Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu
Northwestern University

Manish Purohit
Google Research

Zoya Svitkina
Google Research

Erik Vee
Google Research

Joshua R. Wang
Google Research
Scheduling is a classical problem in theory and in practice

- Cluster data processing management (Google Cloud Dataflow, Spark, Hadoop, Mesos...etc.)
- Machine learning (scheduling training, e.g., Tensorflow...etc.)
Scheduling is a classical problem in theory and in practice.

Efficient Scheduling is Important in Large Data Centers

- Cluster data processing management (Google Cloud Dataflow, Spark, Hadoop, Mesos...etc.)
- Machine learning (scheduling training, e.g., Tensorflow...etc.)
Research Question and Goal

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

Rich body of literature for designing good approximation algorithms for multiprocessor scheduling.

Open: efficient algorithms with good approximations.

Even simple formulations of exact scheduling are NP-hard.

Communication Delay

Goal: Minimize the while computing good approx. schedules.
Scheduling with Communication Delay

Minimize Makespan

Runtime as Close to Linear as Possible

Machine 1

| a | b |

Machine 2

| a | c | d | g | h |

Machine M

| e | a | f |

Unit Length Jobs

b

c

d

e

f

g

h

Precedence Constraints

Duplication of jobs

Communication delay
Previous Results

• With duplication:
  • Unit jobs, fixed uniform communication delay, identical machines, duplication:
Previous Results

• **With duplication:**
  • Unit jobs, fixed uniform communication delay, identical machines, duplication:
    • $O\left(\log \rho / \log \log \rho\right)$-approximation [Lepere-Rapine, STACS ’02], assuming schedule has length at least $\rho$
Previous Results

• With duplication:
  • Unit jobs, fixed uniform communication delay, identical machines, duplication:
    • $O(\log \rho / \log \log \rho)$-approximation [Lepere-Rapine, STACS ‘02], assuming schedule has length at least $\rho$
    • $\Omega(n \ln M + n\rho^2 + m\rho)$ runtime
Previous Results

- **With duplication:**
  - Unit jobs, fixed uniform communication delay, identical machines, duplication:
    - $O(\log \rho / \log \log \rho)$-approximation [Lepere-Rapine, STACS ’02], assuming schedule has length at least $\rho$
    - $\Omega(n \ln M + n\rho^2 + m\rho)$ runtime

**Our Result:** $O(\log \rho / \log \log \rho)$-approximation, $O(n \ln M + m\ln^3 n \ln \rho / \ln \ln \rho)$ runtime, whp, assuming schedule has length at least $\rho$
Lepere-Rapine Algorithm
Lepere-Rapine Algorithm (+Modifications)

• **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors
Lepere-Rapine Algorithm (+Modifications)

- **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors
- **Phases**:
  - Schedule a small subgraph
Lepere-Rapine Algorithm (+Modifications)

• **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors

• **Phases**:  
  • Schedule a small subgraph  
  • List schedule subsets of jobs in *batches*
Lepere-Rapine Algorithm (+Modifications)

• **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors

• **Phases**:
  • Schedule a small subgraph
  • List schedule subsets of jobs in **batches**
  • Add a delay of $\rho$ to the schedule after each batch
Lepere-Rapine Algorithm (+Modifications)

• **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors

• **Phases**:
  • Schedule a small subgraph
  • List schedule subsets of jobs in batches
  • Add a delay of $\rho$ to the schedule after each batch
  • Remove batch from small subgraph
Lepere-Rapine Algorithm (+Modifications)

- **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors
- **Phases**:
  - Schedule a small subgraph
  - List schedule subsets of jobs in batches
  - Add a delay of $\rho$ to the schedule after each batch
  - Remove batch from small subgraph
Lepere-Rapine Algorithm (+Modifications)

New Small Subgraph

Remaining Schedule

Add $\rho$ delay
Lepere-Rapine Algorithm (+Modifications)

- **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors
  \[ \Omega(n\rho^2 + m\rho) \]

- **Phases**:
  - Schedule a small subgraph
  - List schedule subsets of jobs in **batches**
  - Add a delay of $\rho$ to the schedule after each batch
  - Remove batch from small subgraph
Lepere-Rapine Algorithm (+Modifications)

• **Small subgraph**: A maximal subgraph of the input where each vertex has at most $2\rho$ ancestors

• **Phases**:
  • Schedule a small subgraph
  • List schedule subsets of jobs in batches
  • Add a delay of $\rho$ to the schedule after each batch
  • Remove batch from small subgraph
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule

> ½ ancestors + vertex are not in batch
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule

at most $\frac{1}{3}$ not in batch
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule

Graham’s list scheduling, 1971
Lepere-Rapine Algorithm (+Modifications)

• Finding a batch to schedule

\[ \Omega(n\rho^2 + m\rho) \]

Graham’s list scheduling, 1971
Our Result: $O\left(\log \rho / \log \log \rho\right)$-approximation, $\tilde{O}(n + m)$ runtime, whp, assuming schedule has length at least $\rho$. 

Count-Distinct Estimator

Bucketing and Sampling

Pruning
Near-Linear Time Scheduling

• Estimating the Number of Ancestors
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02]
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] Mergeable estimator
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02]  

**Mergeable estimator**

- **Mergeable Estimator 1**
  \[(1 + \varepsilon)-\text{approx. of cardinality of } S_1\]

- **Mergeable Estimator 2**
  \[(1 + \varepsilon)-\text{approx. of cardinality of } S_2\]

**Merged Estimator**
\[(1 + \varepsilon)-\text{approx. of cardinality of } S_1 \cup S_2\]

Provided \(|S_1 \cup S_2| = n\), requires \(O(\log^2 n)\) time to perform merge.
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
  - Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] Mergeable estimator
  - Topsort graph and update estimators in every node
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] Mergeable estimator
  • Topsort graph and update estimators in every node
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
- **Count-Distinct Estimator** [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random '02] Mergeable estimator
  - Topsort graph and update estimators in every node
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] Mergeable estimator
  • Topsort graph and update estimators in every node
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] Mergeable estimator
  • Topsort graph and update estimators in every node
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] ➔ Mergeable estimator
  • Topsort graph and update estimators in every node
Near-Linear Time Scheduling

- Estimating the Number of Ancestors (+ Number of Edges)
- **Count-Distinct Estimator** [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ’02] Mergeable estimator
  - Topsort graph and update estimators in every node
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ‘02] **Mergeable estimator**
    • Topsort graph and update estimators in every node

![Graph Diagram]
Near-Linear Time Scheduling

• Estimating the Number of Ancestors (+ Number of Edges)
  • Count-Distinct Estimator [Bar-Yossef, Jayram, Kumar, Sivakumar, Trevisan Random ’02] [Mergeable estimator]
    • Topsort graph and update estimators in every node

\[(1 \pm \epsilon)\text{-approx. on number of ancestors and edges}\]

\[O((n + m) \log^2 n)\]
**Our Result:** $O(\log \rho / \log \log \rho)$-approximation, $\tilde{O}(n + m)$ runtime, whp, assuming schedule has length at least $\rho$

How does one efficiently find a batch of vertices to schedule?
Scheduling Small Subgraph

- **Partition vertices into buckets** where \( v \) is in bucket \( i \) if the number of ancestor edges is in \( [2^i, 2^{i+1}) \) (starting with \( i = 0 \), first bucket for nodes with no ancestors)

![Diagram of scheduling small subgraph]

- Nodes: \( a, b, c, d, e \)
- Edges: \( a \rightarrow b, c, e \), \( b \rightarrow d \)
- Bucket assignments: 
  - No ancestor
  - \([1, 2)\]
  - \([2, 4)\]
  - \([4, 8)\]
Scheduling Small Subgraph

- **Partition vertices into buckets** where \( v \) is in bucket \( i \) if the number of ancestor edges is in \([2^i, 2^{i+1})\) (starting with \( i = 0 \), first bucket for nodes with no ancestors)

![Diagram showing vertex partitioning and bucket allocation](image)
Sample Vertices from Buckets

$\Box > \frac{1}{2}$ nodes/edges not in $B$

$B$

$\Box$
Sample Vertices from Buckets

>1/2 nodes/edges not in $B$
Sample Vertices from Buckets

>1/2 nodes/edges not in $B$
Sample Vertices from Buckets

Samples vertices until see $\Theta(\log n)$ vertices that cannot be added to $B$. $>1/2$ nodes/edges not in $B$.
Sample Vertices from Buckets

Samples vertices until see $\Theta(\log n)$ vertices that cannot be added to $B$

>1/2 nodes/edges not in $B$

Elements in the same bucket have approx. same size!

Charge the cost of the $O(\log n)$ vertices not added to added to $B$
Sample Vertices from Buckets

Less than a constant fraction of vertices remaining can be added.

Charge the cost of the $O(\log n)$ vertices not added to $B$.

Elements in the same bucket have approx. same size!
Pruning Vertices

0 ancestor edges 1 ancestor edge 2-3 ancestor edges
Pruning Vertices

Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges

0 ancestor edges
1 ancestor edge
2-3 ancestor edges
Pruning Vertices

Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges.

Calculate ratio between new estimate and old estimate—prune if less than 1/2.
Pruning Vertices

Remove scheduled vertices from small subgraph and rerun estimate ancestors/edges

Calculate ratio between new estimate and old estimate—prune if less than 1/2

$O((n_S + m_S) \log^2 n)$
Schedule Bucket

Machine 1

Machine 2

\( B \)

\( a \) \( c \) \( e \)
Schedule Remaining Vertices

vertices: b, d

machines: Machine 1, Machine 2

sequences:
- Machine 1: b
- Machine 2: d

time slots:
- (1, 2): b
- (2, 4): d
- (4, 8): d

No ancestor: b

Decision:
- b: ✓
- d: ✗
Our Result: $O(\log \rho / \log \log \rho)$-approximation, $	ilde{O}(n + m)$ runtime, whp, assuming schedule has length at least $\rho$
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$
• Scheduling Small Subgraph Sampling Runtime:
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  • At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  • At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  • $O(\ln \rho)$ iterations of scheduling batches
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  • At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  • $O(\ln \rho)$ iterations of scheduling batches
  • $O(|E_S| \cdot \ln n \cdot \ln \rho \cdot \ln \rho)$ total cost over all iterations
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added
    vertices to an added vertex
  • At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  • $O(\ln \rho)$ iterations of scheduling batches
    • $O(|E_S| \cdot \ln n \cdot \ln \rho \cdot \ln \rho)$ total cost over all iterations

• Pruning Runtime: $O(|E_S| \cdot \ln^3 n \cdot \ln \rho)$
Runtime

• Estimate the number of ancestors/edges: $O(m \ln^2 n)$

• Scheduling Small Subgraph Sampling Runtime:
  • $O(\ln \rho)$ buckets, each charge at most $O(\ln n)$ not added vertices to an added vertex
  • At most $O(\ln n \cdot \ln \rho)$ charged to each element of a batch
  • $O(\ln \rho)$ iterations of scheduling batches
  • $O(|E_S| \cdot \ln^4 n \cdot \ln \rho)$ total cost over all iterations

• Pruning Runtime: $O(|E_S| \cdot \ln^3 n \cdot \ln \rho)$

Total: $O(m \ln^3 n \ln \rho + n \ln M)$
Conclusion

Main challenge: efficiently determining which jobs to schedule in a batch of jobs

Solution: size-estimation via sketching, sampling and pruning, and work charging argument

Open Questions:

1. Can we get a linear time algorithm?

2. Near-linear time algorithm for non-uniform machines and non-unit jobs.

3. Can we obtain a linear-time transformation for a result without duplication?