Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu¹, Manish Purohit², Zoya Svitkina², Erik Vee², Joshua R. Wang²

[1] MIT CSAIL quanquan@mit.edu
[2] Google Research {mpurohit, zoya, erikvee, joshuawang}@ncsu.edu

**Motivation**

How computationally expensive is it to perform approximately-optimal scheduling?

With growing input sizes and large data centers, highly desirable to obtain a scheduling algorithm whose running time is linear in the size of the input.

*All* current state-of-the-art algos take super-linear time.

**Problem Definition**

Classical scheduling problem with communication delay on identical machines, unit size jobs

Precedence-constrained jobs modeled as directed acyclic graph (DAG), vertex is job, edge indicates order

Given DAG $G = (V, E)$, $n$ unit-sized jobs, $M$ identical machines and communication delay $\rho$, provide a near-optimal schedule in near-linear time.

**Near-Linear Time Algorithm**

- **Previous Result:** Computing an optimal schedule is NP-hard in general. Best previous result by Lepere and Rapine obtain $O(\frac{\ln n}{\ln \rho})$ approx. in $O(mp + n \ln M)$ time
- **Our Result:** $O(n \ln M + \frac{\ln M}{\ln \rho})$ time and $O(\frac{\ln n}{\ln \rho})$ approx. algorithm, tight up to polylog factors

![Graph Problem!](image)

**Analysis Key Insights**

- **Estimating Number of Ancestor Edges:** Our count-distinct estimator algorithm on DAGs return $(1 \pm \epsilon)$-approximations to the number of ancestors (and edges) with probability at least $1 - \frac{\epsilon}{\ln \rho}$ for any constant $\epsilon \geq 1$ in $O\left(\frac{\ln^3 n}{\epsilon^2 \ln \rho}\right)$ time per vertex
- **Partitioning Vertices to Buckets:** A vertex $v$ is in the $i$-th bucket if the estimate of the number of edges in the induced subgraph of its ancestors, $\overline{e}(v)$, is in $[2^i, 2^{i+1})$; we can partition all vertices in $O\left(\frac{\ln^3 n}{\epsilon^2 \ln \rho}\right)$ time
- **Sampling Vertices from Buckets:** A sampled vertex $v$ and all its ancestors are added to $\mathcal{S}$ if its ancestors do not overlap too much with $\mathcal{S}$. Keep sampling until we’ve seen $O(\log n)$ vertices in a row that we do not add to $\mathcal{S}$ or bucket is empty
- **Pruning All State Vertices from Buckets:** Vertices with ancestor sets that overlap too much with $\mathcal{S}$ are pruned or removed from buckets
- **Standard list scheduling jobs in $\mathcal{S}$:** Duplicate all ancestors of a job, schedule a job and all its (duplicated) ancestors on the same machine

---
