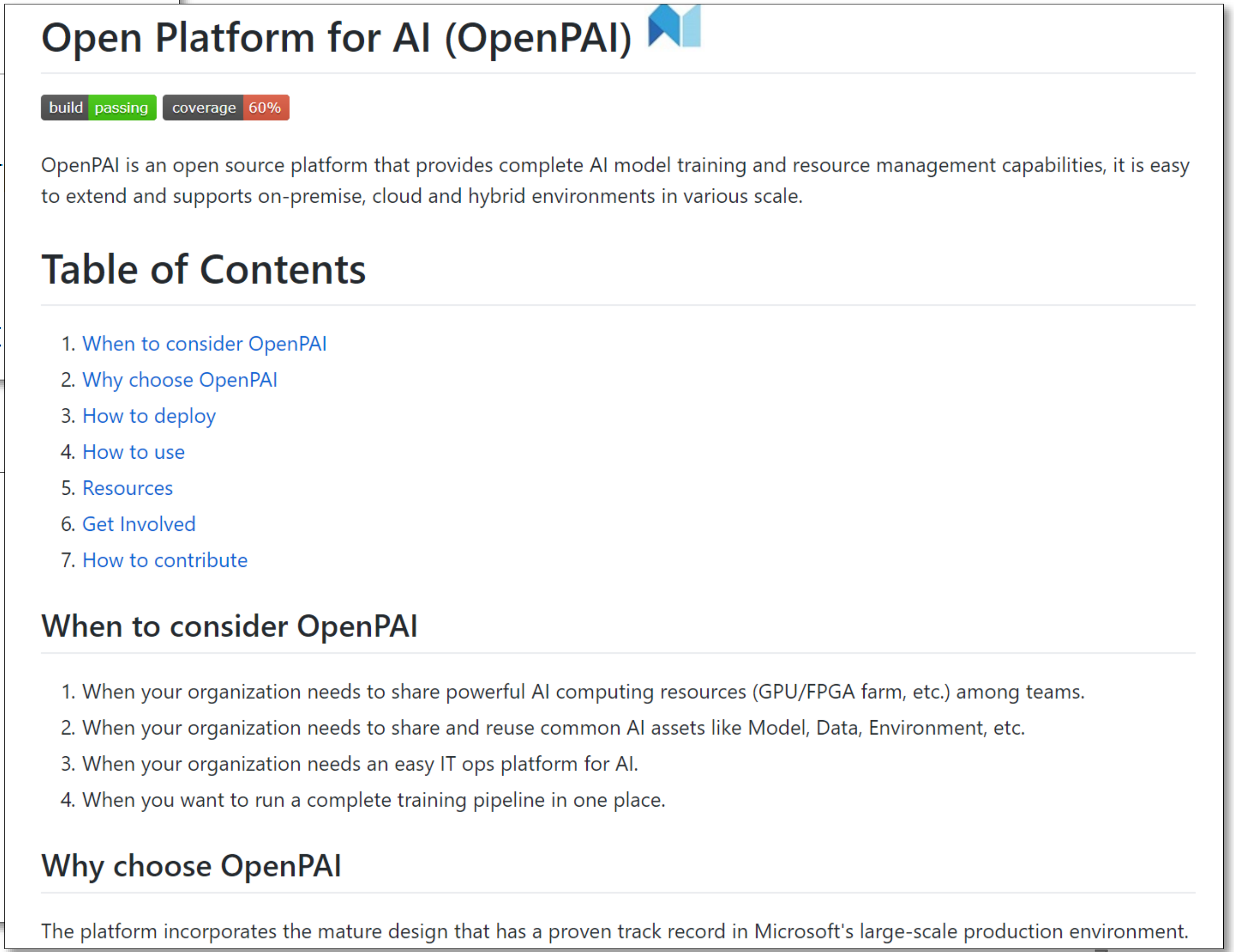
First Class GPUs support in Apache Hadoop 3.1, YARN & HDP 3.0

by Wangda Tan & Vinod Kumar Vavilapalli

IF YOU'RE INTERESTED IN LEARNING MORE, GO TO OUR RECAP BLOG here!

THIS BLOG IS ALSO CO-AUTHORED BY ZIAN CHEN AND SUNIL GOVINDAN FROM HORTONWORKS.

INTRODUCTION – APACHE HADOOP 3.1, YARN, & HDP 3.0



Open Platform for AI (OpenPAI)

build passing coverage 60%

OpenPAI is an open source platform that provides complete AI model training and resource management capabilities, it is easy to extend and supports on-premise, cloud and hybrid environments in various scale.

Table of Contents

When to consider OpenPAI

1. When your organization needs to share powerful AI computing resources (GPU/FPGA farm, etc.) among teams.
2. When your organization needs to share and reuse common AI assets like Model, Data, Environment, etc.
3. When your organization needs an easy IT ops platform for AI.
4. When you want to run a complete training pipeline in one place.

Why choose OpenPAI

The platform incorporates the mature design that has a proven track record in Microsoft's large-scale production environment.

Outline

Overall architecture of GPU cluster

Comm cost of distributed training

Strategy in Philly and Tiresias

Raising a few issues for the future

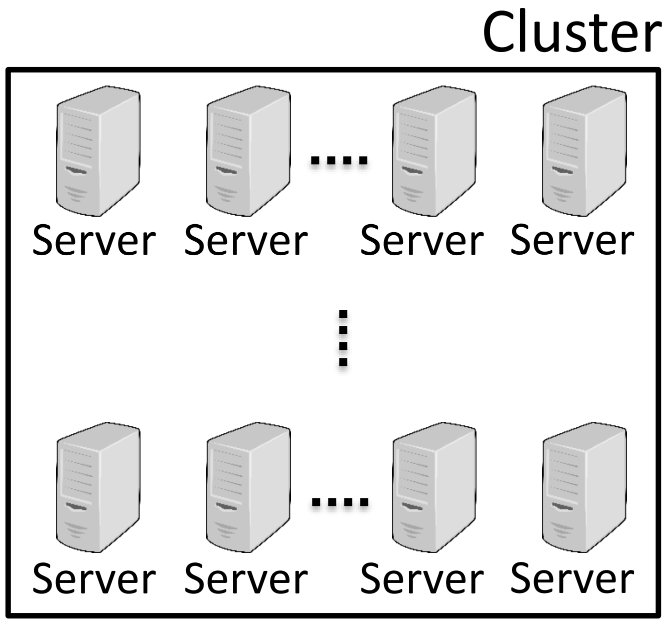
– Comm efficiency

– Failure handling

– More accessibility on HW

Shared GPU Cluster Architecture

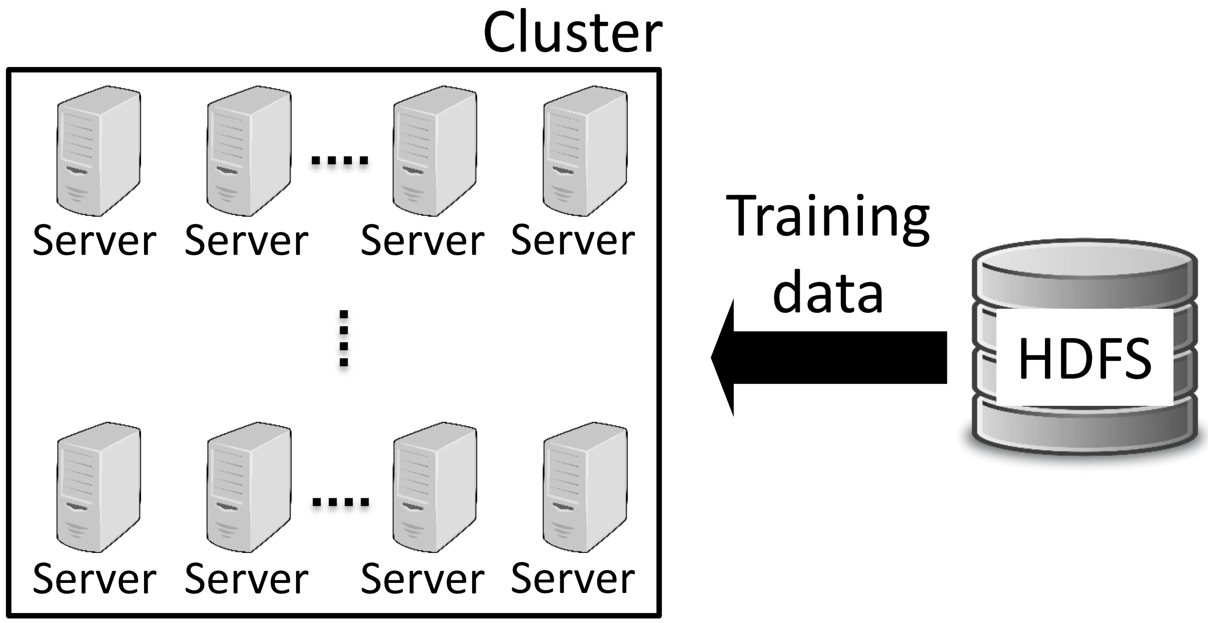
GPU cluster 100s of servers and thousands of GPUs



Shared GPU Cluster Architecture

GPU cluster 100s of servers and thousands of GPUs

HDFS Distributed "shared" storage for training data (and models)

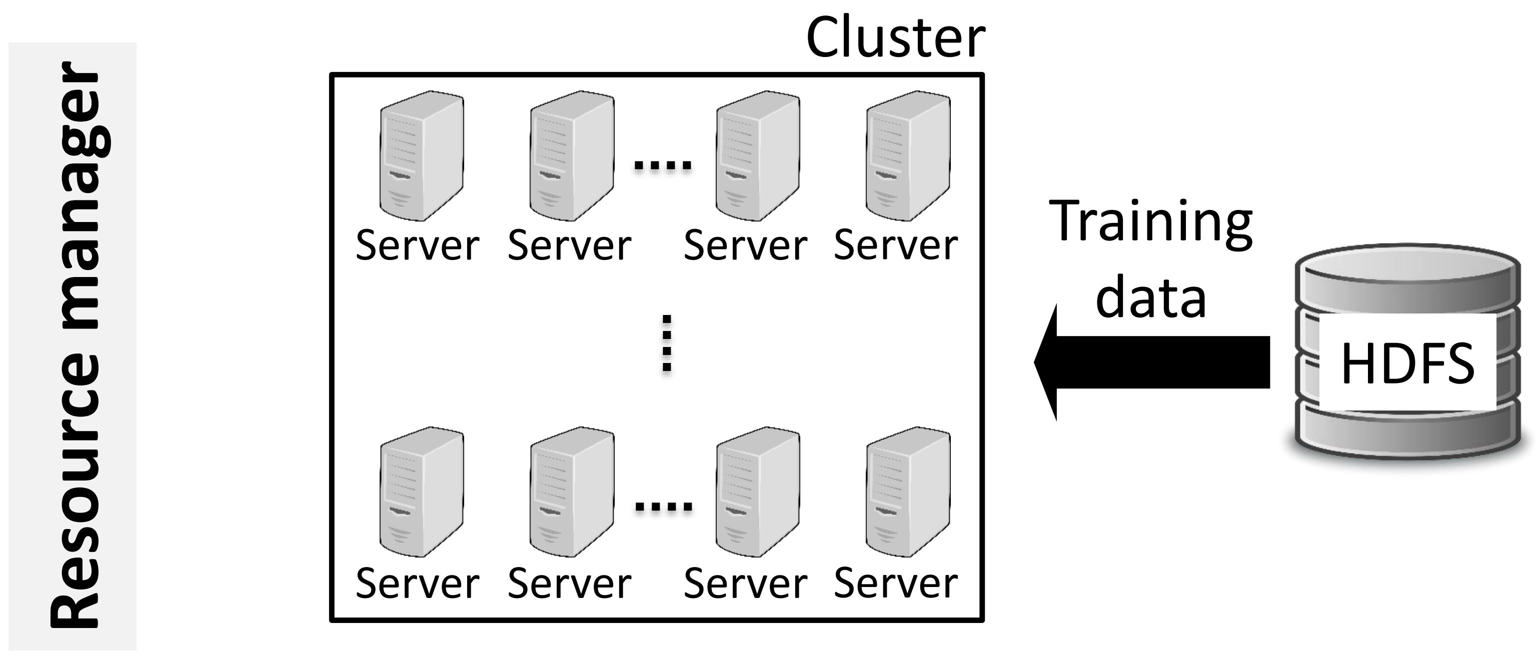


Shared GPU Cluster Architecture

GPU cluster 100s of servers and thousands of GPUs

HDFS Distributed "shared" storage for training data (and models)

RM Managing system resources for jobs submitted online

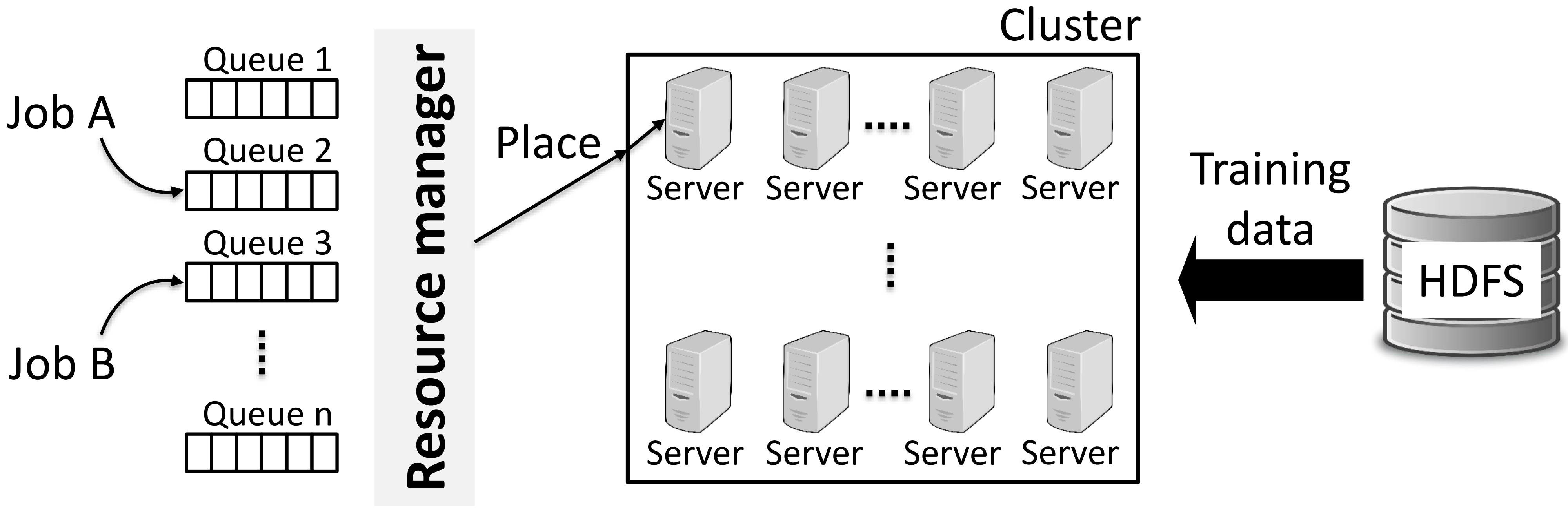


Shared GPU Cluster Architecture

Queues Resource allocation (i.e., number of GPUs) for each group

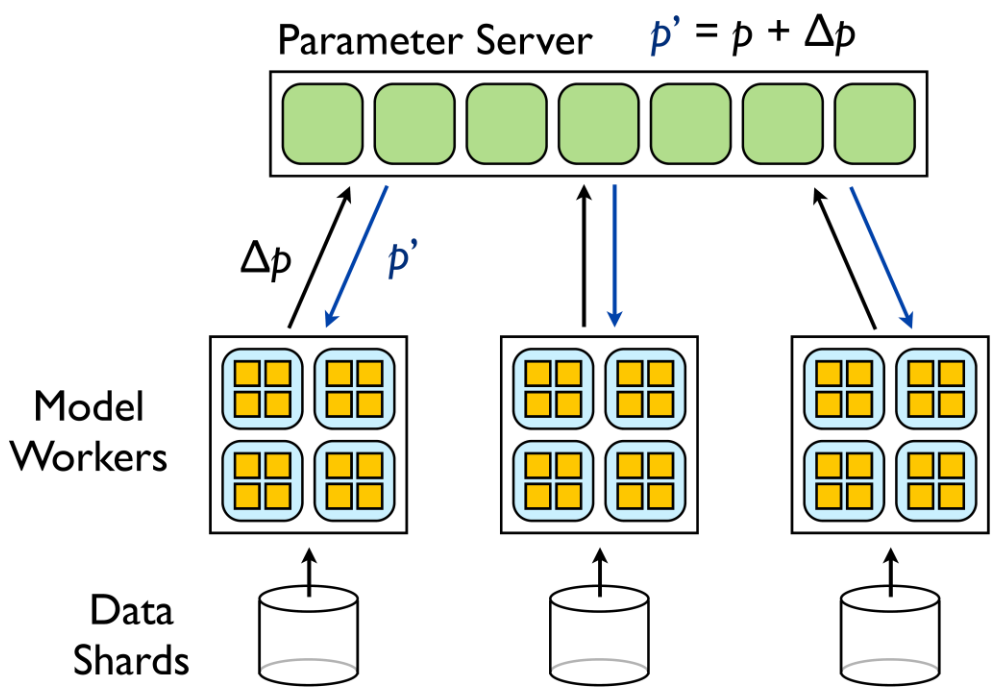
Fairness among groups (e.g., by Apache YARN's Fair Scheduler)

Oversubscription: Allocate idle GPUs to a queue with additional demand



Distributed Training in DL Clusters

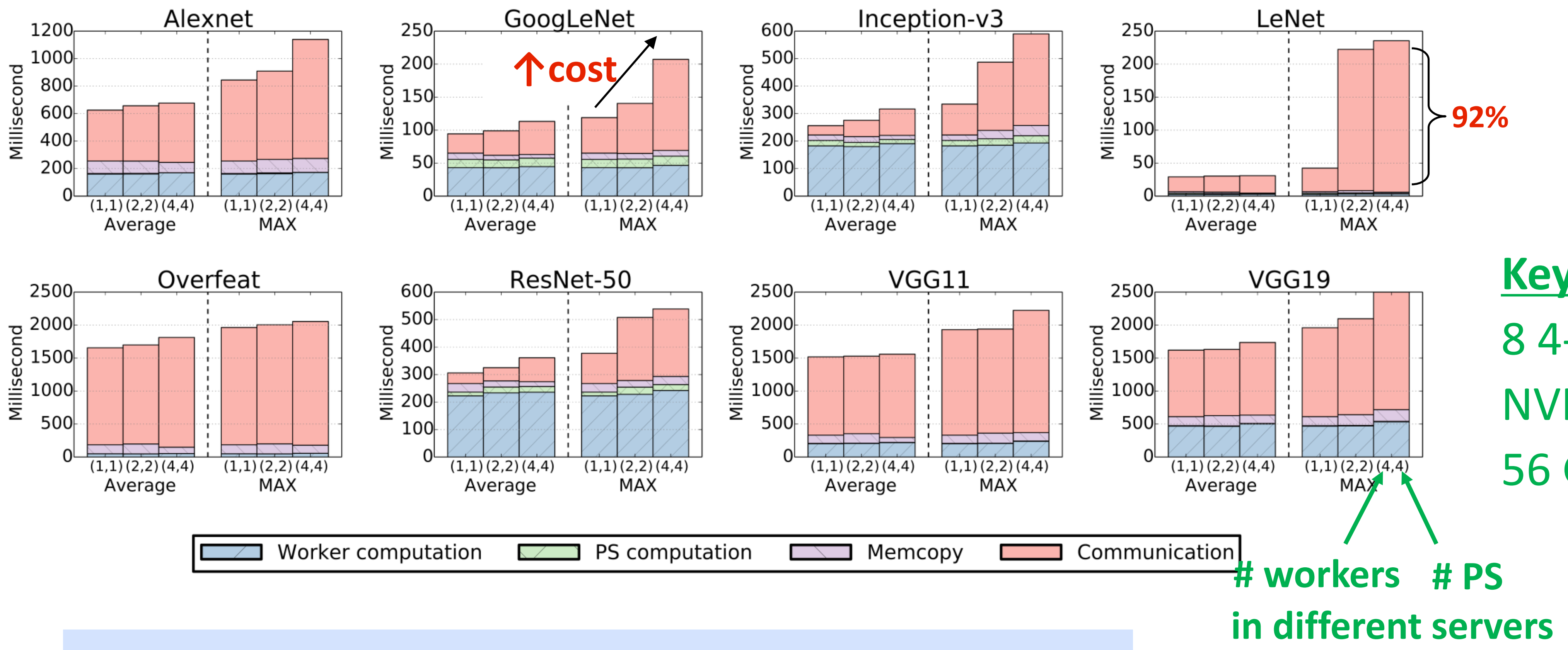
Data parallelism has been most widely used



Train same model on distinct data

(+) Easier to parallelize
(–) High comm cost due to frequent "model" sync

Network Cost in Data Parallel Training



Key HW config

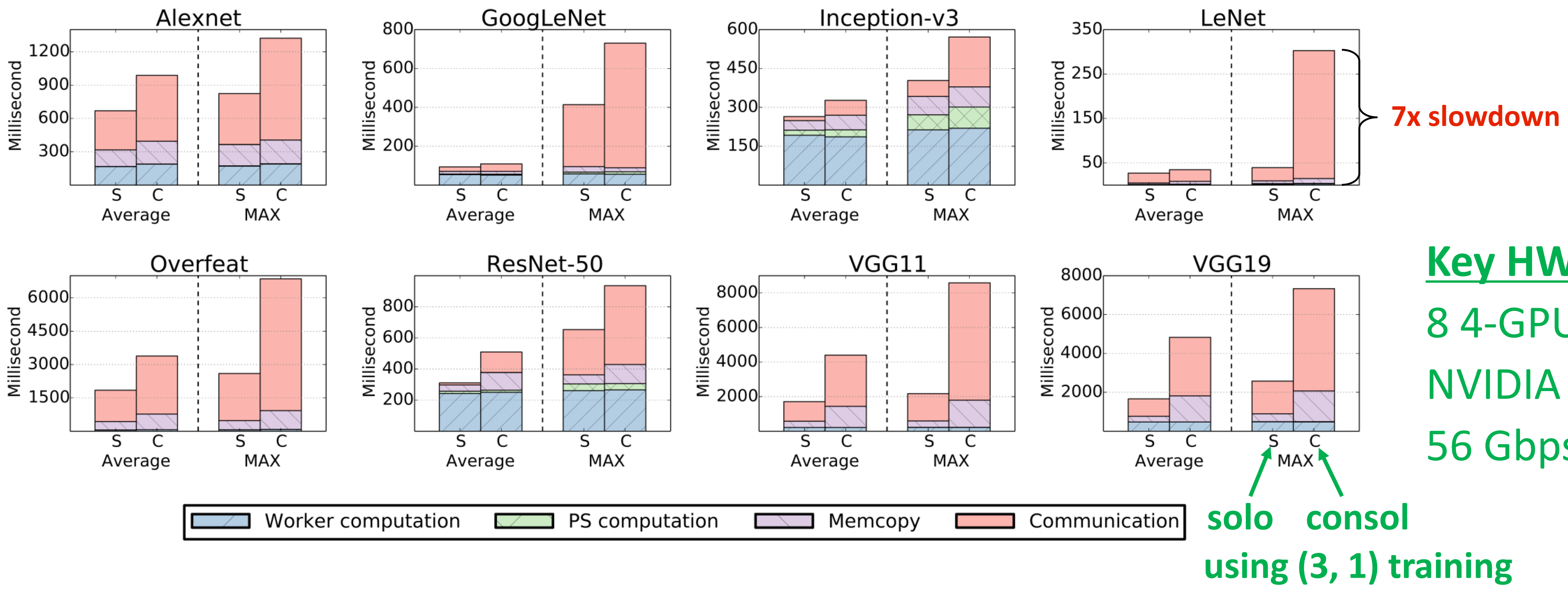
8 4-GPU servers

NVIDIA Titan Xp

56 Gbps InfiniBand

Communication taking 58% on average!

Network Cost in Data Parallel Training



Key HW config

8 4-GPU servers

NVIDIA Titan Xp

56 Gbps InfiniBand

Jobs interfere each other while sharing network!

Outline

Overall architecture of GPU cluster

Comm cost of distributed training

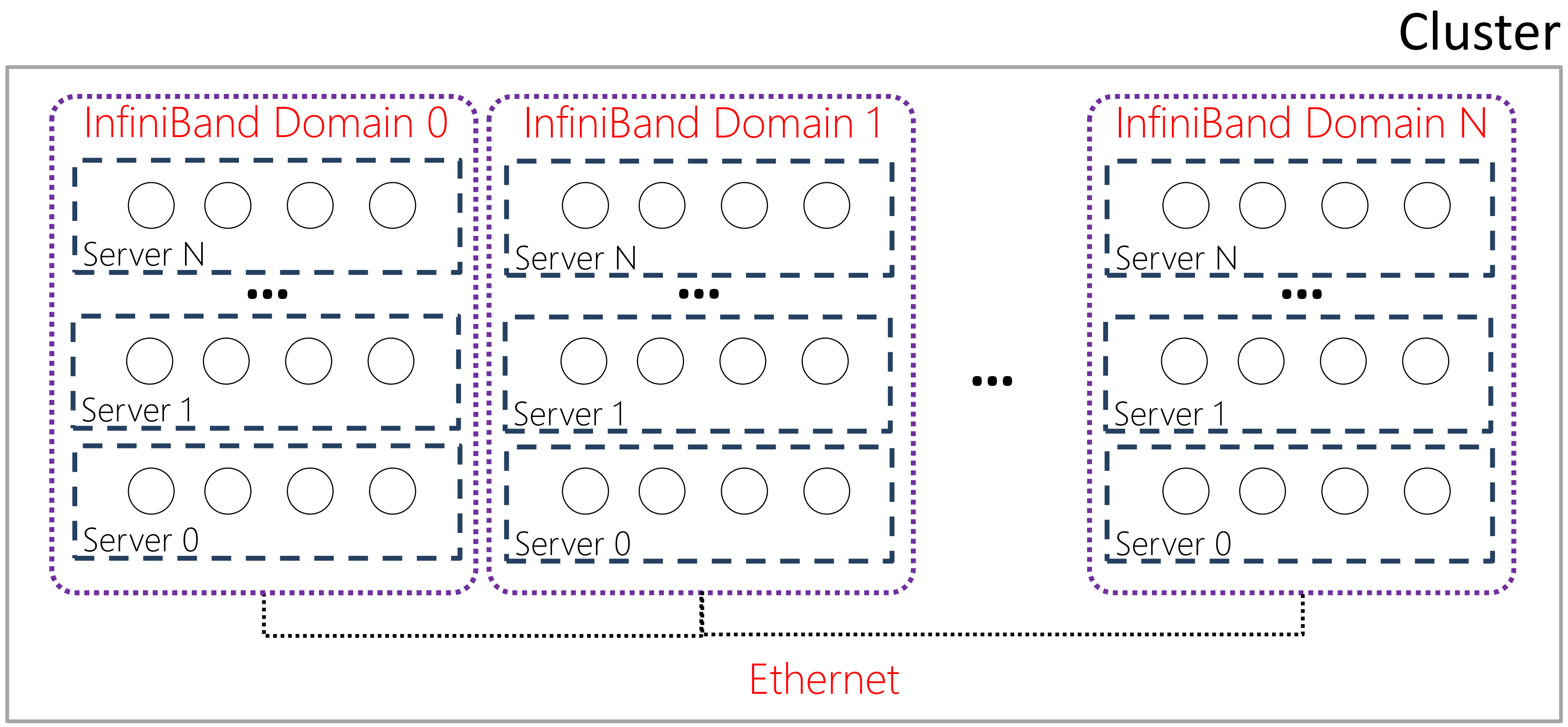
Strategy in Philly and Tiresias

Raising a few issues for the future

- Comm efficiency
- Failure handling
- More accessibility

Deeper into Comm Heterogeneity

1. **Intra-server GPU comm is only for 4 or 8 GPUs**
2. **High-speed network is rack-localized (optional)**



Job Placement in Philly

Job placement must be locality-aware!

Each server has 4 or 8 GPUs

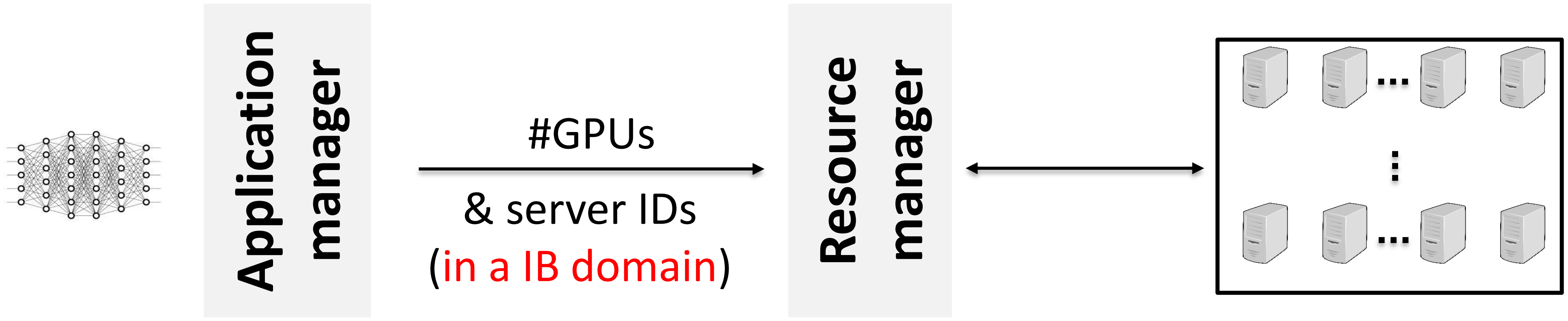
→ ***Pack a job's GPUs onto the smallest number of servers possible***

High-speed network channel is rack-localized

→ ***Pack a job's GPUs within a single InfiniBand domain***

Resource Negotiation

Each job has AM to negotiate resources from RM



CPU/memory assigned proportional to # of GPUs

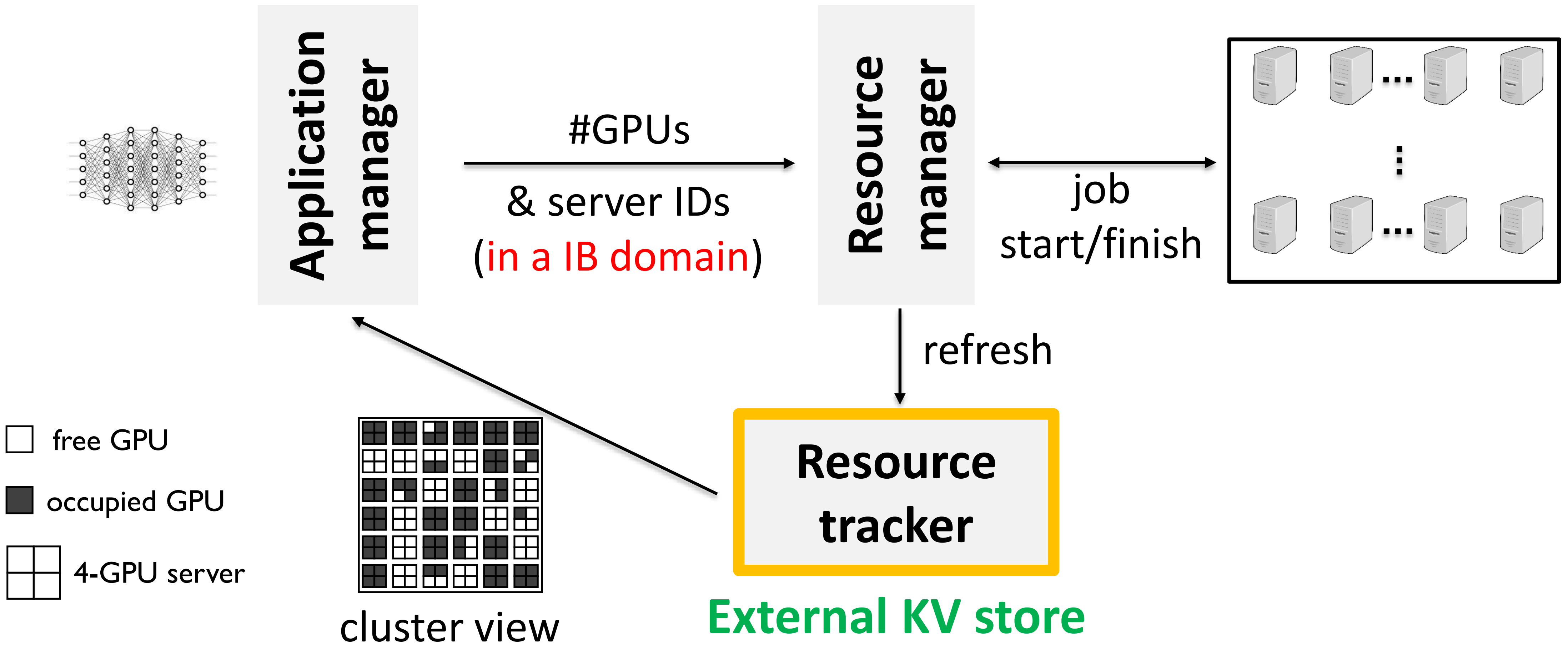
– Simple to ML practioners; Easier resource packing

Specific servers meeting the desired locality in a IB domain

– Leverage near-data affinity feature in existing Bigdata RMs

Decentralized Approach

Let each AM have the global view of the cluster



Scheduling Workflow

Step 1: Make a request to RM

- *Calculate # of servers & GPUs per server*
  - *"Highest locality" at the beginning (i.e., using the fewest servers)*
- *Pick a rack that has such servers most available*
- *Pick servers to be placed*
  - *Servers for better packing*

Step 2: Not all GPUs ready until timeout?

- *Release any acquired GPUs and take a back-off*

} **Avoid starvation**

Step 3: Retry the request

- *Gradually "relax locality" constraints*
  - $\uparrow$ *schedulibility at comm cost*

} **Trade-off training efficiency for higher workload consolidation**

Philly’s Limitations

① Policy not targeting on training performance

– ML practioners care about job completion time (JCT)

② Locality constraints limit job schedulability

– Not all distributed training benefit from GPU locality

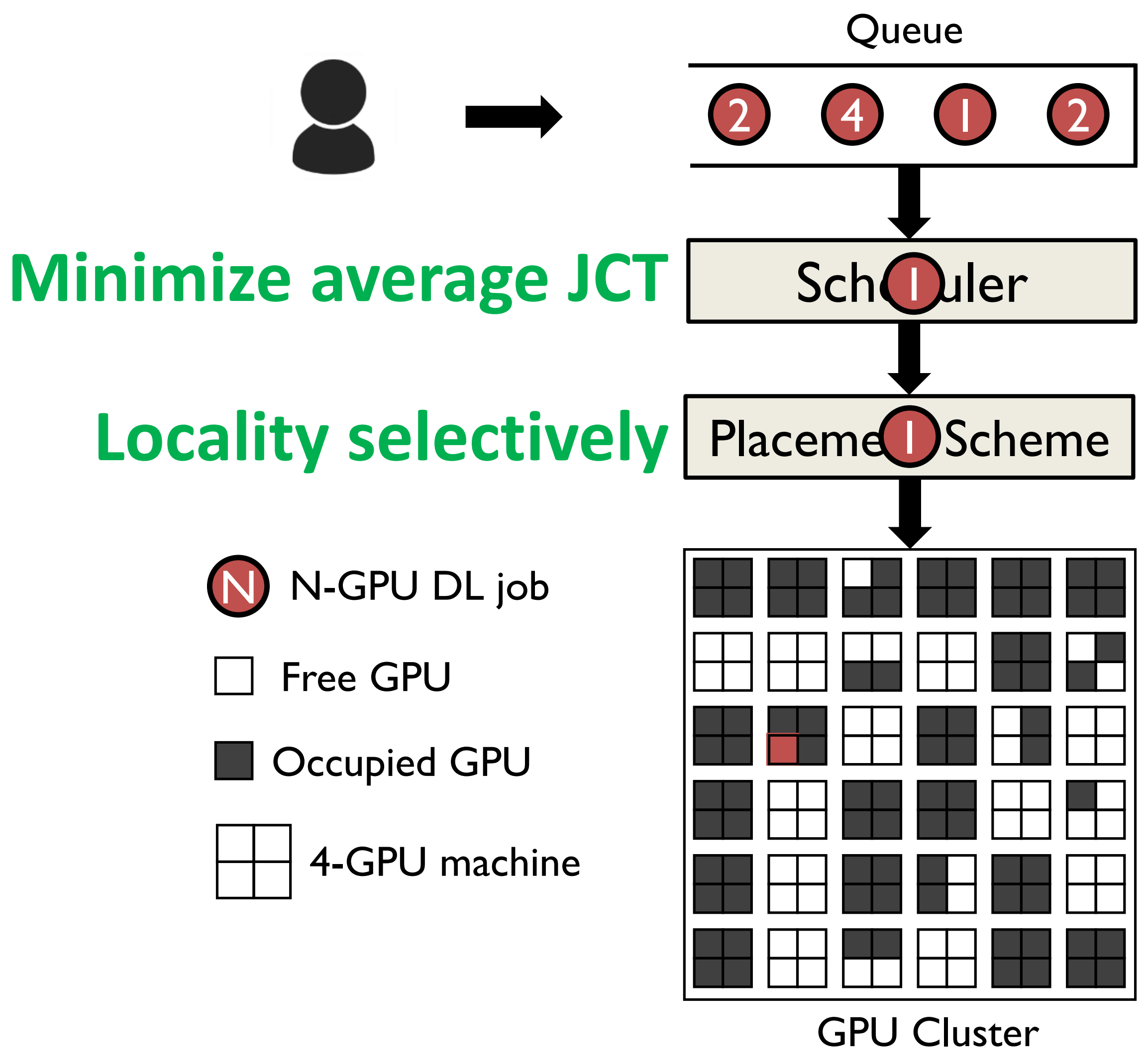
Tiresias Design Objectives

① **Policy not targeting on training performance**

– ML practioners care about job completion time (JCT)

② **Locality constraints limit job schedulability**

– Not all distributed training benefit from GPU locality



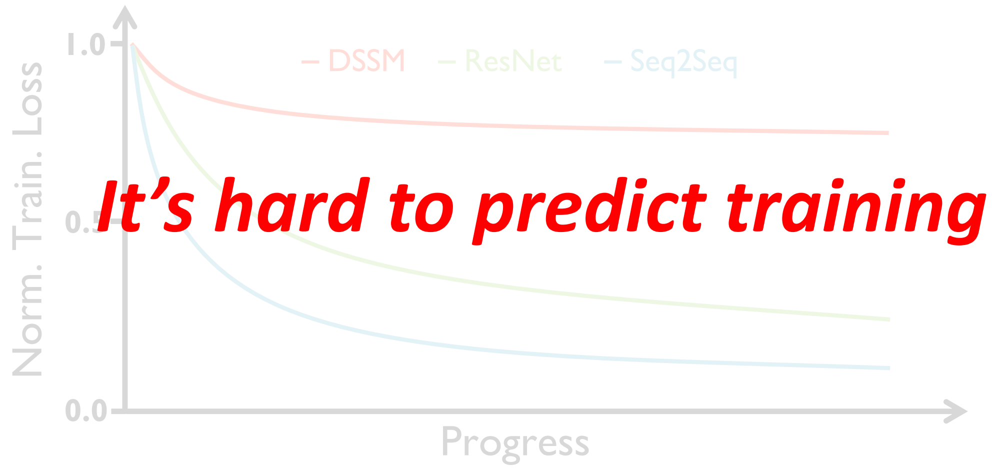
Easy to Predict Job Training Time?

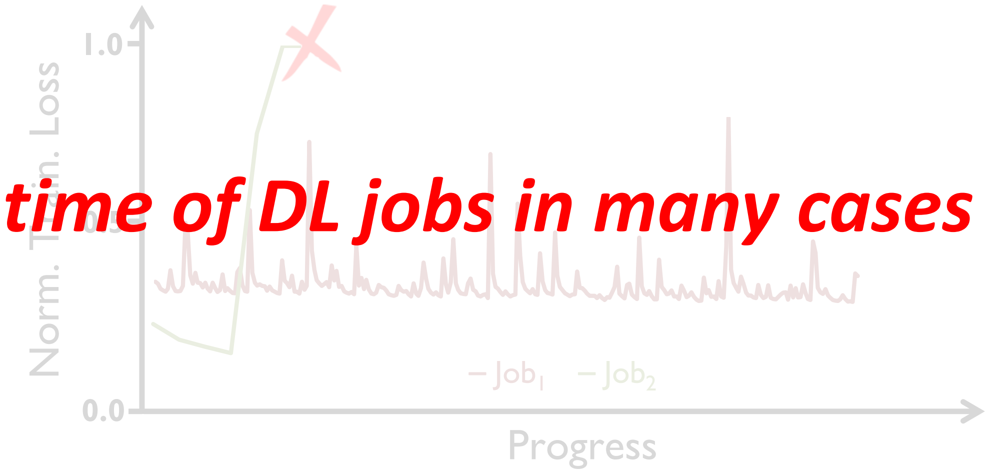
Job training time is useful when minimizing JCT

– Unknown before execution

Can predict job training time

– Use the smooth loss curve of DL training jobs (*Optimus* [1])



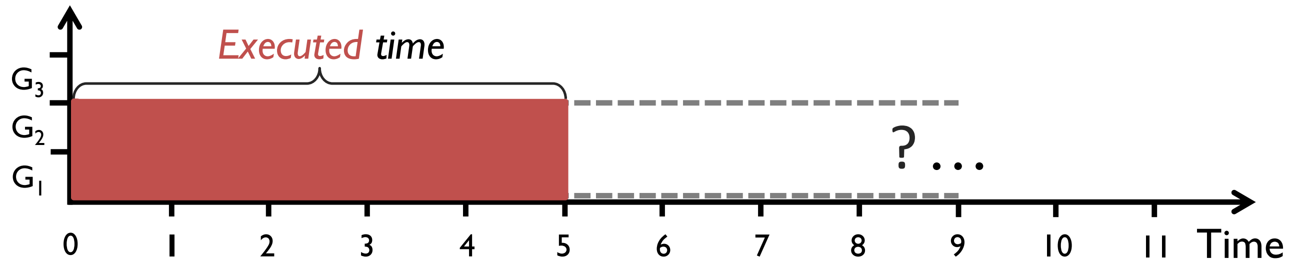


It's hard to predict training time of DL jobs in many cases

[1]. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. EuroSys 2018

Available Job Information

1. Spatial: number of GPUs
2. Temporal: *executed* time (age)



Age-based Scheduler

Gittins Index

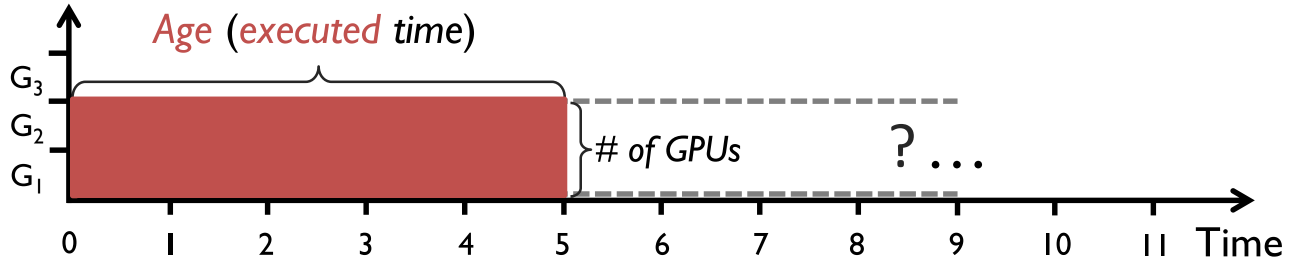
– Need the *distribution of job execution time*

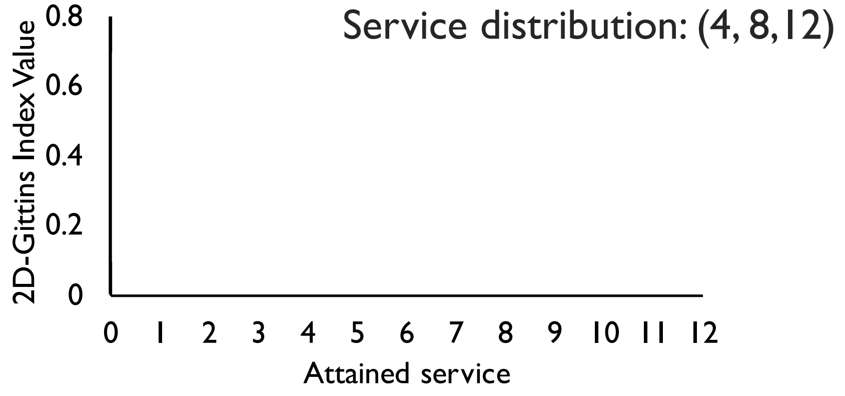
– **Probability** to complete in the near future based on current age

2D-Gittins Index policy

– Age calculated by attained service (# of GPUs × executed time)

– **Prioritize** a job that has the largest Gittins Index

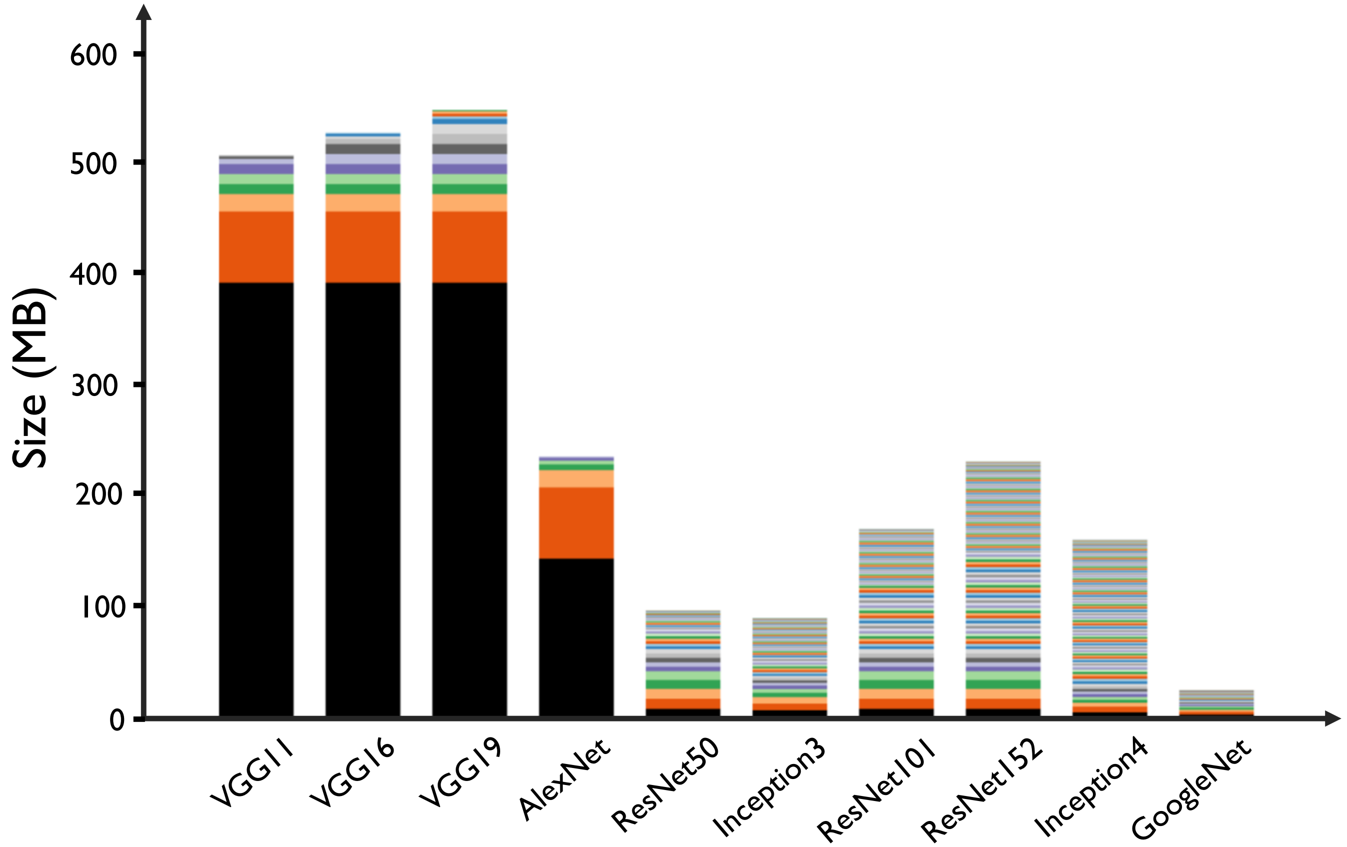




Model Profile-based Placement

Tensor size in DL models

Large tensors cause network imbalance and contention



Consolidated placement is needed when the model is highly skewed in its tensor size

Outline

Overall architecture of GPU cluster

Comm cost of distributed training

Strategy in Philly and Tiresias

Raising a few issues for the future

– Comm efficiency

– Failure handling

– More accessibility

Mitigating Comm Cost

Data-driven approaches

– Model compression

– Model quantization

– Model sync batching

→ ***often comes at the cost of accuracy loss***

Execution-driven approaches

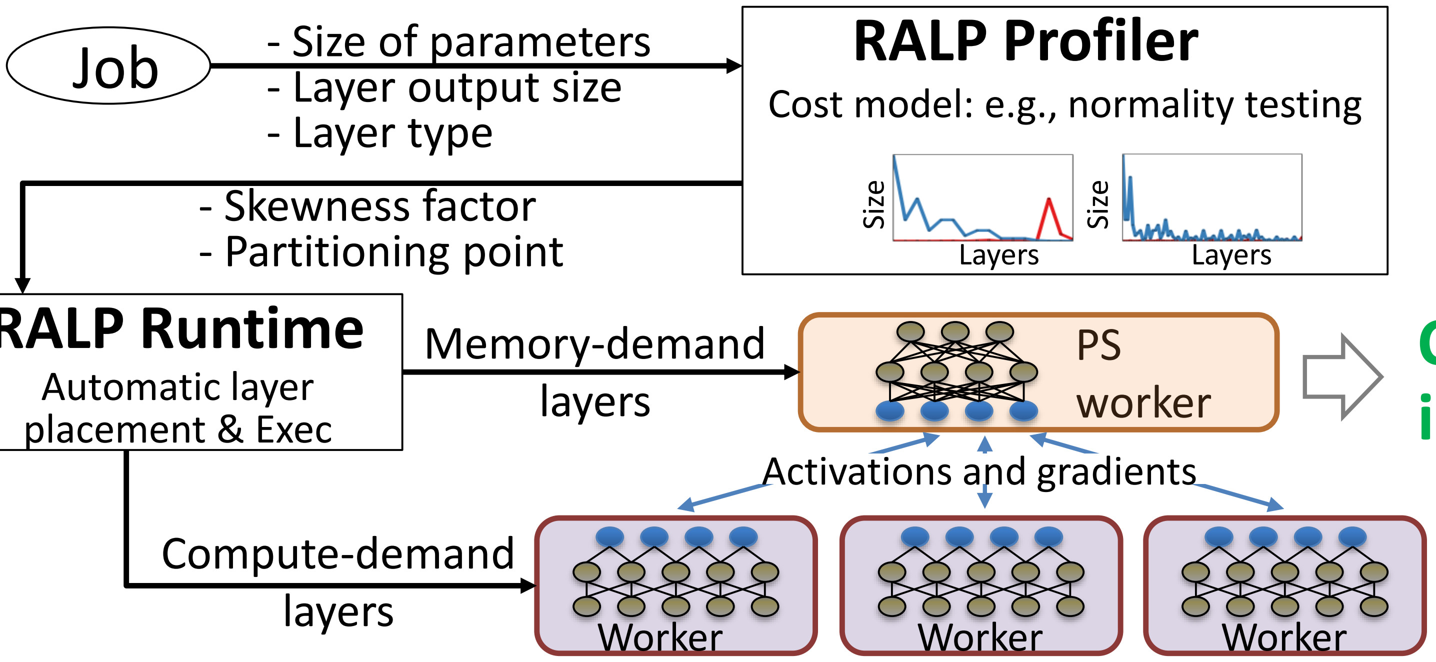
– Comm-aware model parallelism

→ ***difficult to automate***

Mitigating Comm Cost

Automatic resource-aware layer placement using RALP

– Distributed training at low comm cost



Colocated with PS for intra-server comm

Most GPUs Allocated ≠ Effectively Utilized

“Effective” cluster utilization can be hurt by

1. **Relaxed GPU locality**

   GPU utilization of 16-GPU jobs: 2 nodes (43.66%) v.s. 8 nodes (28.56%)

   → ***Prioritize locality more? Colocate training jobs?***

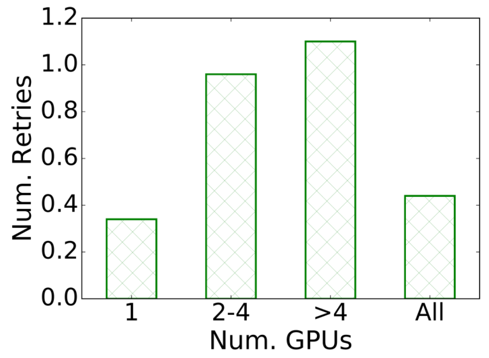
2. **Effectiveness of the last epochs**

   75% jobs reach 0.1% of the best accuracy using only 40% of epochs

   → ***Trade-off the accuracy for large resource savings?***

3. **Training job failures**

   Frequent for distributed training



Mitigating Job Failures

Individual job: User errors in code and configuration

– Many independent components communicating each other

– Not strongly types languages

→ ***Pre-run the first iter on a single GPU (from a pool of cheaper GPUs)***

Across jobs: Input format error or corrupted inputs

– Difficult to prevent while generating data

→ ***Need input data blacklisting***

SW & Trace, then HW is Accessible?

Having open platforms is more than necessary!

1. Own training infrastructure setup
   - The number of GPUs in distributed training keeps increasing
     - 32 (2016) → 128 (2017)
   - 128 GPUs = $1M

2. Borrowing resources from cloud
   - 128 GPUs for 12 hours = $5K

Summary

Shared GPU cluster is coming popular for DL training

– Need to design cluster managers for diverse circumstance

Network cost during distributed training is detrimental

– Worse with increasing use of many GPUs

– Cluster managers can mitigate the cost

Many improvements are desired for better future

Writing a Systems Paper

Factors evaluated in top systems conferences

- Novelty
- Performance improvement
- Important/Practical problem (motivating the work)
- How practical the solution is: Usability/Applicability/Generality
- …

Strategy

- Try to address as many as possible, but….
- Novelty is nowadays difficult to meet, and people know….

Motivate Your Problem Comprehensively

Measurement is the King (& reviewers want free lunch!)

- Comprehensive measurements while motivating the problem
  - If work is about single-job optimization, motivate the problem considering
    - How would it be running on a single machine?
    - How would it be running on a small/large cluster?
    - How would it be if network is shared with others
    - …
- The same strategy applies to evaluation section
  - Various scales
  - Diverse parameters
  - Interesting sensitivity tests

Pay Attention on Usability/Applicability/Generality

I think this is the most important factor these days

Take RALP as an example

- When I first met RALP, it was just partitioning CONV layers and FC layers in CNN
- What I proposed
  - RALP profiler for usability (automatic & selective partitioning) and applicability (model-agnostic solution)
  - RALP on other AI engines for applicability
  - RALP on many GPUs and multi-point partitioning for generality
  - What RALP can bring out in practice
    - Mitigating network interference, resource saving v.s. higher perf, better cluster scheduler

Pay Attention on Usability/Applicability/Generality

NOTE! Reviewers often challenge you for these aspects

Support with data as much as possible!

- If you put some efforts (while not perfect), the reviewers will appreciate it
  - Many papers are on the borderline
  - In many cases, reviewers reject papers as they do not agree on (or do not see data on) how practical the solution "could be"
- I prefer to say even in texts (if I do not have time to get data) for one or two very critical issues

Do NOT Underestimate Rebuttal

My advisors used to address only a few points from reviews

– Never got positive feedback from rebuttal

Try to address all concerns from reviews

– Then you will sometimes get positive feedback, and..

– Survive from borderline

Consider prioritization and lookup efficiency in rebuttal

– First address common questions

– Then address individual questions for each reviewer

Thank You!

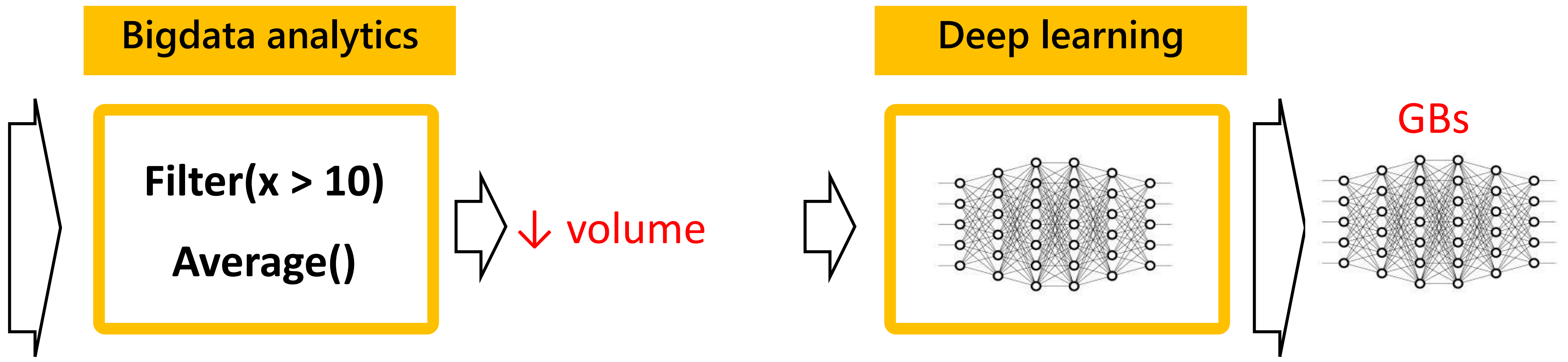
mjjeon@unist.ac.kr

Comm Cost in Data Parallelism

Data parallelism is most widely used in DL clusters

Periodic voluminous communication

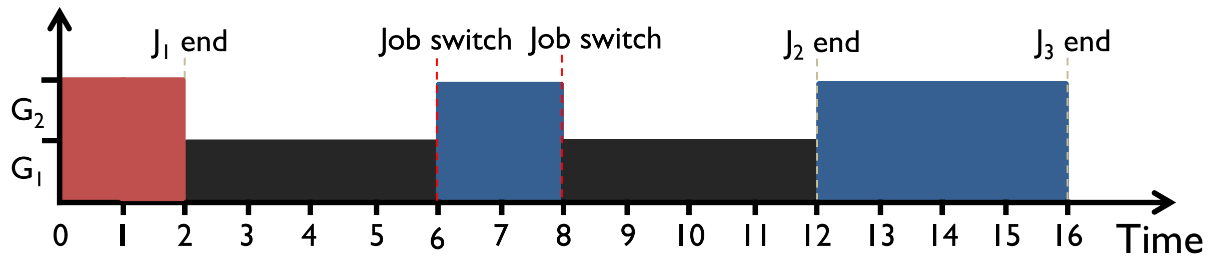
- Workers running on multiple GPUs synchronize training progress



2D-Gittins Index: Partial Information

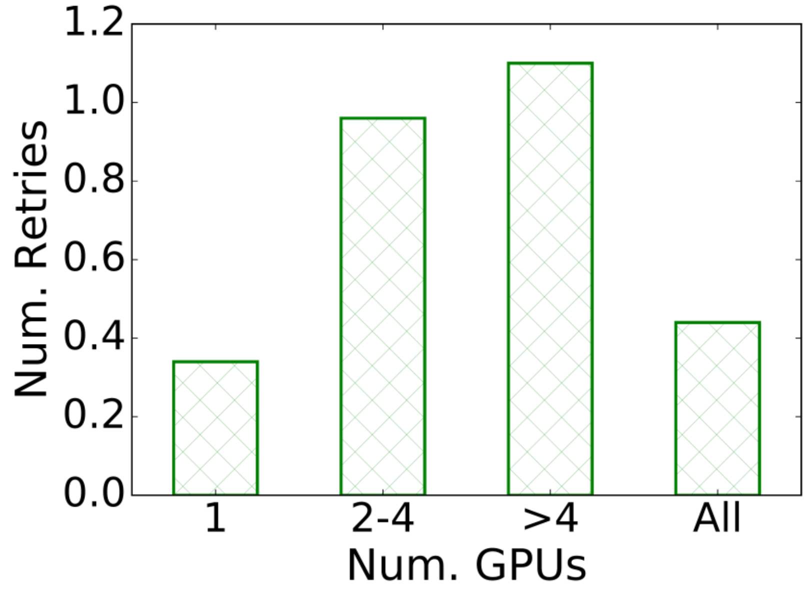
- Higher *probability to complete* (*Gittins Index*), higher priority

	# of GPUs	Distribution
J_1	2	2
J_2	1	(4, 8, 12)
J_3	2	6



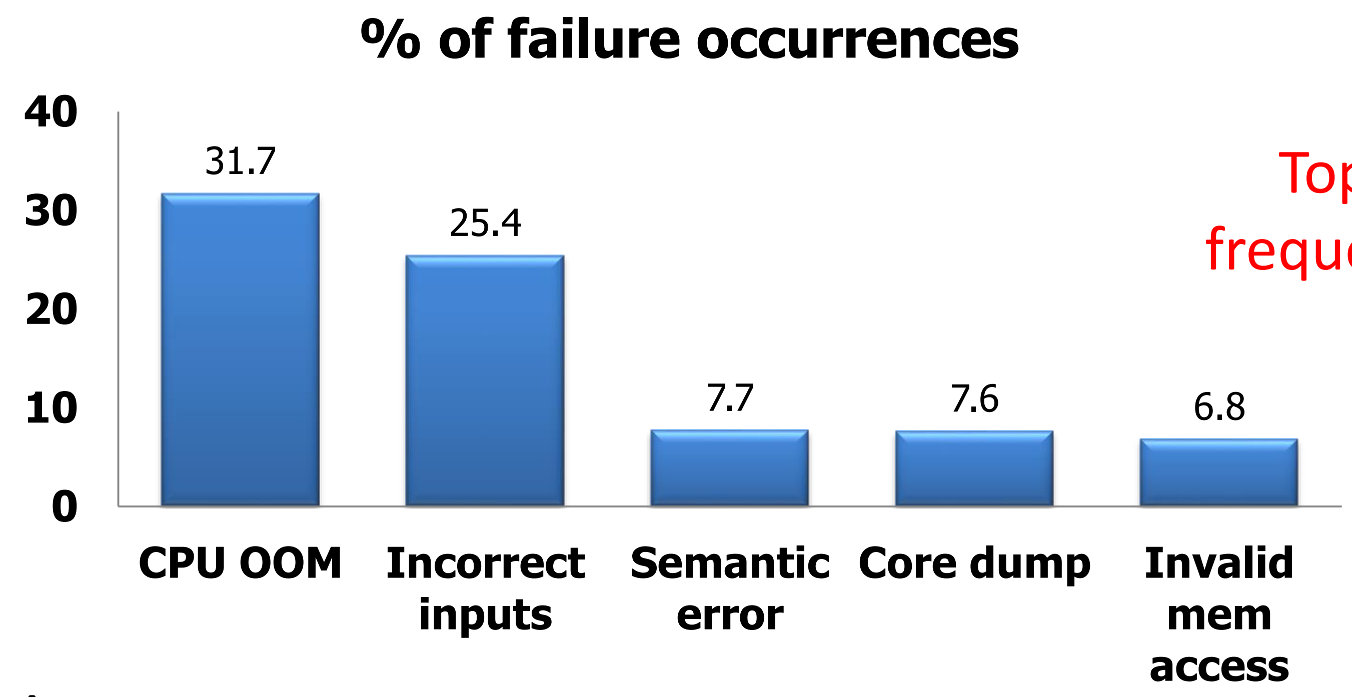
Job Failures

- A job is retried upon **failure**
- A job is **unsuccessful** if it *repeatedly* fails
  - Up to a pre-defined number of retries (e.g., 4 retries)



Observation 1:

Many failures by user/programming mistakes

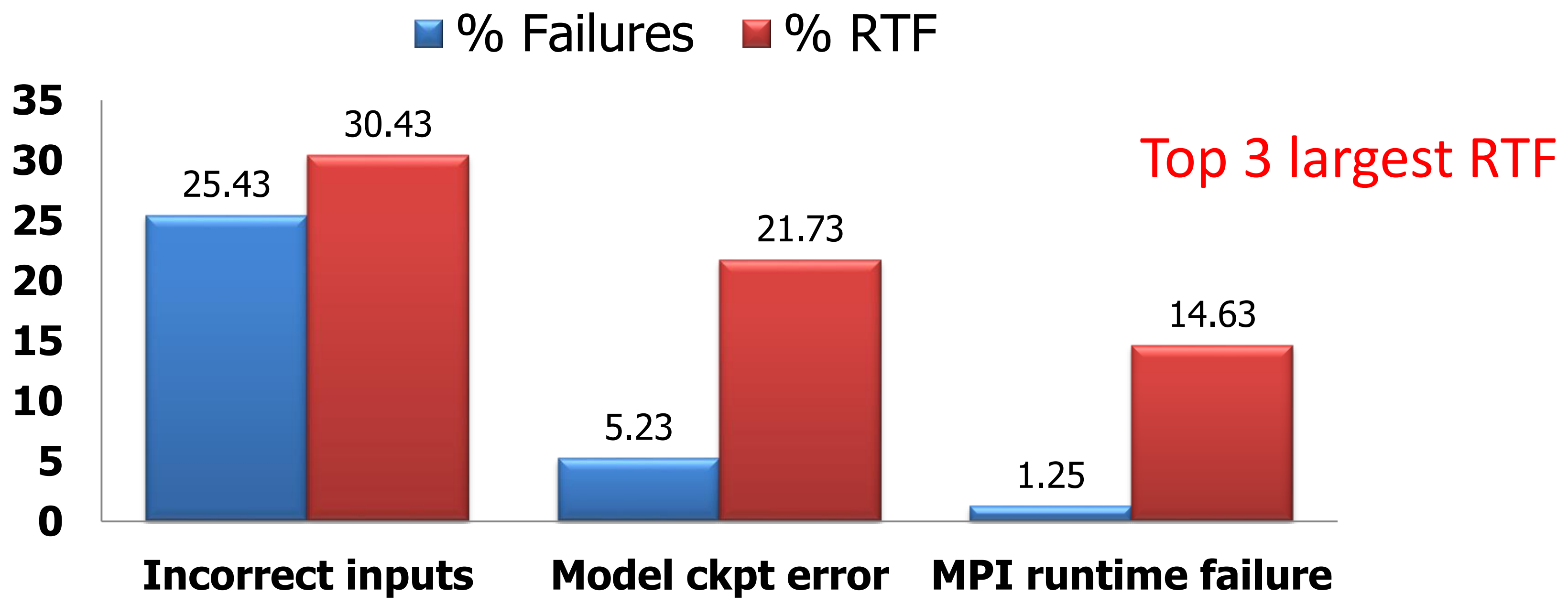


Top 5 most frequent failures

- Primary factor:
  - Many independent components
  - Not strongly typed languages

Observation 2:

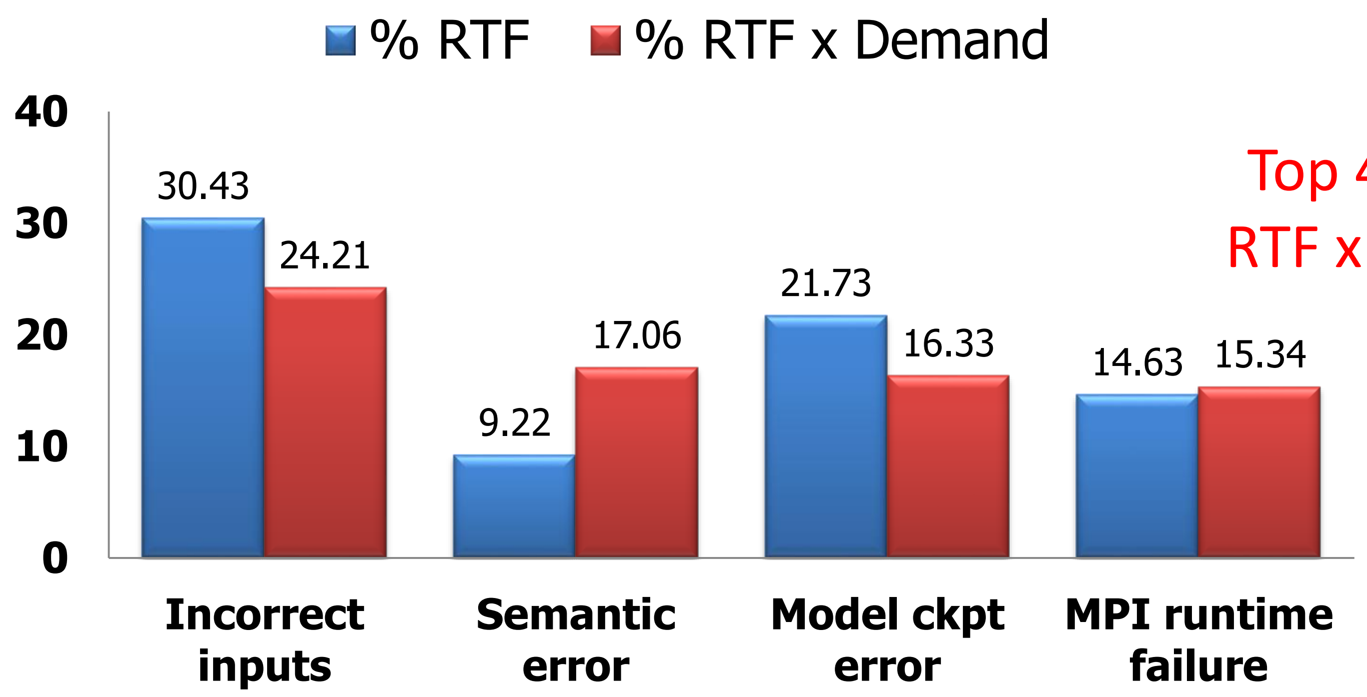
Long RTF by infrequent infrastructural failures



- Primary factor:
  - Nondeterministic error in program-to-storage and program-to-program communication

Observation 3:

Long RTF by semantic error for larger jobs



Top 4 largest RTF x Demand

- Primary factor:
  - Send/receive/access data in an inconsistent way during model sync