# Scavenger: A Cloud Service for Optimizing Cost and Performance of ML Training

Sahil Tyagi and Prateek Sharma

INDIANA UNIVERSITY

# Distributed Training in the Cloud

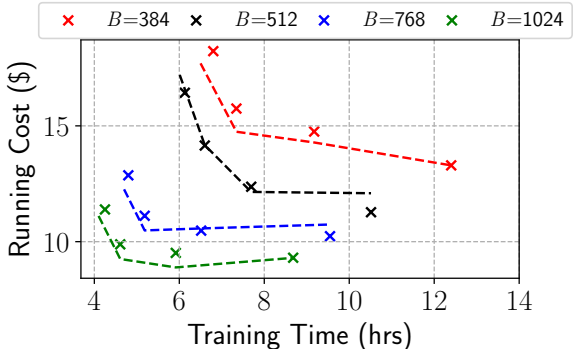
- Variety of VM types and sizes in the cloud
- Distributed training has configuration parameters that affect time and cost:
  - **Cluster-size (K)**: # workers training in parallel
  - **Batch-size (B)**: cumulative batch-size processed in a training step
- Each **(K, B)** results in different cost-time tradeoffs!

# What is the right job configuration?

- Cloud resources and a training job present a huge configuration space!

- **Scavenger accurately predicts performance and cost of different model and (K,B) configurations**

- **Uses an Online approach:**
  - **Leverages iterative nature of training**
  - **No need for prior offline profiling**
  - **Works in a black-box manner**
  - **Tensorflow-based middleware**



# Distributed Data-Parallel Training

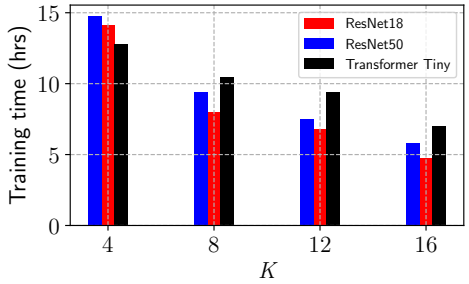
- With Parameter server (PS), workers train independently on i.i.d. data by pulling updates from a central server and push updates for aggregation: $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{1}{K} \frac{1}{b} \sum_{k=1}^{k=K} \nabla f(\mathbf{x}_{k,t}) \rightarrow$ elastic

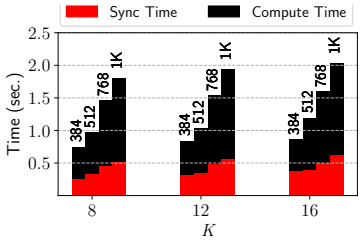
- Each training step is an iterative, repetitive process comprised of:
  - **Computation**: In forward-backward pass, evaluate loss and compute gradients
  - **Communication**: push/pull updates to/from the PS

- Tradeoffs in synchronous data-parallelism: **K** → **more communication** and **B** → **more computation and memory consumption**

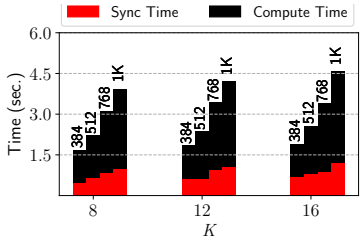
# Horizontal scaling: Increasing K

- Training can be scaled out by adding more workers and **increasing K**
- Training time to desired accuracy reduces as K increases
- However, **scaling is not linear**; increasing K by 4x does not reduce training time proportionately! → **due to communication overheads!**

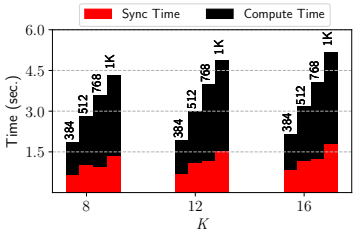



ResNet-18



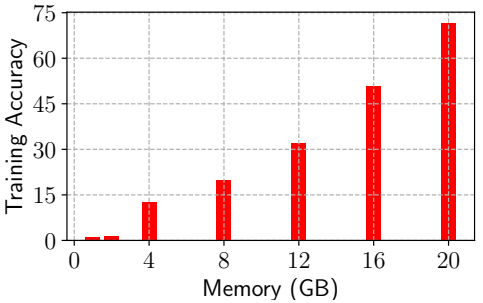

ResNet-50

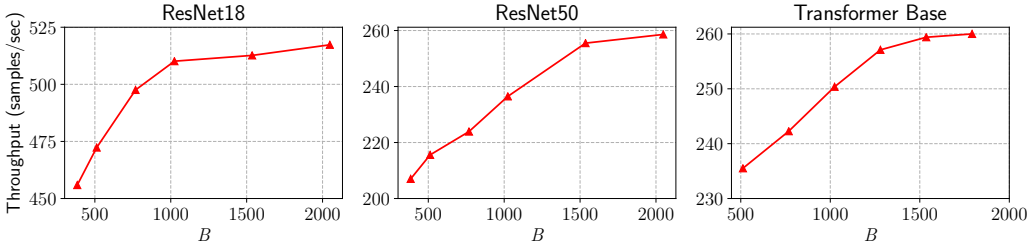


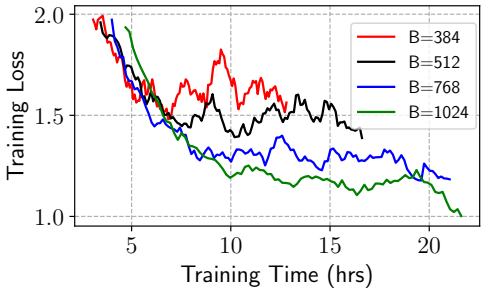

Transformer base

# Vertical scaling: Increasing B

- ML training can be scaled up by **increasing batch size B** (reducing communication overhead)
- Scaling up VMs by increasing memory allows for larger batch-sizes that converge better
- DNNs converge faster on larger B (until device saturation)



# Performance tradeoffs $\Longrightarrow$ **Parallel and Statistical efficiency**

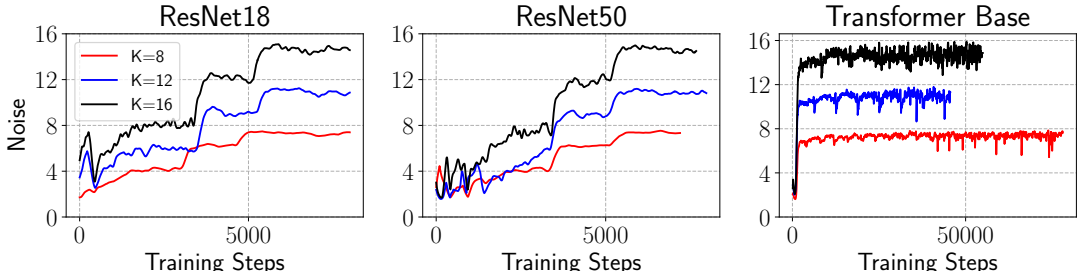
- **Parallel efficiency**: Amdahl's law → lower scaling with larger workers
- **Statistical efficiency**:
  - Specific to deep learning due to stochastic nature of SGD
  - Work performed on all iterations **not** equally useful
  - Gradients computed on a small mini-batch of data can be **noisy**
  - Gradient noise increases the iterations needed for convergence

# How to measure Statistical Efficiency?

- **SGD noise compares gradient variance among different workers**

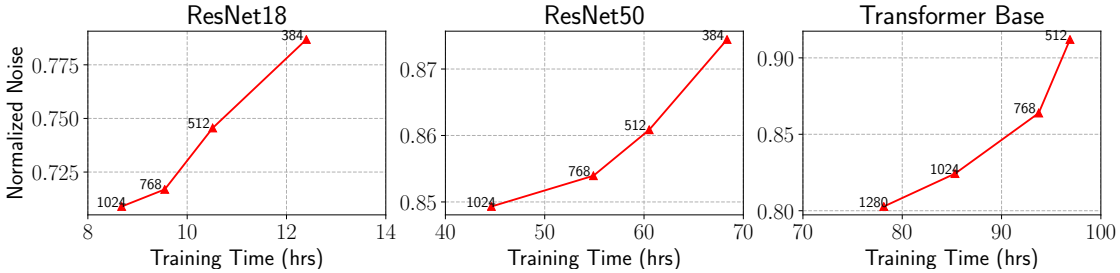
$$\gamma(t) = \frac{\mathbb{E}[\frac{1}{K}\sum_{k=1}^{K}||g_t^{(k)}||^2]}{\mathbb{E}[||\tilde{g}_t||^2]}$$

- Integrates easily into the backpropagation routine



# SGD Noise efficacy to predict performance

- Gradients are volatile in early stages so noise is low
- Once stabilized, normalize noise by K to compare performance across different Bs
- SGD Noise reduces as B increases and thus training time decreases



# Scavenger Performance and Cost Model

$$\text{Cost} = (\#\ \text{VMs}) \times \text{Time} \times \text{price}_{VM}$$

$$\text{Time} = t_{iteration} \times (\#\ \text{Iterations})$$

- **Statistical efficiency**: from empirically measured SGD noise
- **Parallel efficiency**: from running/inferring for a (K, B) configuration

# Performance and Cost model

- **Cost** proportional to **number of steps** and **time per-step; Per-step time** comprises of **computation** and **communication time**.
- Job profiling and searching:
  - **Full/Offline search**:Run each configuration briefly; use checkpointing to resume
  - **Online search**:Run on extremums and interpolate from linear model
  - **No-search**:Use DNN-agnostic approximate model for all jobs

# Modeling Parallel Performance

- Model training is repetitive and parallel performance identical over the iterations
- Iteration time comprises of compute and communication

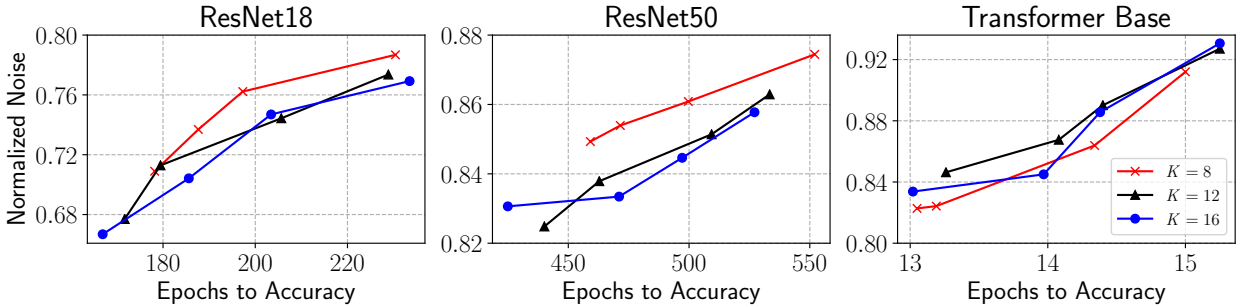
$$t_{iteration} = t_{compute} + t_{communication}$$

- **Computation time**: $t_{compute} \propto$ batch-size (B)
- **Communication time**: $t_{communication} \propto$ # workers $(K)$

# Modeling Statistical Performance

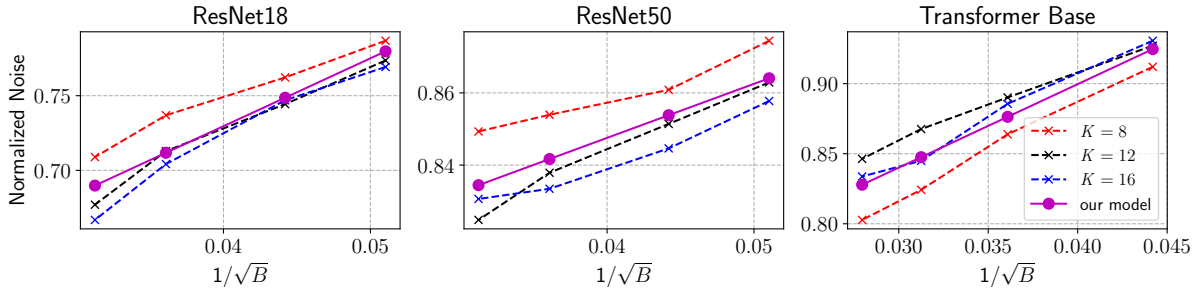
- **Need to know the epochs needed to converge to a specific accuracy target**
- Relative to Full GD taking $e^*$ epochs to converge, training with B takes more epochs to converge:

$$e^{(K,B)} = e^* + \theta \cdot \gamma_{(K,B)} \implies e^{(K,B)} \propto \gamma_{(K,B)}$$



# Modeling Statistical Performance

- For partial search, no need to profile each configuration
- Model noise as a function of B and fit a **global** linear model on a DNN: $\gamma_{(K,B)} \propto \frac{1}{\sqrt{B}}$
- Fit a **global** linear model for different DNNs



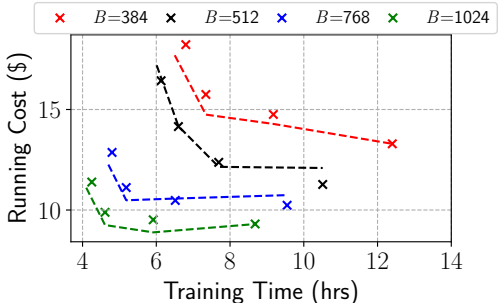
# Resource Allocation policy

- Build time-cost tradeoff curves using parallel and statistical performance model using full, partial or no-search policy
- Choose configuration that optimizes either **time, cost or in-between (knee-point)**
- Constraints on (K,B) influence the degree of exploration space → **specified by user**

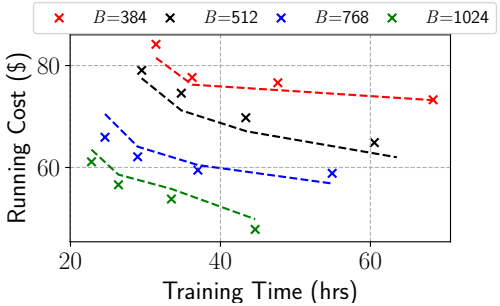
# Cost-Time Tradeoffs

- Per-VM cost is fixed

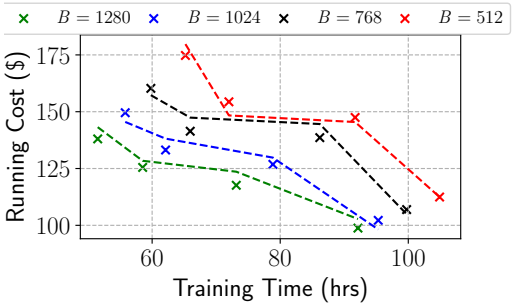
ResNet-18




ResNet-50




Transformer base

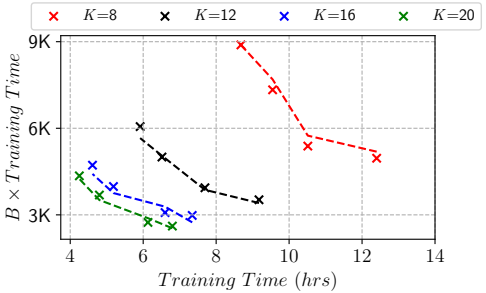



Cost-time tradeoffs to reach same convergence for various B across decreasing cluster sizes [20,16,12,8] on GCP. Dashed lines show the cost predicted by our full-search performance model

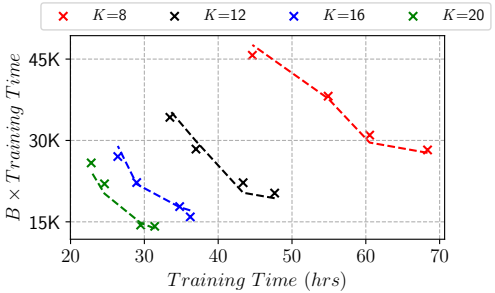
# Cost-Time Tradeoffs

- VMs are priced based on their allocated memory size

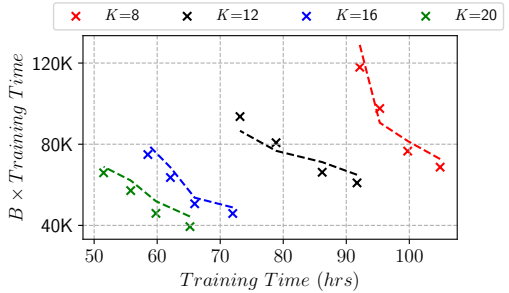
ResNet-18




ResNet-50



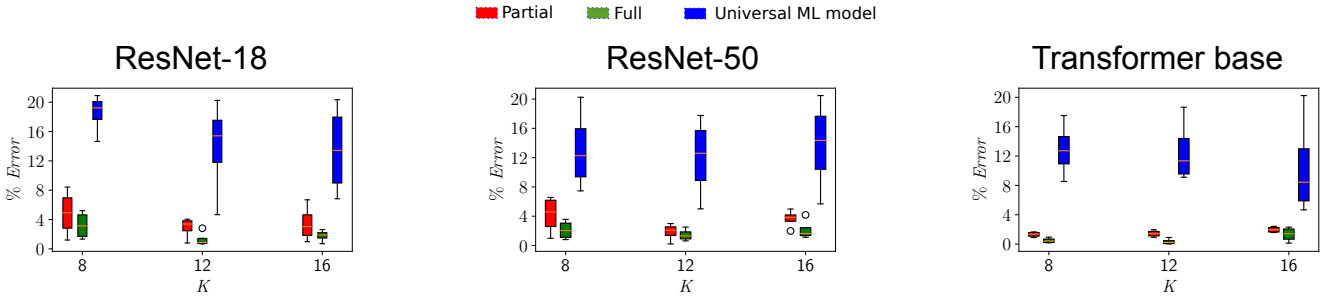

Transformer base




Scavenger works with different pricing policies. Here, VMs are priced based on their allocated memory size (proportional to B), resulting in different cost-time tradeoffs

# Model Error for different search strategies

- **Full**: runs every configuration for few iterations → **most accurate!**
- **Partial**: runs extreme (K,B) configurations and interpolates via performance models
- **Universal**: predicts time-cost in DNN-agnostic way → **least accurate!**




Prediction error of performance models. Error for full and partial search is low, and even acceptable for DNN-agnostic universal model.

# Thank you!

INDIANA UNIVERSITY