Understanding Stragglers in Large Model Training Using What-if Analysis

Jinkun Lin[1] , Ziheng Jiang[2] , Zuquan Song[2] , Sida Zhao[2] , Menghan Yu[2] , Zhanghan Wang[1] , Chenyuan Wang[2] , Zuocheng Shi[4] , Xiang Shi[3] , Wei Jia[2] , Zherui Liu[2] , Shuguang Wang[2] , Haibin Lin[2] , Xin Liu[2] , Aurojit Panda[1] , Jinyang Li[1]

[1]New York University, [2]ByteDance Seed, [3]ByteDance, [4]Zhejiang University

Motivation

- Stragglers often observed at ByteDance
  - What causes stragglers and their severity remained unknown

- Why prior work is lacking
  - FALCON [arXiv'24], MegaScale [NSDI'24], Imbue, Llama 3.1 [arXiv'24]
  - Only **scattered case-studies**

*Our goal: **studying stragglers** in LLM training using 5-month production traces from ByteDance cluster*

Outline

- How we analyze stragglers: what-if analysis
- Analysis result
- In-production use of the analyzer

What is a straggler?

- A worker that is slower than others at synchronization

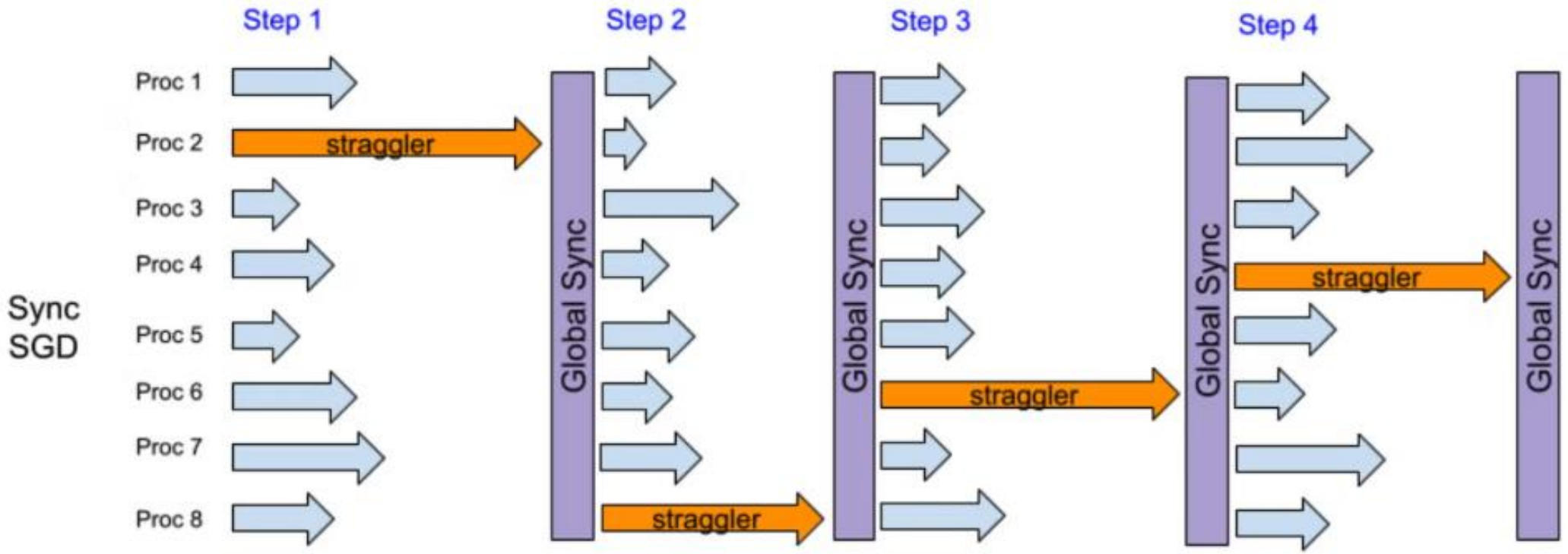
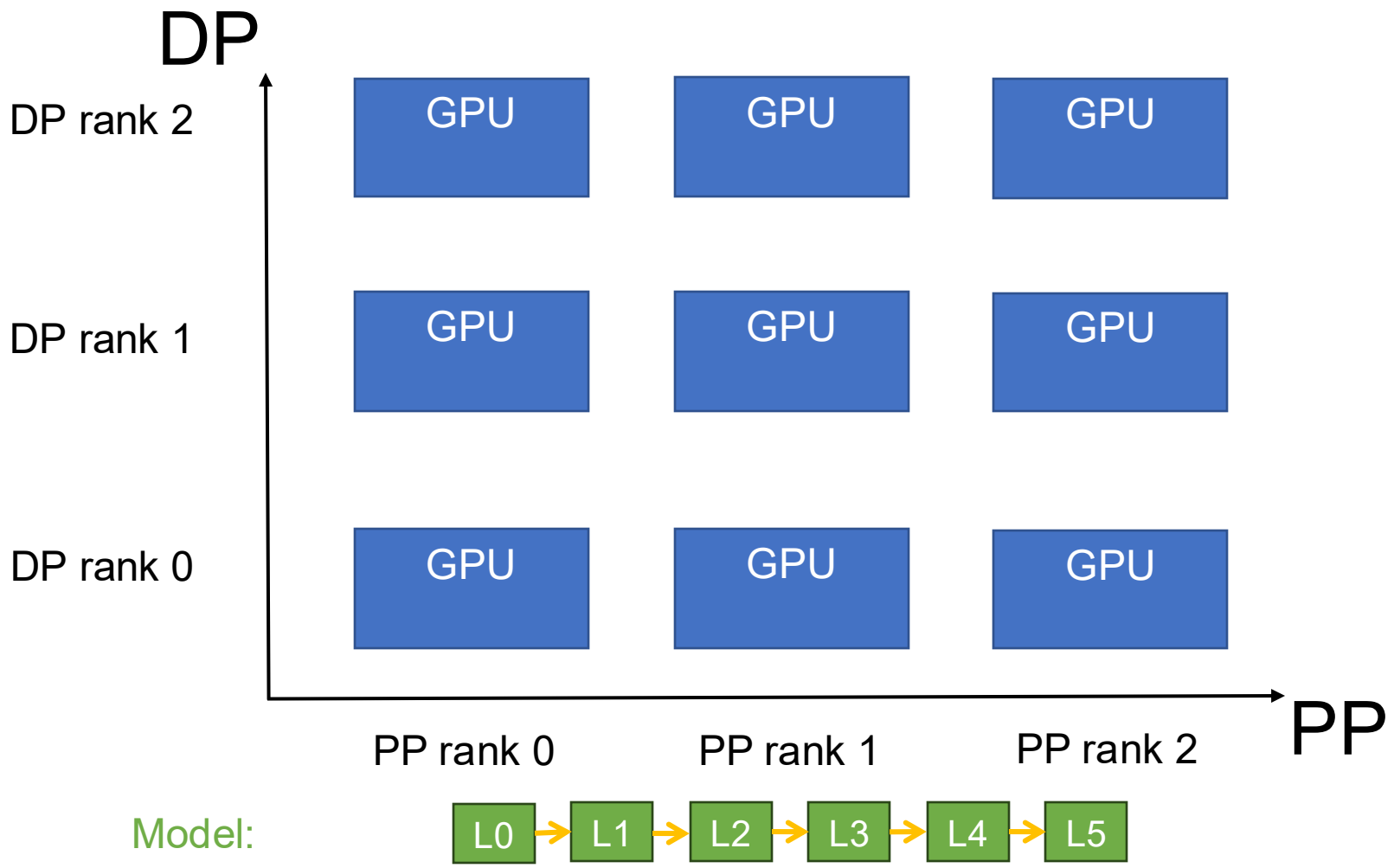


Image source: https://pytorch.org/blog/straggler-mitigation/

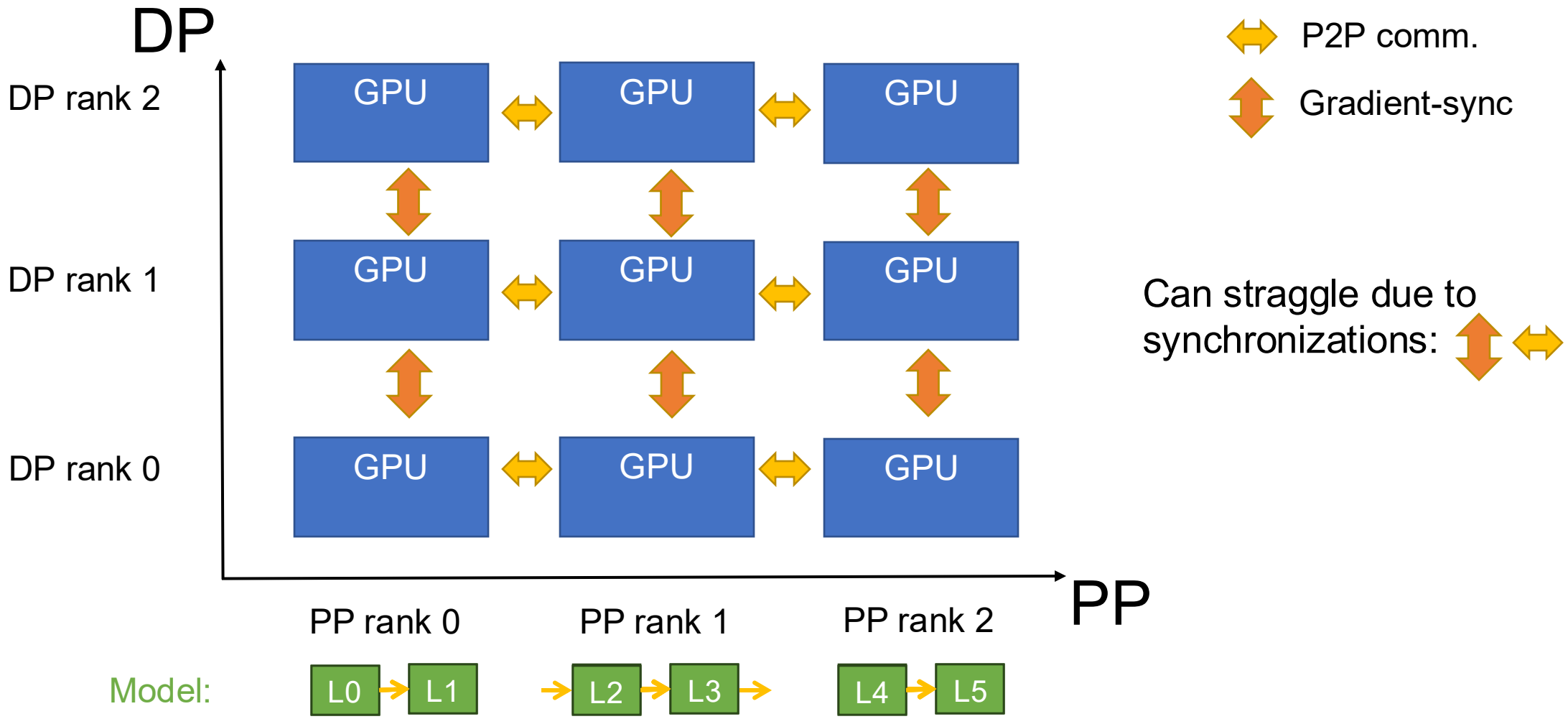
Why are stragglers relevant to LLM training?

- LLM training uses several parallelisms, eg, data + pipeline parallel:



Why are stragglers relevant to LLM training?

- LLM training uses several parallelisms, eg, data + pipeline parallel:

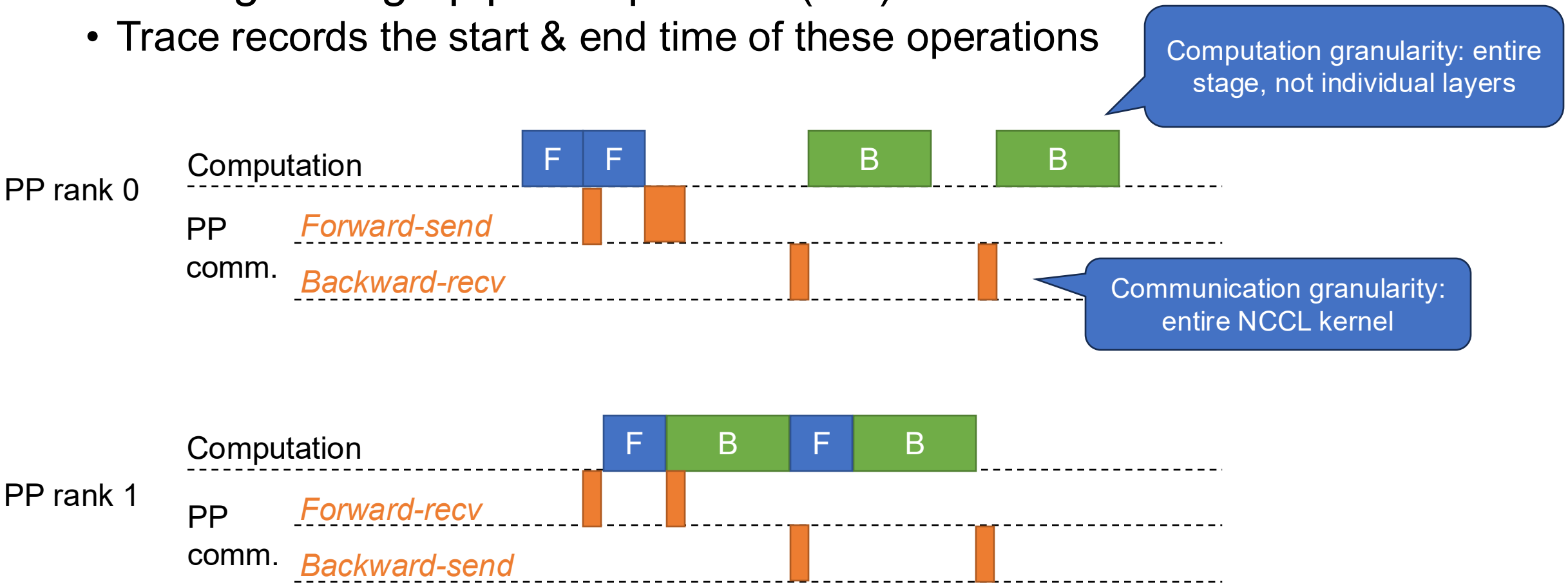


What do our traces capture?

- 3079 LLM pretraining jobs, collected from homogeneous clusters dedicated for LLM training
  - Job sizes range from 128 to >5k GPUs
  - Data, pipeline, tensor & context parallel
- Each job trace has ~10% sampled steps
  - Traced by in-house profiler *NDTimeline*
  - Contains timestamps of coarse-grained operations at ***every worker***

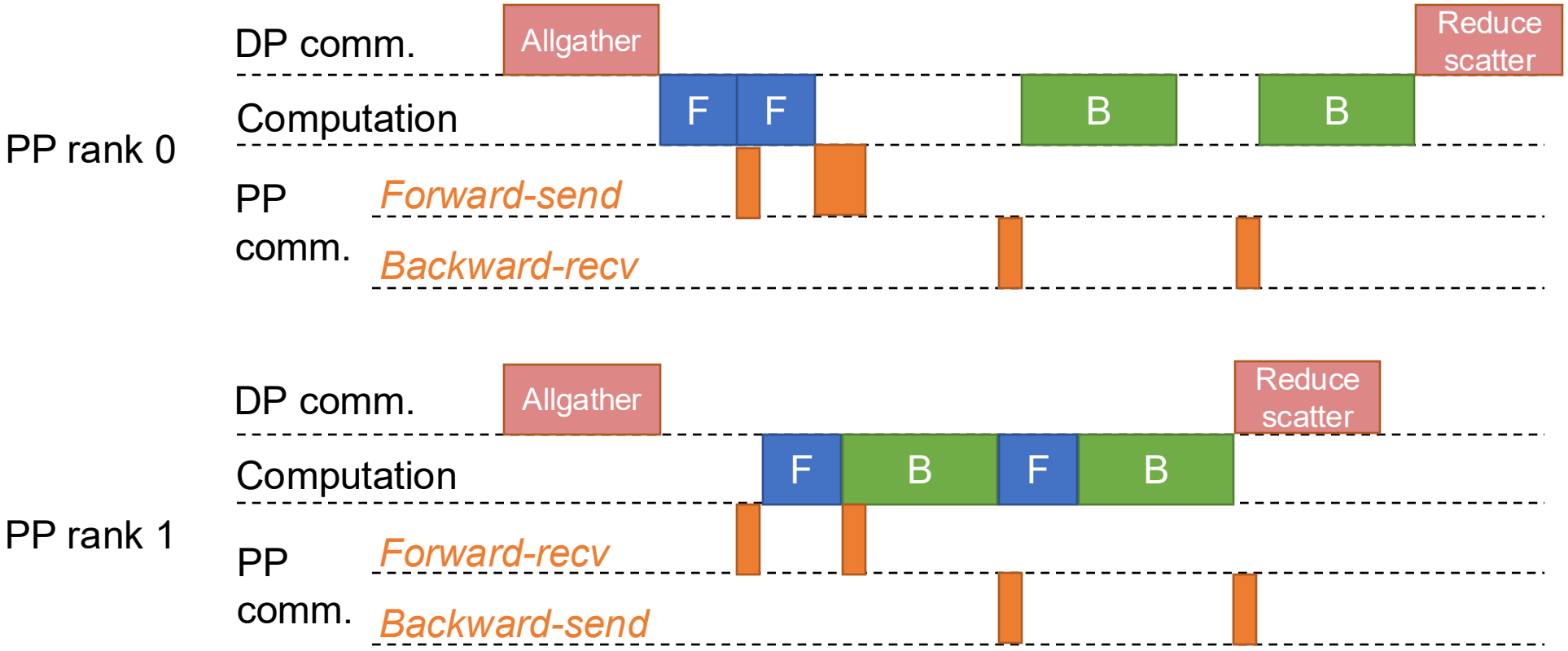
An Example in Timeline View

- Job using 2-stage pipeline parallel (PP) and 2 microbatches
  - Trace records the start & end time of these operations



An Example in Timeline View

- If the job further uses DP (ZeRO-1):



How to know if a job trace suffers from stragglers?

- Use counterfactual reasoning.
  - If a job were ***straggler-free***, then:
    - All forward-compute operations should take the same time.
    - All backward-compute …
    - All reduce-scatter...
    - ...
  - Since all workers progress in same speed

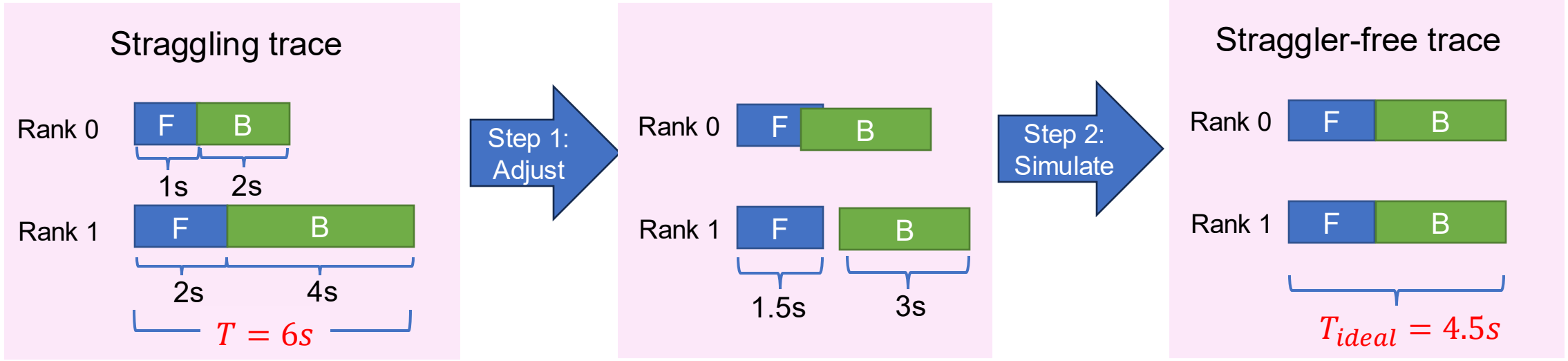
- A straggler-free job has the lowest job completion time: T_{ideal}

- Straggler-caused job slowdown $S = \frac{T}{T_{ideal}}$
  - T: actual job completion time

Calculate T_{ideal} with what-if simulation

- Step 1: Adjust every operation's duration to "ideal"
  - Average (median) for computation (communication)

- Step 2: Simulate the job-run using adjusted duration
  - Topological sort based on manually encoded dependency graph



Analysis Result & Findings

Questions to Answer in the Stragglers Analysis

- Prevalence and impact:
  - How many jobs are straggling?
  - How much slowdown do stragglers cause for each job?

- Symptom:
  - Temporal pattern: Do stragglers tend to be transient or persistent?
  - Spatial pattern: Do they tend to concentrate on a few workers, or uniformly affect all?

- Common root causes & their relative severity

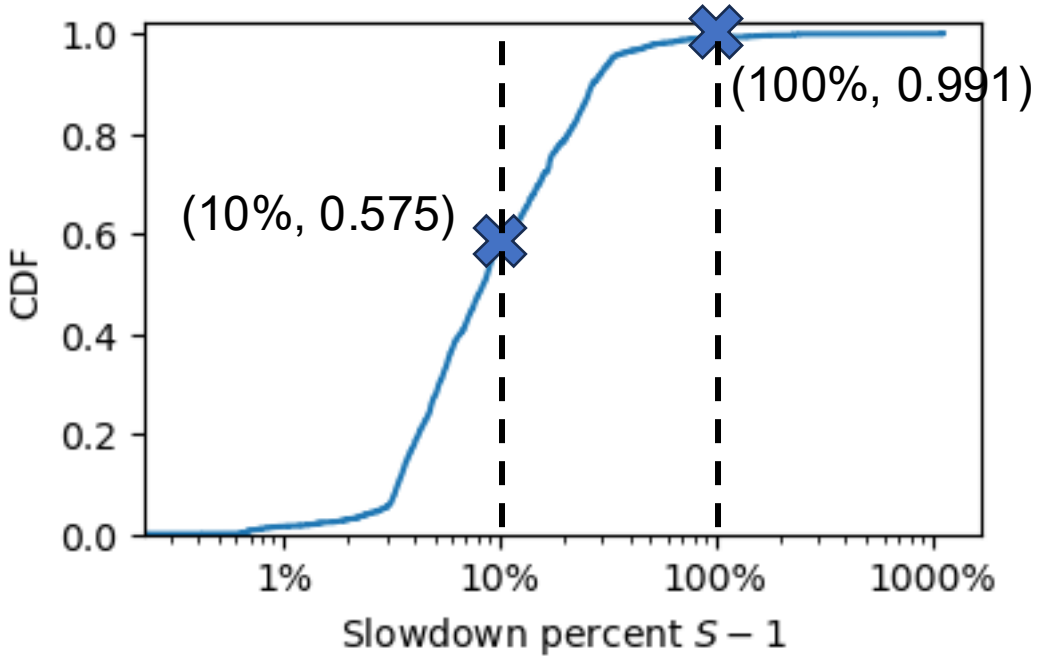
Overall Straggler Prevalence

- Figure: CDF of slowdown $S = \frac{T}{T_{ideal}}$ across all jobs
  - X-axis: slowdown in percentage = $100 \times (S - 1)$

- Stragglers are prevalent: 42.5% jobs straggle (def. $S > 1.1$)

- Average: $S = 1.15$

- Tail (~P99): $S = 2$ 😩



What leads to stragglers?

- Identified major causes by:
  - Sample large jobs with high slowdown
  - Manual inspect & run further analysis

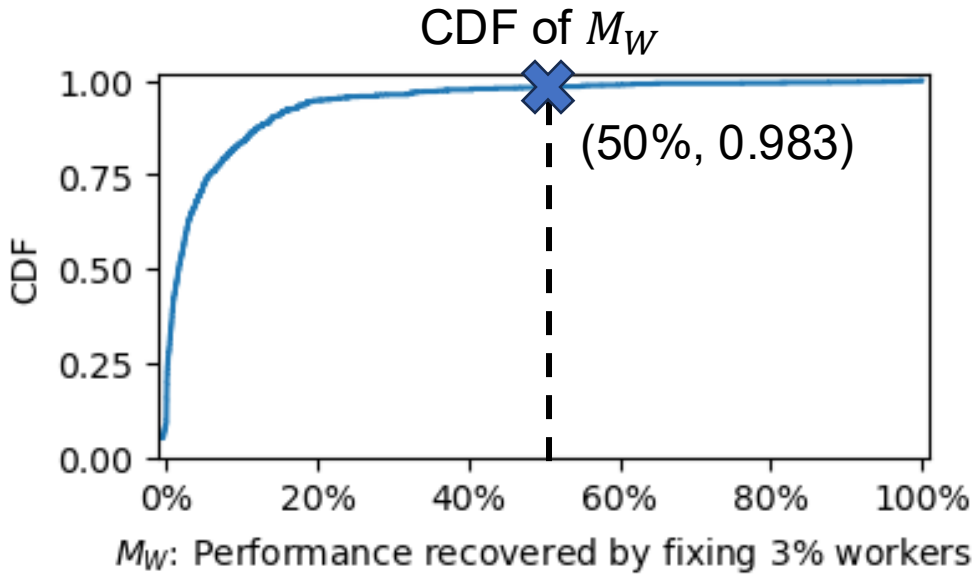
Root Cause 1: Worker Problems

- Caused by a few slow workers, e.g., due to hardware problems

- Quantify # affected jobs:
  - Idea: Fixing few workers recovers most performance
  - T_{ideal}^{W}: time had **slowest 3% workers** W been fixed
    - Simulated by **adjusting only** operations on workers W
  - How slow a worker is: *worker-slowdown* metric
  - Metric: $M_W = \frac{\text{Perf. recoverd by fixing } W:}{\text{Total perf. to recover:}} = \frac{T - T_{ideal}^{W}}{T - T_{ideal}}$ should be large

Root Cause 1: Worker Problems

- Large M_W indicates worker problems
- Finding:
  - Worker problems are not a common cause
  - But they cause higher slowdown:
    - Jobs with $M_W > 50\%$ have average slowdown 3.04
    - But all straggling jobs have average slowdown 1.28



- Implication: traditional health metrics are important but still insufficient to help avoid most stragglers

What are the actual root causes?

- Common causes:
  - PP stage partitioning imbalance due to loss layers (39.3% affected)
  - Sequence length imbalance (21.4% affected)
  - Python’s garbage collection

- Other uncommon but interesting causes:
  - GPU memory fragmentation
  - False kernel dependency

- Details in the paper

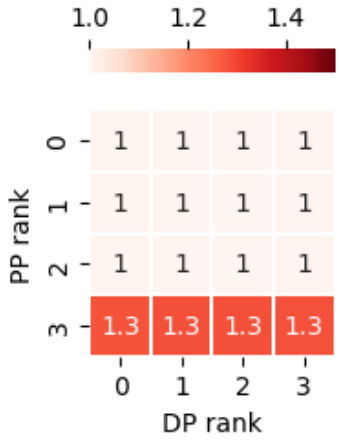
In-production Use with *SMon*

- Deployed part of the analysis at ByteDance called *SMon*

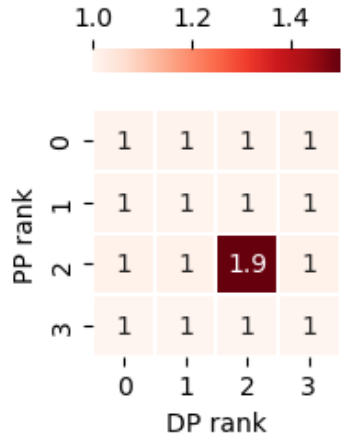
- Benefit 1: straggler monitoring
  - Automatically analyze each uploaded tracing session
    - E.g., estimation of the slowdown
  - Configure various alert triggers based on analysis report
    - E.g., alert on large slowdowns, patterns indicating worker problems

In-production Use with *SMon*

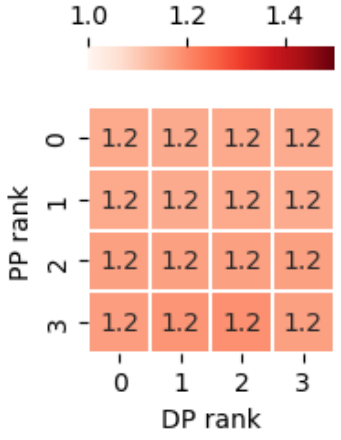
- Benefit 2: facilitates manual trouble shooting for straggling jobs



Stage partitioning imbalance



Worker problem



Sequence length imbalance

- Heatmap: cell = worker; color = worker-slowdown
- Help locate slow workers
- Identify root causes with the ***heatmap patterns***

Summary of SMon


paper


code

- Comprehensive & quantitative study of stragglers on a production trace dataset, using what-if analysis

- Studied straggler impact, symptoms and root causes
  - Stragglers are prevalent: caused 42.5% jobs to slow down >10%.
  - Worker problems not a prevalent cause
  - Common causes include: PP stage partitioning imbalance, sequence length imbalance, Python's garbage collection

- SMon helps monitor and diagnose stragglers in production