Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters

Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chuanxiong Guo

The University of Hong Kong Bytedance Inc.

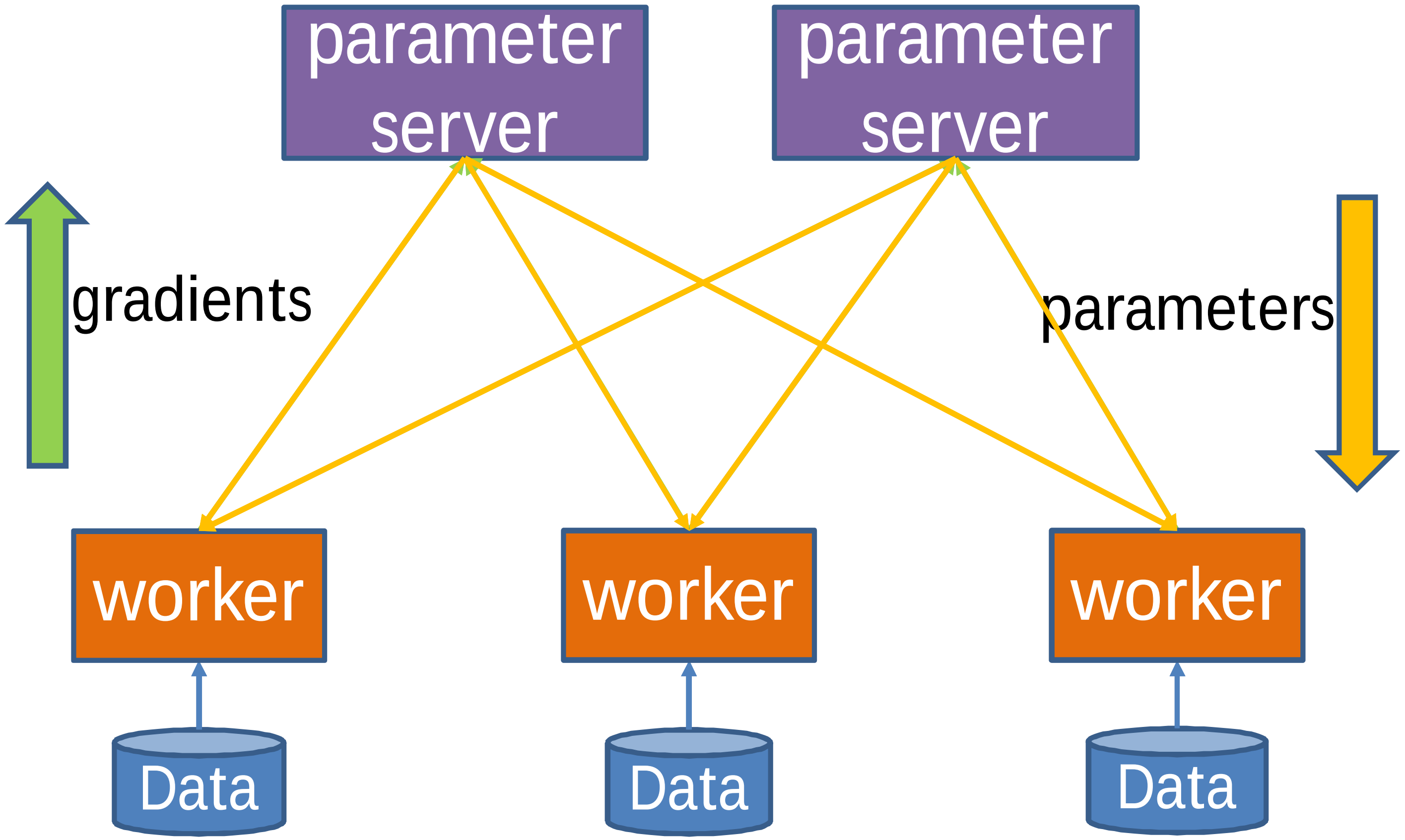
Deep Learning

- Increasing deep learning workloads in production clusters
  - Speech recognition
  - Object classification
  - Machine translation

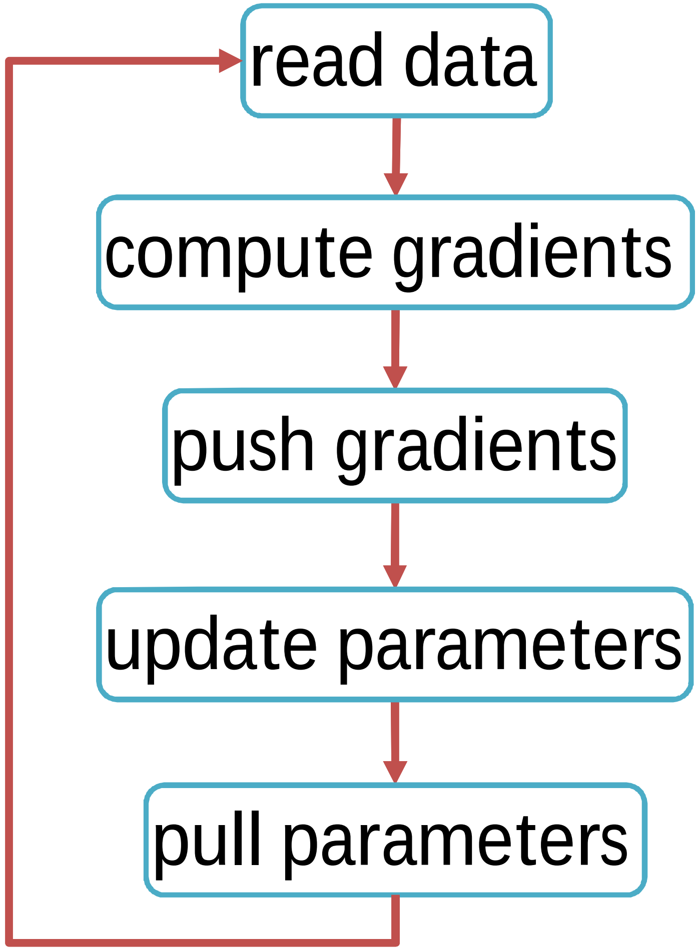
- Many machine learning frameworks
  - TensorFlow
  - MXNet
  - PaddlePaddle

Distributed Training

- Parameter server (PS) architecture



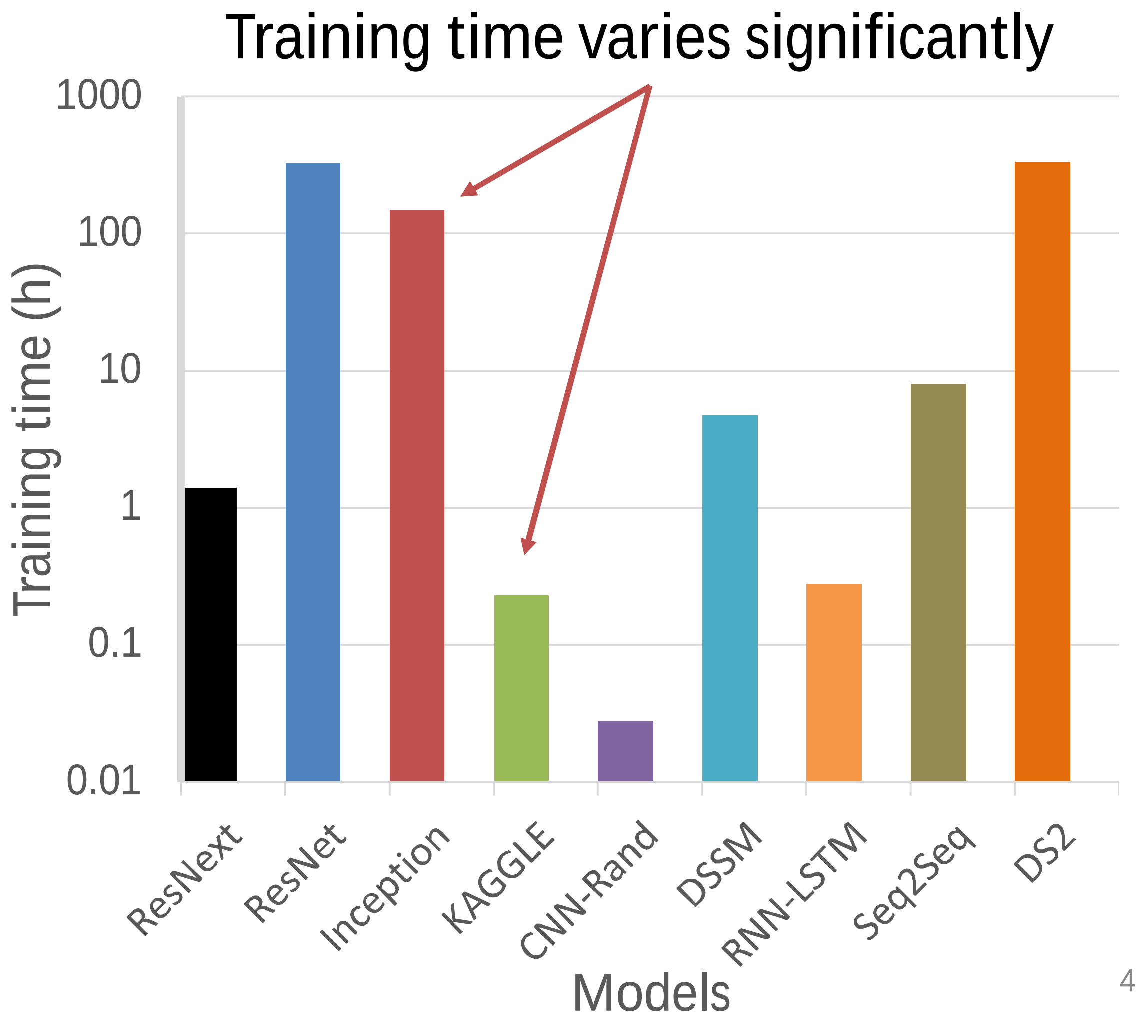
- Iterativeness



Cluster Scheduling — Current Practice

- Static allocation, i.e., fixed numbers of parameter servers and workers during training
  - § Varying resource availability

- Job size unawareness
  - § Long jobs may block short jobs



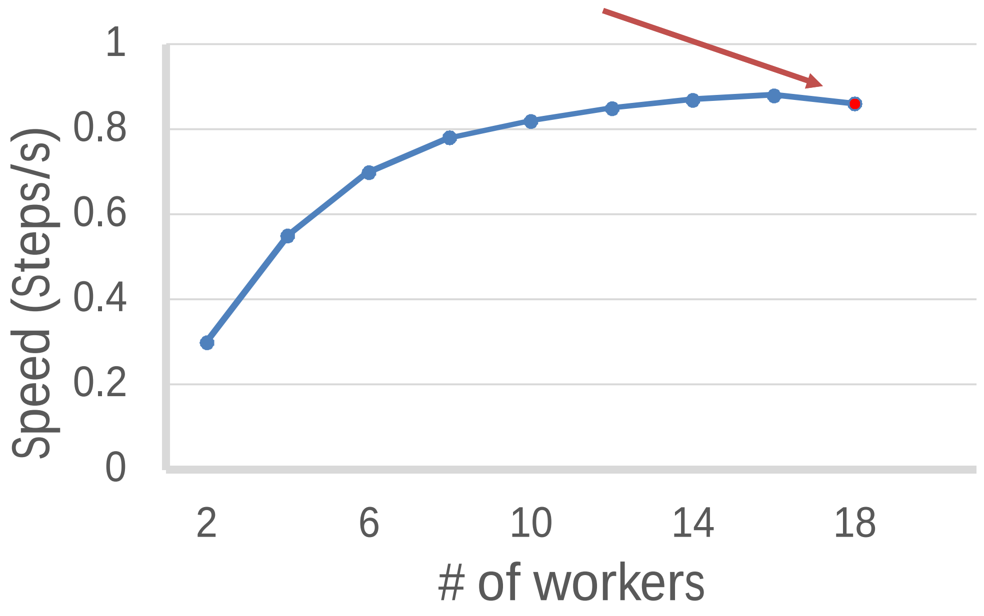
Cluster Scheduling — Current Practice

- Manually specify resource configuration
  - § Suboptimal

Maximal speed: 8 workers and 12 ps

20 ps and workers in total

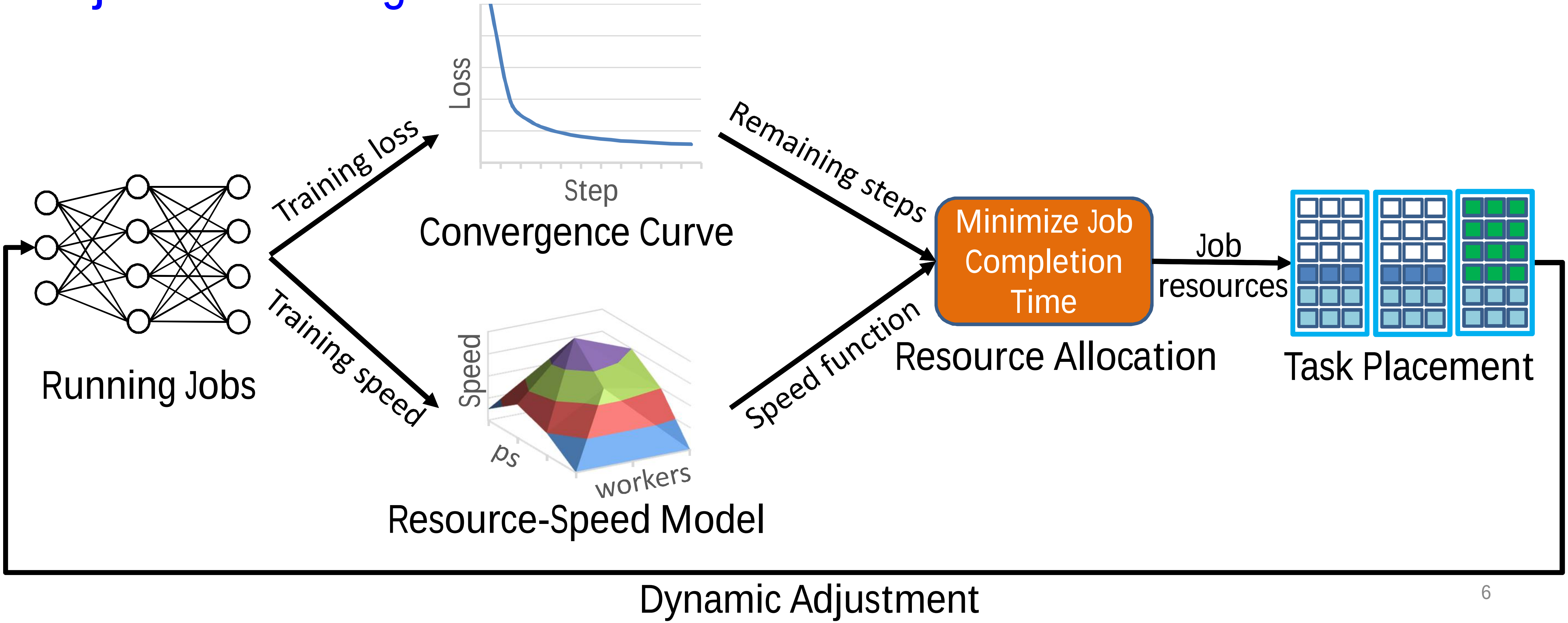
No linear speed improvement, even slower



ps:worker = 1:1

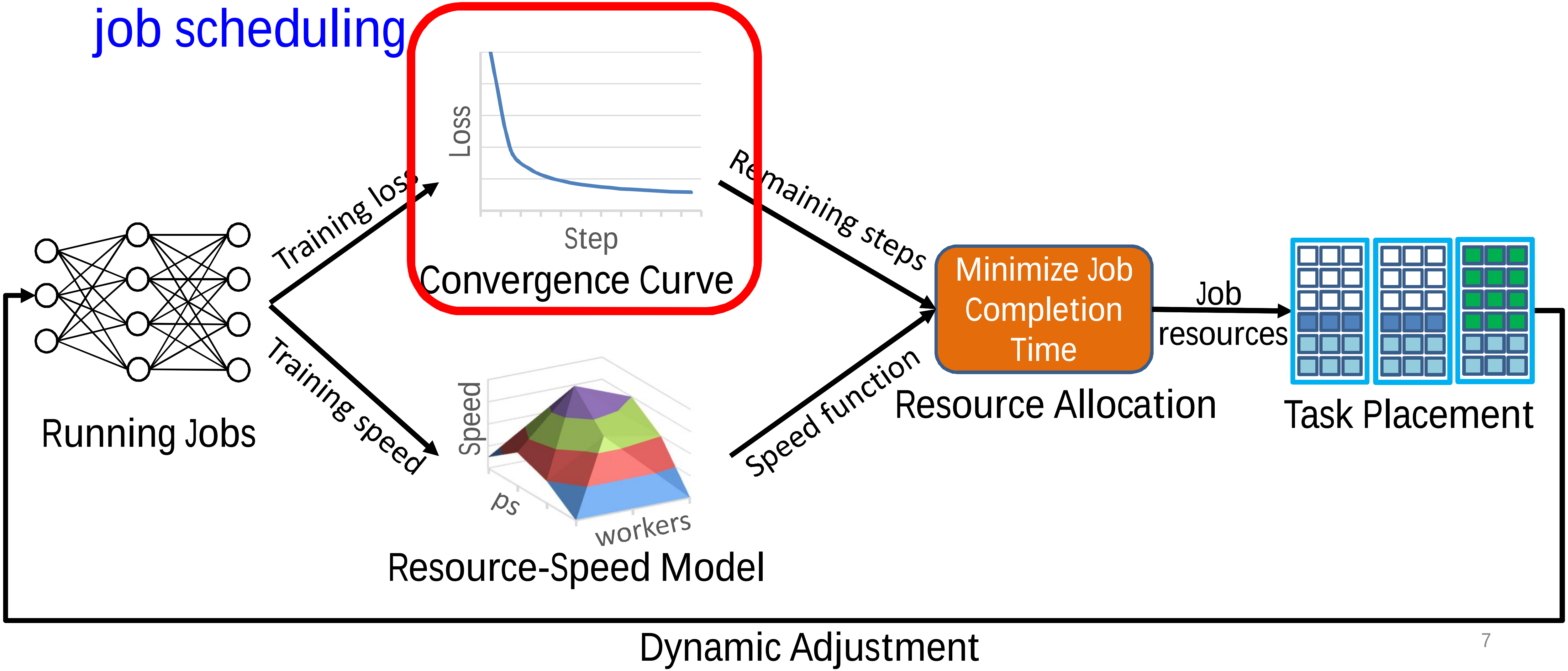
Optimus in a Nutshell

- Our approach: exploit DL/framework characteristics to customize job scheduling



Optimus in a Nutshell

- Our approach: exploit DL/framework characteristics to customize job scheduling

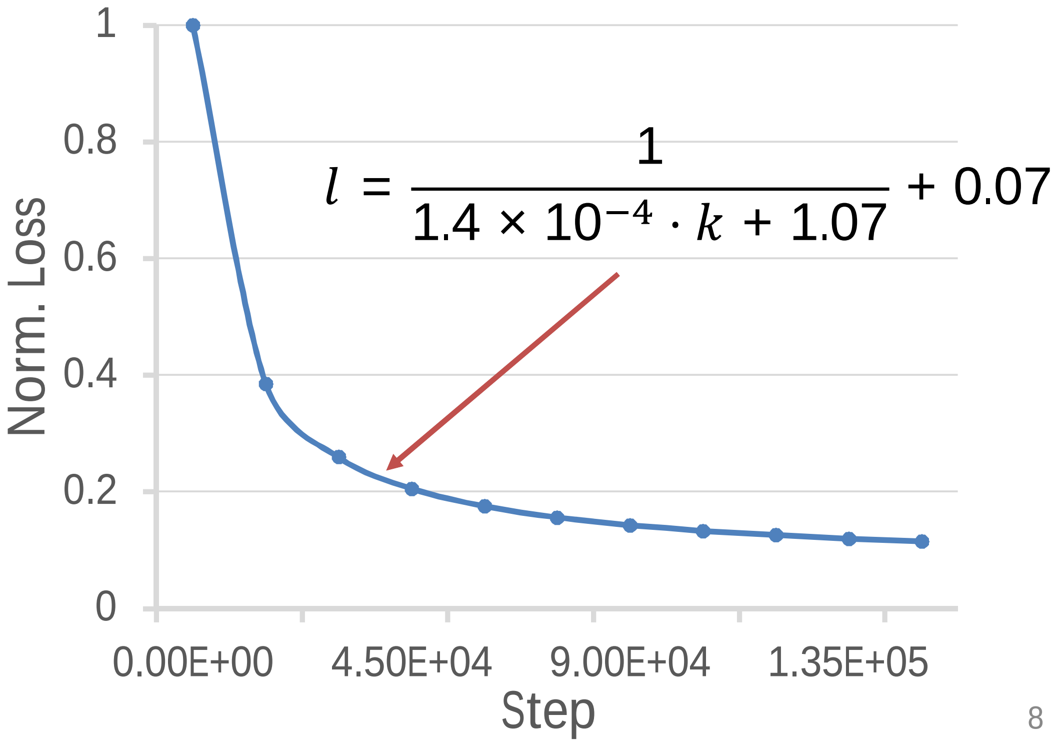


Learning the Convergence Curve

- Most DL jobs use SGD to update parameters
  - Converge at a rate of $O(1/k)$ in terms of step k
  - Estimate training loss l as a function of step k

$$l = \frac{1}{\beta_0 \cdot k + \beta_1} + \beta_2$$

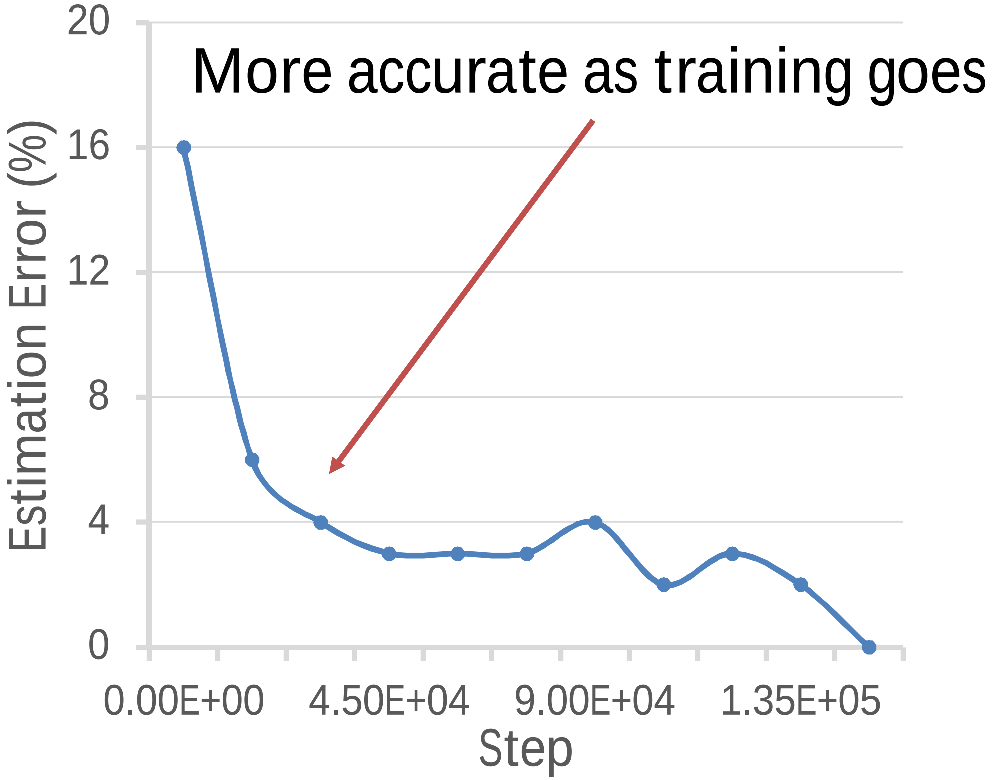
loss step coefficients



Learning the Convergence Curve

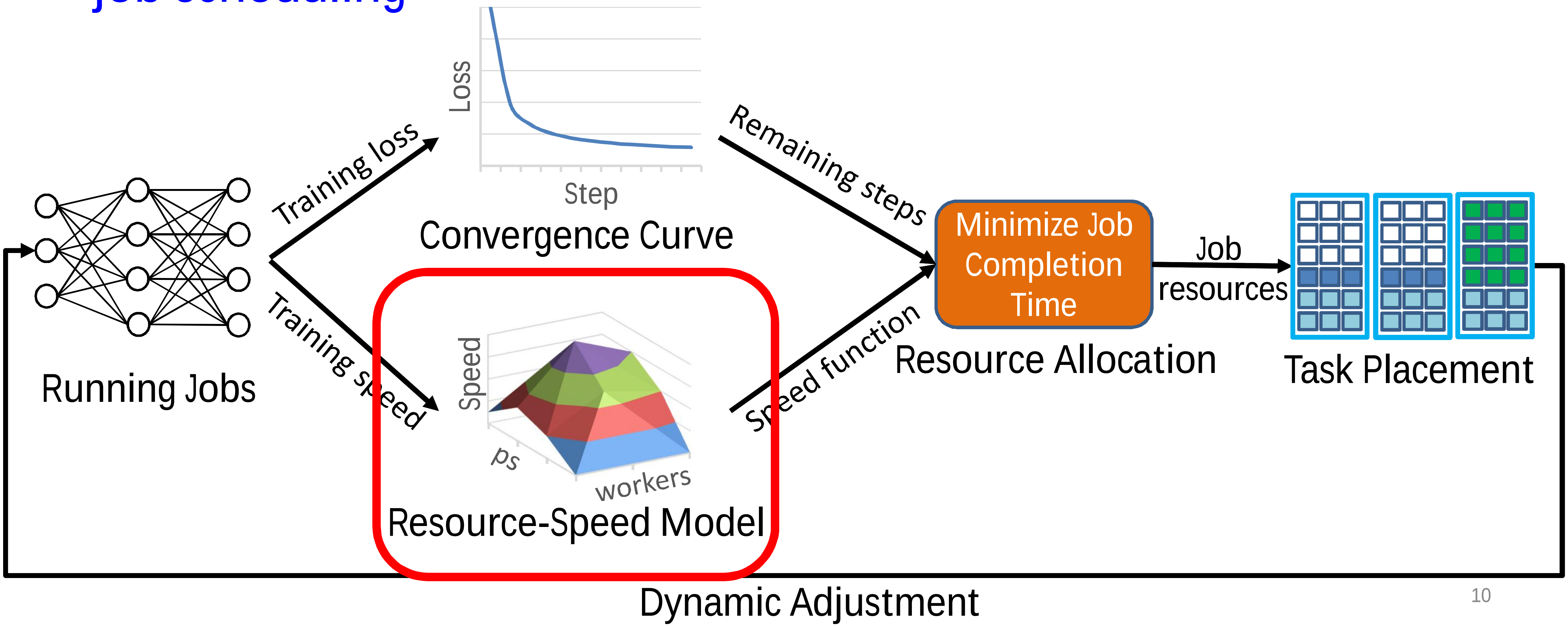
- Online fitting
  - Collect and preprocess training loss
  - Use non-negative least squares solver to find best $\boldsymbol{\beta}$ so far
  - Estimate remaining steps to converge

$$\underbrace{l}_{\text{loss}} = \frac{1}{\beta_0 \cdot \underbrace{k}_{\text{step}} + \beta_1} + \underbrace{\beta_2}_{\text{coefficients}}$$



Optimus in a Nutshell

- Our approach: exploit DL/framework characteristics to customize job scheduling



Resource-Speed Modeling

- Build A performance model for parameter server architecture

$$\boxed{T} = \max_{p}\left[\boxed{m \cdot T_{forward} + T_{back}} + \boxed{2\frac{s/p}{B/w_p'}} + \boxed{T_{update} \cdot \frac{w_p'}{p}} + \boxed{\delta \cdot w + \delta' \cdot p}\right]$$

time of one step | gradient computation | data tar… | …erhead

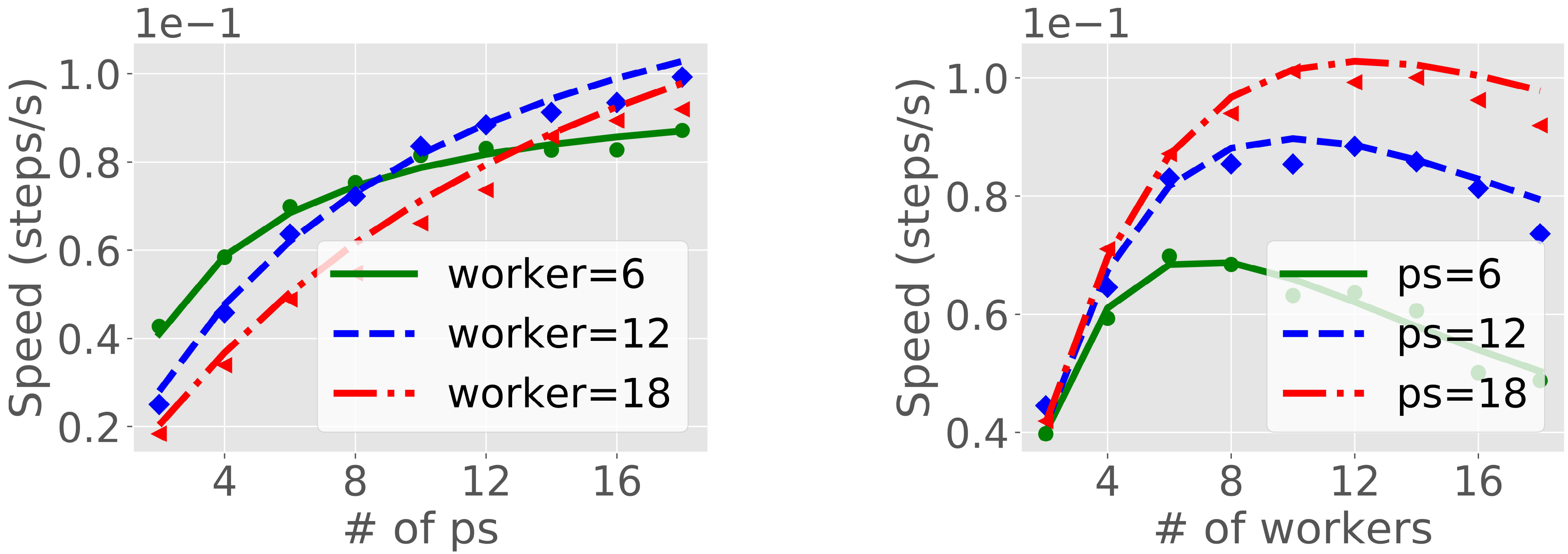
- Derive training speed $f(p,w)$ as … ers of parameter servers p and work ers w

Replace unknown constants with coefficients

$$f(p,w) = \left(\theta_0 \cdot \frac{M}{w} + \theta_1 + \theta_2 \cdot \frac{w}{p} + \theta_3 \cdot w + \theta_4 \cdot p\right)^{-1}$$

Resource-Speed Modeling

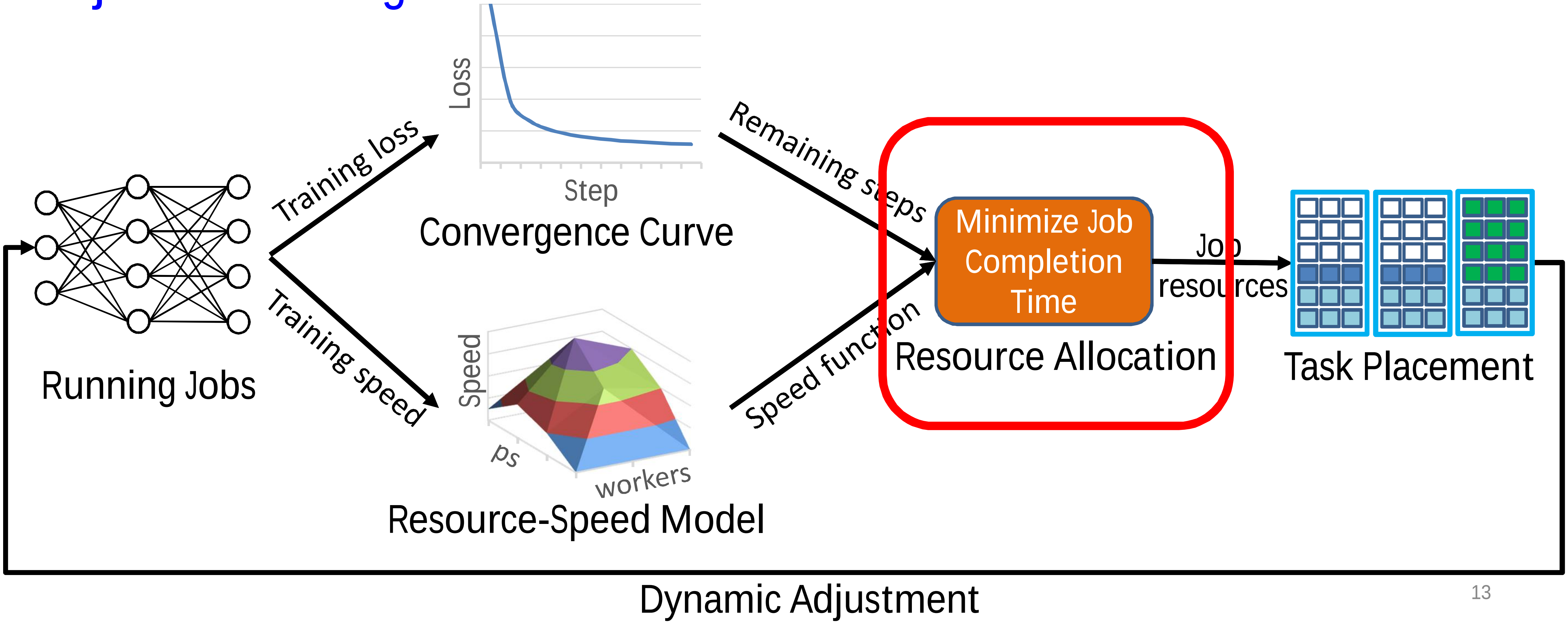
- Speed function fitting
  - Collect data points $(p, w, f(p, w))$ from several sample runs
  - Get a less than 10% error using 10 data points



$$f(p, w) = \left(\frac{40.8}{w} + 4.92 \cdot \frac{w}{p} + 0.02 \cdot p + 2.78\right)^{-1}$$

Optimus in a Nutshell

- Our approach: exploit DL/framework characteristics to customize job scheduling



Resource Allocation Problem

- Decide the numbers of parameter servers and workers for each job based on performance models

minimize $\sum_{j \in J} t_j$ → Minimize job completion time

remaining steps predicted by the convergence model

subject to: $t_j = \dfrac{\boxed{Q_j}}{\boxed{f(p_j, w_j)}} \quad \forall j \in J$ → job remaining time

training speed estimated by the speed function

$\sum_{j \in J} (w_j \cdot O_j^r + p_j \cdot N_j^r) \le C_r \quad \forall r \in R$ → capacity constraint

$p_j \in Z^+, w_j \in Z^+ \quad \forall j \in J$

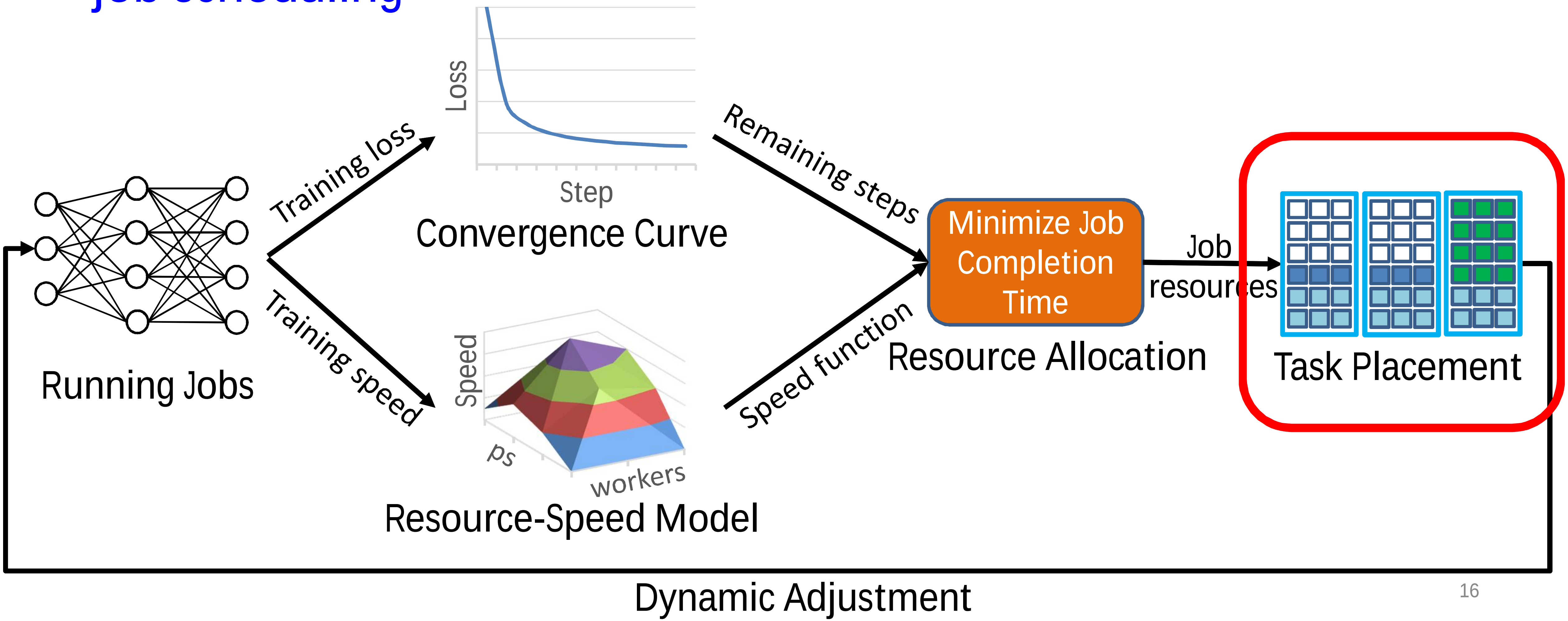
p_j: number of parameter servers of job j w_j: number of workers of job j

Greedy Resource Allocation Algorithm

- Marginal gain: reduced job completion time per unit resource
- In each iteration:
  - Try to increase one parameter server or one worker for each job and calculate the marginal gain
  - The job with highest marginal gain is selected
  - Allocate one parameter server or worker depending on which brings higher gain
  - Update marginal gain and available resources
- Stop when some resource is used up, or the marginal gain is non-positive.

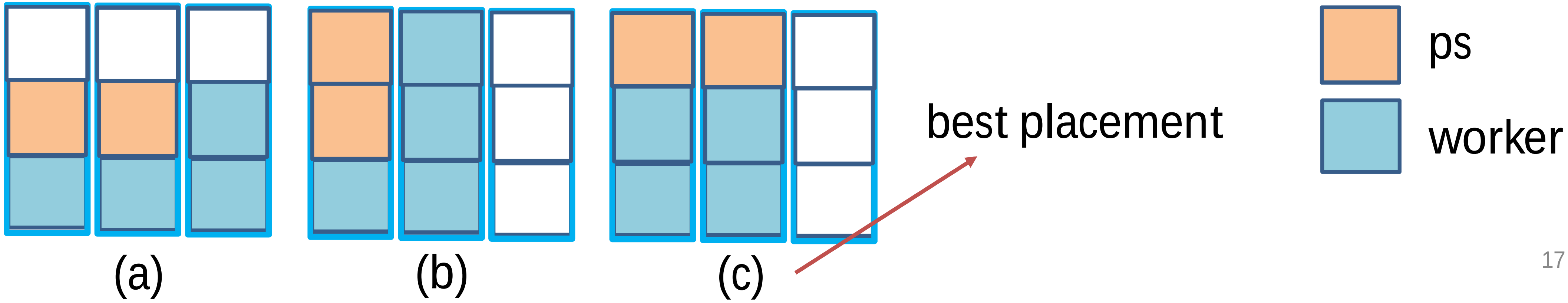
Optimus in a Nutshell

- Our approach: exploit DL/framework characteristics to customize job scheduling



Task Placement

- Decide the optimal placement given the numbers of parameter servers and workers of a job
  - Minimize communication overhead, i.e., cross-server data transfer
- Optimal placement principles
  - Co-locate parameter servers and workers
  - Each physical servers hold the same number of parameter servers and workers



Implementation

- Load balancing on parameter servers
  - Uneven parameter assignment among parameter servers
  - Best fit decreasing algorithm to balance load
- Elastic training on MXNet
  - Checkpoint model and restart job
- Scheduling on Kubernetes
  - Run parameter servers and workers in containers
  - Deploy Optimus as a normal pod

Evaluation

- Testbed
  - 13 servers
- Trace
  - 9 types of DL jobs
- Baselines
  - DRF
  - Tetris
- Metrics
  - Average Job Completion Time (JCT)
  - Makespan

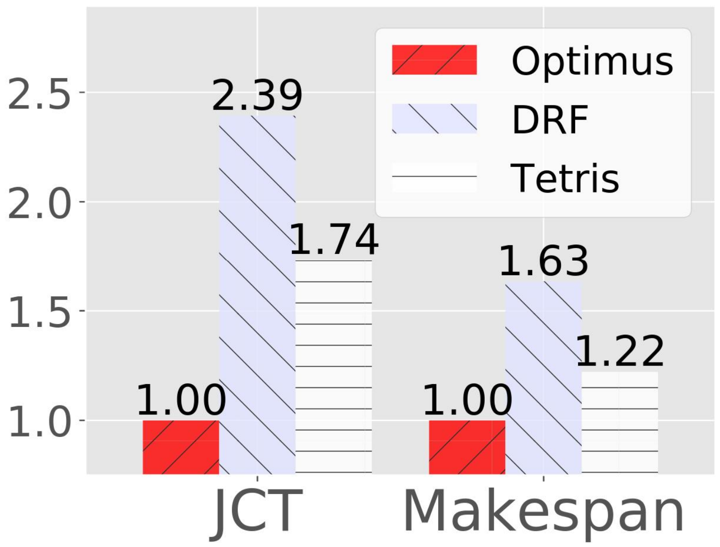
Model	# of parameters (Million)	Net-work	Application domain	Dataset
ResNet-50	25	CNN	image classification	ImageNet
ResNext-100	1.7	CNN	image classification	CIFAR10
Inception-BN	11.3	CNN	image classification	Caltech
KAGGLE	1.4	CNN	image classification	Kag-NDSB1
CNN-rand	6	CNN	sentence classification	MR
DSSM	1.5	RNN	word representation	text8
RNN-LSTM	4.7	RNN	language modeling	PTB
DS2	38	RNN	speech recognition	LibriSpeech
Seq2Seq	9.1	RNN	machine translation	WMT17

Evaluation

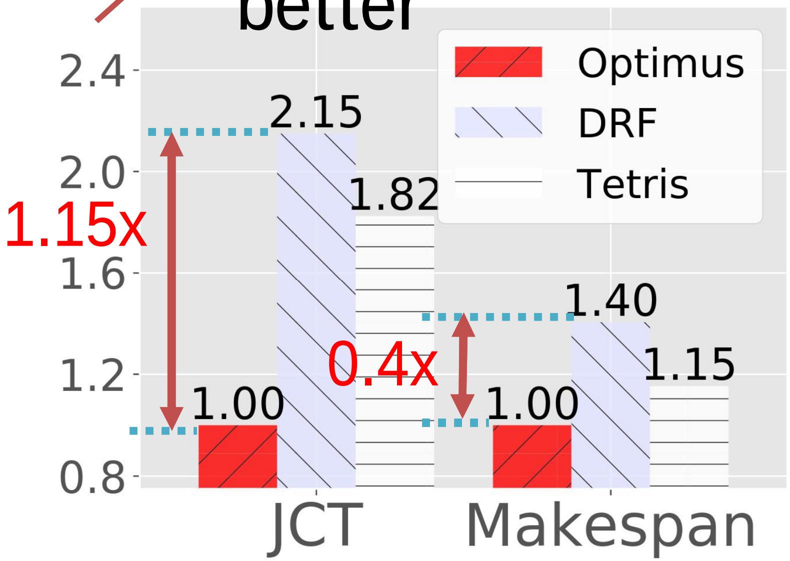
- Performance comparison under different job arrival distributions

$$\frac{\text{DRF/Tetris}'\ \text{Metric}}{Optimus'\ \text{Metric}}$$

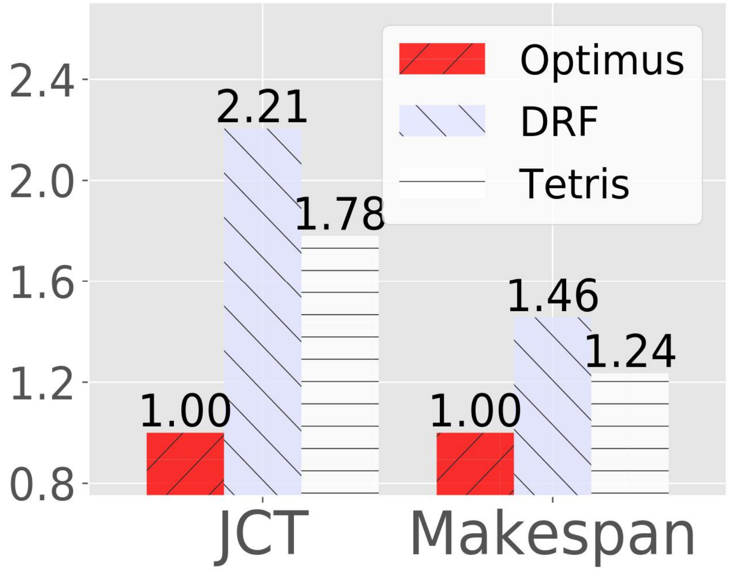
the lower the better



Uniform



Poisson

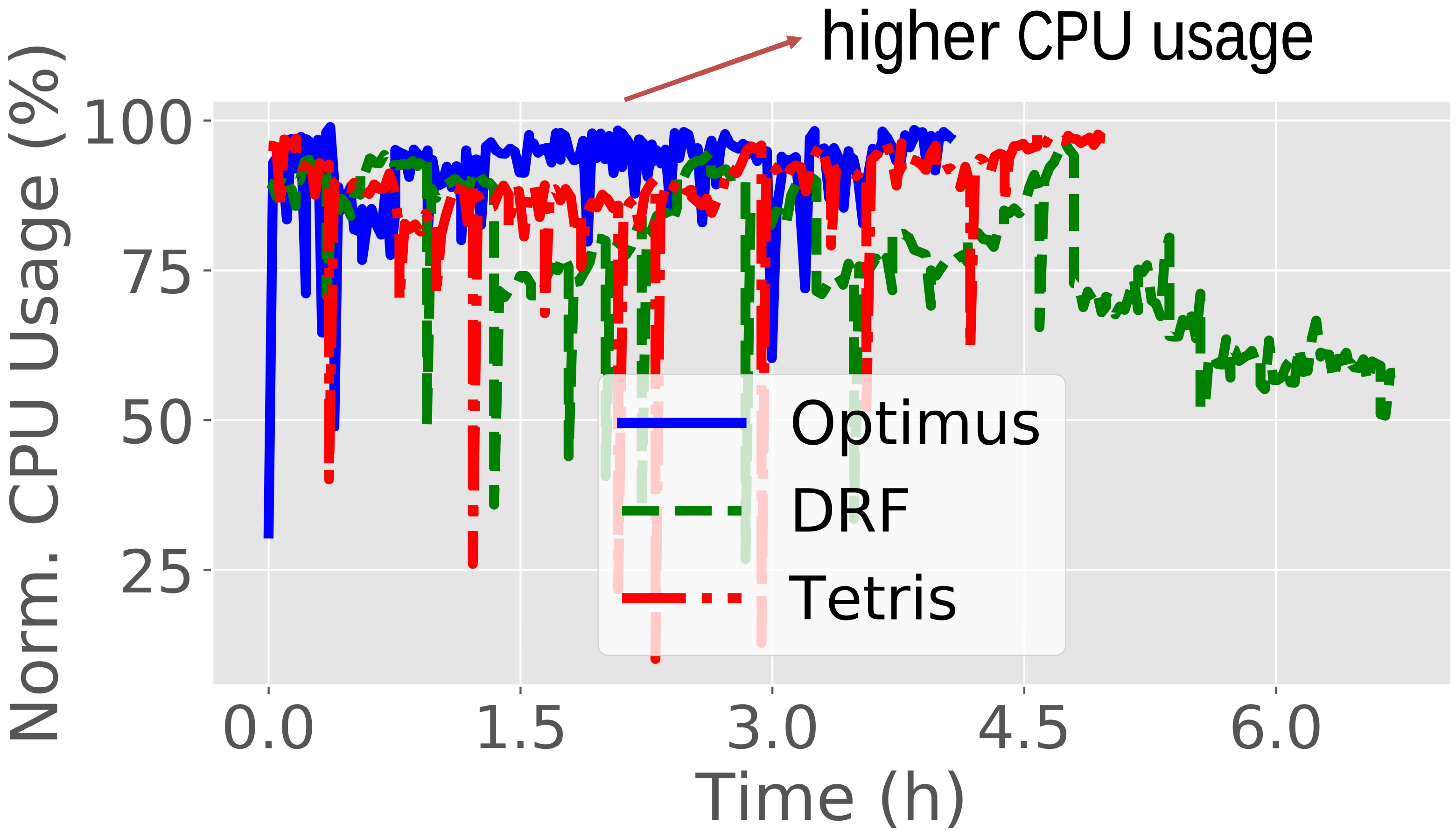


Google trace

Over 1x and 0.4x speedup in JCT and Makespan compared to DRF

Evaluation

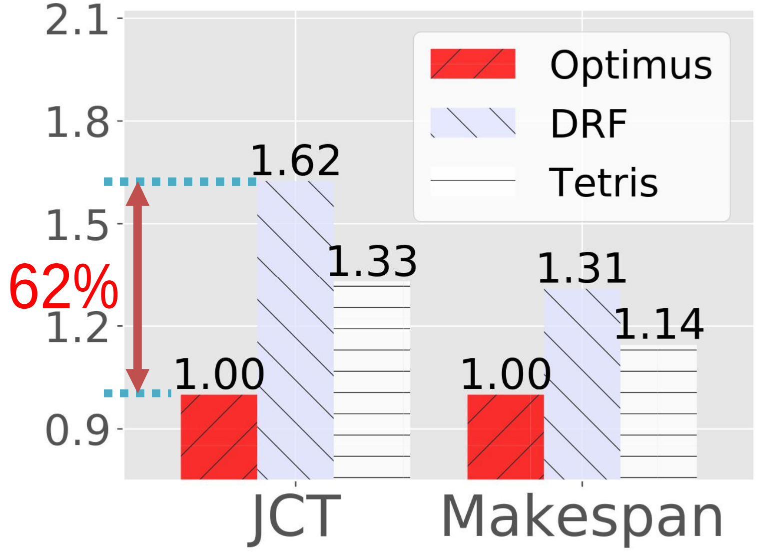
- Normalized CPU usage on parameter servers



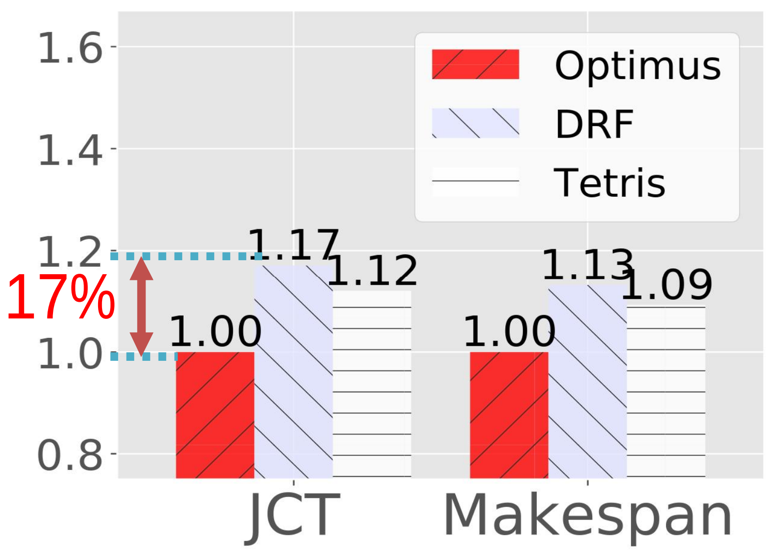
Optimus has higher resource efficiency

Evaluation

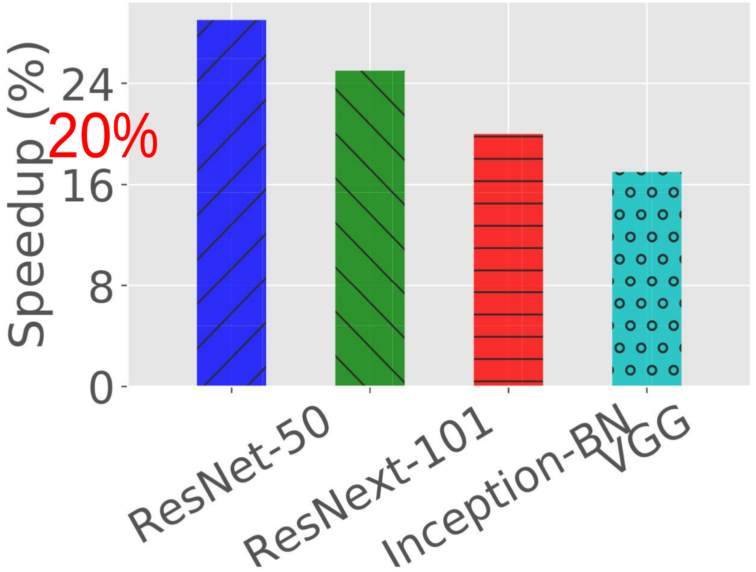
- Performance contribution of each component



Resource allocation



Task placement



Parameter server load balancing

Resource allocation, task placement and parameter server load balancing contribute by 62%, 17%, 20% respectively

Conclusion

- *Optimus*: a customized cluster scheduler targeting high training performance and resource efficiency
  - The core is the performance model for DL jobs

- Future work
  - Extend *Optimus* to handle more DL/ML workloads
  - Dealing with inaccurate performance model for robust scheduling

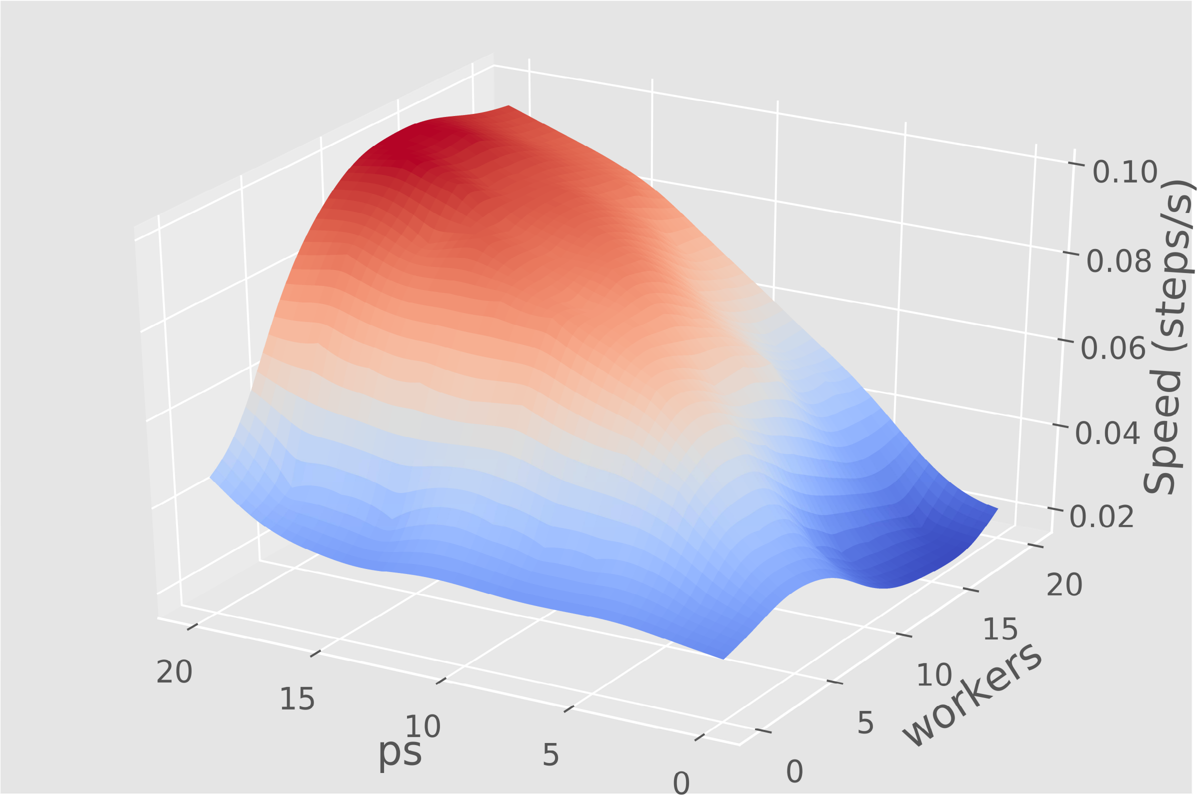
Questions

yhpeng@cs.hku.hk

Backup

Resource-Speed Modeling

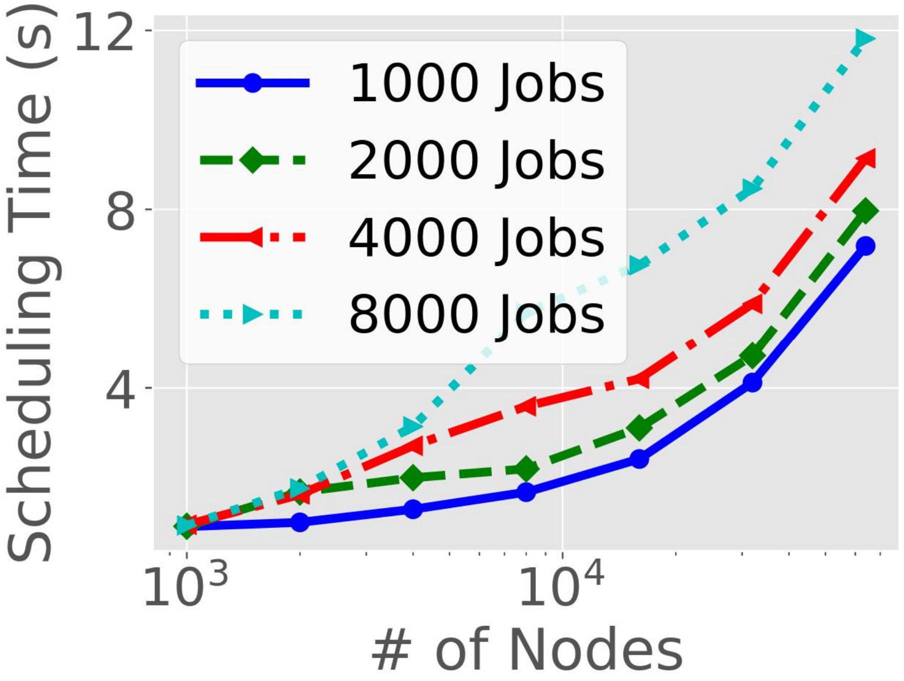
- Speed-resource function



$$f(p, w) = \left(\frac{40.8}{w} + 4.92 \cdot \frac{w}{p} + 0.02 \cdot p + 2.78 \right)^{-1}$$

Evaluation

- Scalability



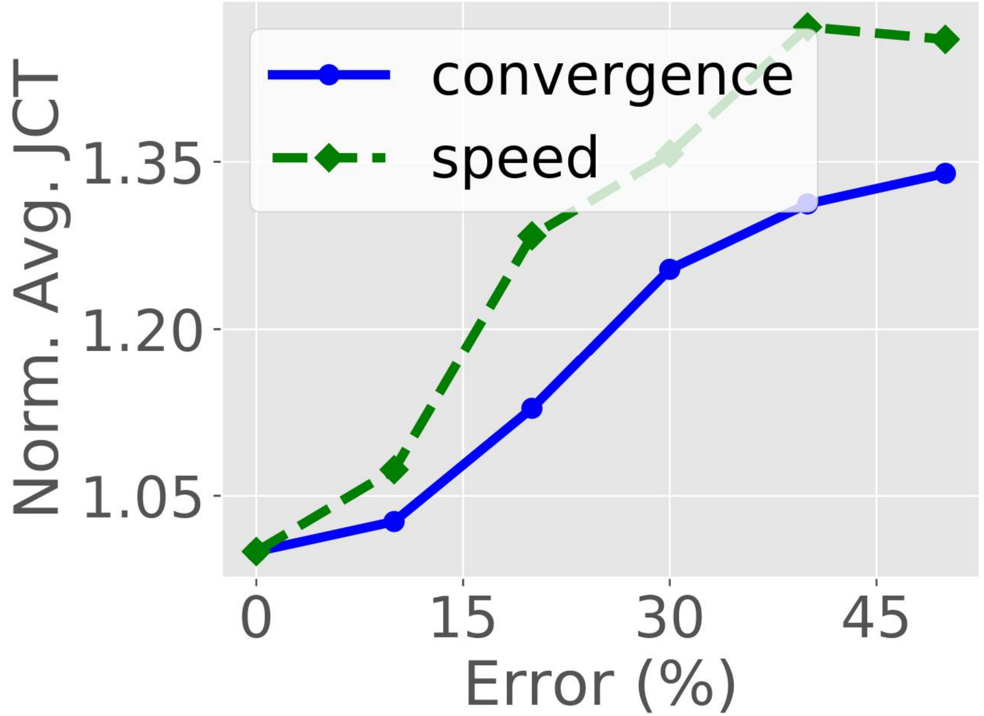
4000 jobs in 5 seconds on a cluster of 10000 nodes

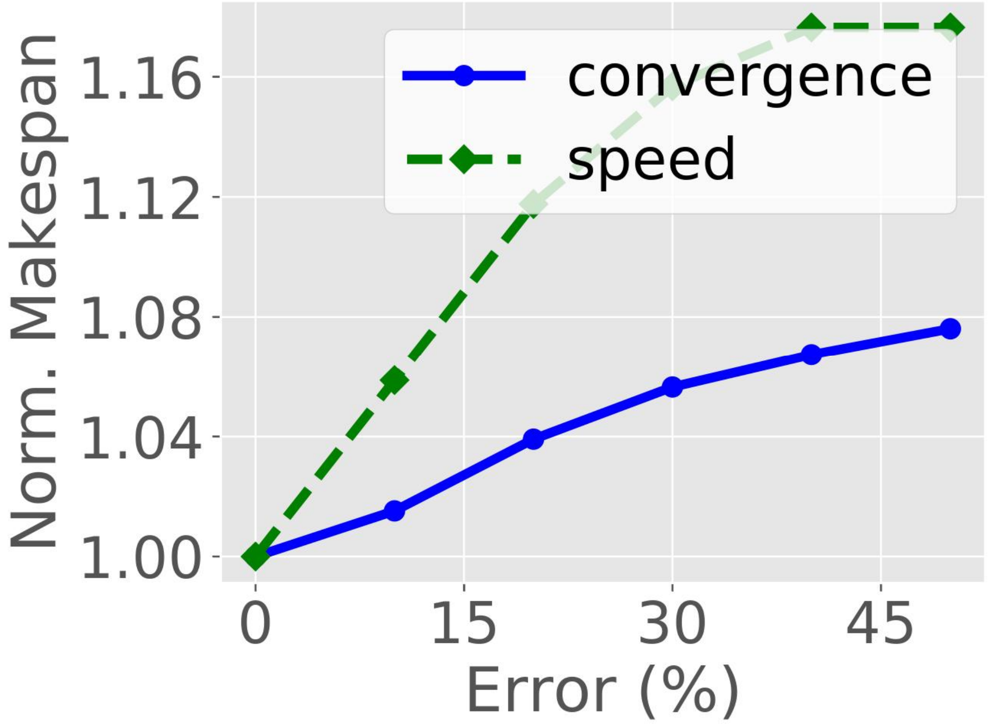
- Overhead

2.54% of training time

Evaluation

- Sensitivity to estimation error





20% performance gap compared to no estimation error