On the Approximability of Related Machine Scheduling under Arbitrary Precedence

Vaneet Aggarwal, Tian Lan, Suresh Subramaniam, and Maotong Xu

***Abstract*—Distributed computing systems often need to consider the scheduling problem involving a collection of highly dependent data-processing tasks that must work in concert to achieve mission-critical objectives. This paper considers the unrelated machine scheduling problem for minimizing weighted sum completion time under arbitrary precedence constraints and on heterogeneous machines with different processing speeds. The problem is known to be strongly NP-hard even in the single machine setting. By making use of Queyranne's constraint set and constructing a novel Linear Programming relaxation for the scheduling problem under arbitrary precedence constraints, our results in this paper advance the state of the art. We develop a $2(1 + (m-1)/D)$-approximation algorithm (and $2(1 + (m-1)/D) + 1$-approximation) for the scheduling problem with zero release time (and arbitrary release time), where m is the number of servers and D is the task-skewness product. The algorithm can be efficiently computed in polynomial time using the Ellipsoid method and achieves nearly optimal performance in practice as $D > O(m)$ when the number of tasks per job to schedule is sufficiently larger than the number of machines available. Our implementation and evaluation using a heterogeneous testbed and real-world benchmarks confirms significant improvement in weighted sum completion time for dependent computing tasks.**

I. Introduction

Next-generation computing systems such as distributed learning are becoming increasingly sophisticated and heterogeneous, often involving a collection of highly dependent data-processing tasks that must work in concert to achieve mission-critical objectives, going beyond traditional considerations like throughput or congestion. For instance, data processing frameworks like MapReduce execute tasks in multiple sequential stages. Visual Question Answering (VQA) applications [1] often perform multiple steps of reasoning and processing, each time refining the representations with contextual and question information. In general, such precedence constraints that exist in distributed computing can be formulated as a partial order among all tasks belonging to the same job, i.e., $i \preceq j$ if task i must be completed before task j starts. The problem of task scheduling with precedence constraints arises in multi-cloud environments [2], [3], [4], where the precedence constraint is important to consider for scheduling on related servers, and is the subject of this paper.

V. Aggarwal is with the School of Industrial Engineering and the School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907, email: vaneet@purdue.edu. T. Lan, S. Subramaniam, and M. Xu are with the Department of ECE, The George Washington University, DC 20052, USA, email: {tlan, suresh, htfy8927}@gwu.edu. This work was supported in part by ONR Grant N00014-20-1-2146, U.S. National Science Foundation under Grant CNS-1618335, and CISCO research gift 1155109.

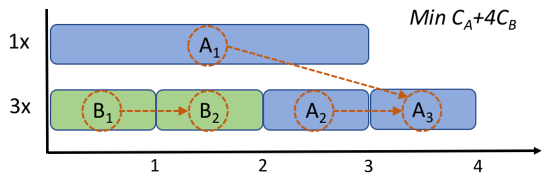


Fig. 1: Related machine scheduling under precedence constraints is NP-hard even in special cases. Algorithms that either assume identical machine speed or are oblivious of such dependence would result in substantially higher weighted completion time than the optimal solution shown here.

In this paper, we consider the scheduling problem of minimizing weighted completion time of multiple learning jobs under precedence constraints (that are modeled as an arbitrary Directed Acyclic Graph (DAG), or dependence graph) and on heterogeneous machines (with different processing speeds). Each vertex i in the dependence graph denotes a job, while each arc (i, j) represents a precedence constraint $i \preceq j$ between jobs i and j, i.e., all tasks of job i must be completed before any task of job j starts. A job is completed if all its constituent tasks are finished. An example of scheduling problem is illustrated in Figure 1, with two jobs $\{A_1, A_2, A_3\}$ and $\{B_1, B_2\}$ under precedence constraints between their constituent tasks. The weighted completion time is $w_A C_A + w_B C_B$, where C_A and C_B are the completion times of the jobs A and B, respectively. In Fig. 1, we assume $w_A = 4$ and $w_B = 1$. It is easy to see that an optimal solution minimizing weighted completion time must take precedence constraints into account, while algorithms that either assume identical machine speed or are oblivious of such dependence would result in substantially higher weighted completion time. The machines are assumed to have different speeds, e.g., speed of second machine is three times that of the first machine in Fig. 1. This scheduling problem with different machine speeds and under precedence constraints has been studied dating back to the 1950s [5], but still remains an open problem despite recent progress on approximation algorithms in a few special cases [6], [7], [8], [9], [10], [11], [12]. When there is a single job, thus giving no precedence constraints, the problem of scheduling jobs on machines with different processing speed has been studied as the *Related Machine Scheduling* problem, which is known to be strongly NP-hard even in the single machine setting, and APX-hard even when all jobs are available to schedule at time 0 (referred to as zero release time) [9]. The best known result is a 1.5-approximation algorithm for zero release times [12], and 1.8786-approximation algorithm for arbitrary arrival times [9]. Later, for the special case of identical machines and multiple jobs, and when the dependence graph reduces to a complete bipartite graph, 3-approximation and 7-approximation algorithms are proposed in [13], [14] for zero and general release times, respectively.

Our results in this paper advance the state of the art on related machine scheduling under (i) arbitrary precedence constraints (i.e., any dependence graph) and (ii) heterogeneous machine speeds, when each job consists of multiple parallel tasks. In particular, we consider the problem of minimizing weighted sum completion time $\sum_s w_s C_s$, where C_s denotes the completion time of job s, which is determined by the completion of all its constituent tasks, and w_s is a non-negative weight for job s. We develop a $2(1+(m-1)/D)$-approximation algorithm for the scheduling problem with zero release time, where m is the number of machines and D is a metric quantifying the task-skewness product, which is defined as the minimum (over all jobs in a set) ratio of the sum of task sizes (in a job) to the largest task size (in that job). Since the number of tasks to schedule is normally much larger than the number of machines available, we have $D > O(m)$, which implies that our proposed algorithm achieves an approximation ratio of $2+\epsilon$ in practice. Further, we show that the competitive ratio becomes $2(1+(m-1)/D)+1$ for general job release times, or $3+\epsilon$ when $D > O(m)$. The key idea of our approach is to make use of the Queyranne's constraint set [15] and construct a novel Linear Programming (LP) relaxation for the scheduling problem under arbitrary precedence constraints. Then, we show that the proposed LP can be efficiently computed in polynomial time using the Ellipsoid method [16]. It yields a scheduling algorithm with provable competitive ratio with respect to the optimal scheduling solution. Even when restricted to complete bipartite dependence graphs, our results significantly improve prior work, namely 37.86-approximation algorithm proposed in [17] (though their result is for a more general setting of unrelated machines, while only for zero release times), and achieves nearly optimal performance when the number of tasks to schedule is sufficiently larger than the number of machines available. We also note that 54-approximation algorithm proposed in [18] is for the case of disjoint processors in bi-partite graphs, where the Map and Reduce jobs do not happen on the same servers, and thus the setup is not directly comparable. In addition, we do not consider preemption of jobs like in [19]. The setting of related machine scheduling is studied in [20], [21], where approximation guarantee in [21] is $O(\log m/\log\log m)$. For $D > O(m \log\log m/\log m)$, we outperform the state of the art results in [21]. The results are compared in Table I, which includes a summary of different results for identical and unrelated machines.

We implement the proposed scheduling algorithm and evaluate its performance in Hadoop, whose map and reduce tasks satisfy a complete bipartite dependence graph. Modifications to the Application Master and Resource Manager are made to ensure that task/job execution follow the desired order, as given by our optimization solution. Our extensive experiments, using a combination of WordCount, Sort, and TeraSort benchmarks on a heterogeneous cluster and on real-world datasets (resulting in high level of execution time uncertainty), validate that our proposed scheduler outperforms baseline strategies including FIFO in default Hadoop, Identical-machine [13], [14], and Map-only [9], in terms of sum weighted completion time. Especially, the scheduler can achieve the smallest total weighted completion time for scheduling benchmarks with heavy workloads in reduce phase, *e.g.,* TeraSort. We also perform simulations to evaluate our scheduler under dependence graphs with multiple waves of execution. We note that for equal-sized map and equal-sized reduce jobs, D will be the number of map-reduce jobs and thus the algorithm would work in $D > O(m)$ regime.

The main contributions of the paper are as follows.

- We consider the related machine scheduling problem for minimizing weighted sum completion time, under arbitrary precedence constraints and on heterogeneous machines.
- The proposed scheduling algorithm is based on the solution of an approximated linear program, which recasts the precedence constraints and is shown to be solvable in polynomial time.
- We analyze the proposed scheduling algorithm and quantify its approximation ratio with both zero and arbitrary release times, which significantly improves prior art, especially when the number of tasks per job is large.
- Our implementation and evaluation using Hadoop shows that the scheduler outperforms other baselines by up to 82% in terms of total weighted completion time.

The rest of the paper is organized as follows. We present the system model and formulate the problem in Section II. The approximation algorithm and its analysis are provided in Section III. The implementation details of the proposed algorithm are provided in Section V, and evaluation results are presented in Section VI.

II. System Model and Problem Formulation

We consider scheduling N jobs, where the set of jobs is denoted by $\mathcal{J}$. Each job $j \in \mathcal{J}$ consists of t_j tasks, where the set of tasks of job j is denoted by $\mathcal{T}_j$. Task $t \in \mathcal{T}_j$ of job j has processing data size $p_{j,t}$. Without loss of generality, we assume that the tasks of a job are ordered in a non-increasing order of the task sizes, or

$$p_{j,1} \geq p_{j,2} \geq \cdots \geq p_{j,t_j}.$$

The objective of the problem considered in this paper is to schedule different tasks on m heterogeneous machines. Further, the scheduling is assumed to be non-preemptive, i.e., once a task is started on a machine, it cannot be stopped until it is complete. We assume that the speed of machine $i \in \{1, \cdots, m\}$ is v_i. The time taken for processing task $t \in T_j$ on server i is $p_{j,t}/v_i$. Without loss of generality, we assume that machines are ordered in non-increasing order of their speeds, or

$$v_1 \geq v_2 \geq \cdots \geq v_m.$$

We assume that the different tasks in a job have no constraints, and thus can be scheduled in parallel. The set of jobs have precedence constraints which can be represented

Scheduling problems				Performance bounds
Arbitrary machines;	Single-vertex graphs;	Arbitrary release time	[9]	1.8786
Identical machines;	Bipartite graphs;	Zero release time	[13]	3
Identical machines;	Bipartite graphs;	Arbitrary release time	[14]	7
Identical machines;	Arbitrary graphs;	Zero release time	[17]	4
Identical machines;	Arbitrary graphs;	Zero release time	[21]	$2 + 2\log 2 + \epsilon$
Unrelated machines, disjoint processors;	Bipartite graphs;	Zero release time	[18]	54
Unrelated machines;	k-partite graphs[1];	Zero release time	[17]	$(k+1+\frac{k}{\epsilon})\frac{(2\alpha^2-1)(1+\epsilon)}{\alpha-1}+\alpha+\frac{\alpha}{\epsilon}$
Related machines;	Arbitrary graphs;	Zero release time	[20]	$O(\log m)$
Related machines;	Arbitrary graphs;	Zero release time	[21]	$O(\log m/\log\log m)$
Related machines;	Arbitrary graphs;	Zero release time	[This paper]	$2(1+\frac{m-1}{D})$, or $(2+\epsilon)$ as $D > O(m)$
Related machines;	Arbitrary graphs;	Arbitrary release time	[This paper]	$1+2(1+\frac{m-1}{D})$, or $(3+\epsilon)$ as $D > O(m)$

TABLE I: Comparing performance bounds of related machine scheduling under different conditions and precedence constraints (i.e., dependence graphs), for minimizing weighted completion times $\sum_i w_i C_i$.

[1] The bound holds for $\alpha > 1$ and $\epsilon > 0$, and reduces to 37.87 for $k = 2$, i.e., bipartite graphs.

by a directed acyclic graph $G = (V, E)$, where every node represents a job. Every directed edge $(j_1, j_2) \in E$ is a constraint that job j_2 cannot start until the completion of job j_1. In other words, no task of j_2 can start before all tasks in j_1 are completed. The graph is assumed to be acyclic since there is no possible ordering of jobs that satisfies the precedence constraints if there is a cycle. Every job j has release time r_j, and has weight w_j. Without loss of generality, we assume that if $(j_1, j_2) \in E$, then $r_{j_2} \geq r_{j_1}$. This is because the start time of every task in j_2 is after the completion time of all tasks in j_1 which is at least r_{j_1}. The aim is to minimize the weighted completion time of the jobs. Let C_j^{OPT} represent the completion time of job j based on the scheduling of different tasks, such that $\sum_j w_j C_j$ is minimized, where C_j is the completion time of job j.

We introduce some notations that are employed in this paper to simplify the analysis and discussions. Let μ denote the total processing rates of m machines, i.e., $\mu = \sum_{l=1}^{m} v_l$. Further, let q_j denote the maximum number of concurrent tasks in a job, i.e., $q_j = \min\{t_j, m\}$, which can be scheduled in parallel. We define μ_j to be the sum processing speed of the fastest q_j machines, or $\mu_j = \sum_{l=1}^{q_j} v_l$. It is easy to see that μ_j is the maximum possible processing speed of job j, since its tasks can only occupy q_j distinct machines at any given time. We denote the total processing data size of all tasks of job j as p_j, i.e., $p_j = \sum_{t \in T_j} p_{j,t}$.

We define the task-skewness product of a job as the ratio of the total size of job j and the maximum task size in job j, i.e., $D_j = p_j / p_{j,1}$. This can also be seen as the number of tasks in a job times the average-to-max ratio of the task sizes of the job. Thus, if there are multiple tasks of equal size, D equals the number of tasks in the job. The task-skewness product of all jobs, denoted by D, is the minimum of D_j for all $j \in \mathcal{J}$, i.e., $D = \min_j p_j / p_{j,1}$.

III. Approximation Algorithm

In this section, we develop an algorithm, S-PC (Scheduling under Precedence Constraints), to solve the weighted completion time minimization problem on heterogeneous machines and under precedence constraints. The algorithm is based on first solving a linear optimization, referred to as the LP-Schedule problem. The solution is then used to obtain a feasible schedule for executing the tasks on the machines.

A. LP-Schedule

We formulate LP-Schedule as follows:

$$\min \sum_{j=1}^{N} w_j C_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in S} p_j C_j \geq f(S), \quad \forall S \subset \{1, \cdots, N\}; \tag{2}$$

$$C_j \geq p_j/\mu_j + r_j \qquad \forall j \in \{1, \cdots, N\}; \tag{3}$$

$$C_j \geq p_j/\mu_j + C_{j'} \qquad \forall j \in \{1, \cdots, N\}, (j', j) \in E \tag{4}$$

where

$$f(S) = \sum_{j \in S} \frac{1}{2\mu_j} (p_j)^2 + \frac{1}{2\mu} \Big(\sum_{j \in S} p_j \Big)^2.$$

In this formulation, C_j represents the completion time of job j. We note that constraint (2) is based on the Queyranne's constraint set [15], which has been used to give 2-approximation for concurrent open shop scheduling [22], [23] without precedence constraints. The extension to machines with different processing speeds is due to [11], which formulated different versions of the polyhedral constraints based on Queyranne's constraint set. It was shown in [11] that this constraint is necessary for deterministic processing times.

Constraint (3) means that the completion time of a job is at least the sum of the release time and the processing time of the job on the fastest q_j machines. Constraint (4) replaces the release time in (3) by the completion time of the jobs which have to finish prior to job j based on the precedence constraints. It is easy to see that these constraints are also necessary for the proposed problem.

Because constraints (2)-(4) are necessary for any feasible solution of the weighted completion time optimization, any optimal solution of the LP-Schedule provides a lower bound for the weighted completion time optimization. This lower bound may not be tight, and the optimal solution may not be feasible in the original formulation, since LP-Schedule does not take into account all sufficient constraints. Nevertheless, we show that a feasible schedule for executing different tasks on different machines can be obtained from the optimal solution of the LP-Schedule.

B. Complexity of Solving LP-Schedule

Note that (2) contains an exponential number of inequalities. We are still able to solve the linear programs in polynomial time with the Ellipsoid method by using a similar separation oracle as in [11].

Definition 1 (Oracle LP-Schedule). *Define the violation as*

$$V(S) = \frac{1}{2}\Big[\frac{(\sum_{j\in S} p_j)^2}{\mu} + \frac{\sum_{j\in S}(p_j)^2}{\mu_j}\Big] - \sum_{j\in S} p_j C_j.$$

Let $\{C_j\} \in \mathbb{R}^N$ be a potentially feasible solution to our LP-Schedule (1)-(4). *Let τ denote the ordering when jobs are sorted in increasing order of $C_j - p_j/(2\mu_j)$. Find the most violated constraint in* (2) *by searching over $V(S)$ for S of the form $\{\tau(1), \cdots, \tau(j)\}$. If maximal $V(S^*) > 0$, return S^* as a violated constraint for* (2). *Otherwise, check* (3)-(4) *in linear time.*

The difficulty of testing whether a set of job completion times is a feasible solution to (1)-(4) is to find the most violated constraint in (2) for any subset of jobs. The above definition only provides us some set of jobs of the form $\{\tau(1), \cdots, \tau(j)\}$ (where $j \in \{1, \cdots, N\}$) instead of all subsets of jobs. The following lemma shows that it is sufficient to guarantee that any choice of jobs does not violate (2) once every set of jobs in the form $\{\tau(1), \cdots, \tau(j)\}$ (where $j \in \{1, \cdots, N\}$) does not violate (2).

Lemma 1 (Special case of Lemma 5, [11]). *For $P(A) := \sum_{j\in A} p_j$, we have $x \in S^* \Leftrightarrow C_x - p_x/(2\mu_x) \le P(S^*)/\mu$.*

Based on this lemma, we can find S^* in $O(n\log(n))$ time (sorting $C_{\tau(j)} - p_{\tau(j)}/2\mu_{\tau(j)}$ and compute each set), which implies that Oracle LP-Schedule runs in $O(n\log(n))$ time. By the equivalence of separation and optimization, we have the following result.

Theorem 1. *LP-Schedule can be solved in polynomial time.*

C. Proposed Algorithm

The proposed algorithm for scheduling with precedence constraints on heterogeneous machines is denoted S-PC, and is described in Algorithm 1. This algorithm is based on list scheduling.

We first solve the LP-relaxation problem (1)-(4) using the release times, job weights, machine speeds, directed acyclic graph for job precedence, and the task processing times. We assume that the solution of C_j from (1)-(4) is C_j^{LP} for all $j \in \{1, \cdots, N\}$, and the jobs $\sigma(1), \cdots, \sigma(N)$ are sequenced in non-decreasing order of $C_{\sigma(j)}^{LP}$. Thus, the tasks of job $\sigma(i)$ are scheduled before that of $\sigma(j)$ for $i < j$. In order to schedule different tasks in a job, we use Algorithm 2.

Note that different tasks in a job are ordered in non-increasing order of processing data sizes. We schedule the tasks of a job in order. Each task is assigned to a machine that produces the earliest completion time, with respect to all tasks already assigned to the machine, the task t's release time, and the required processing speed of task t on the machine. The detailed procedure can be seen in Algorithm 2.

Given the above procedure, we obtain a list of an ordering of all tasks on each machine. The tasks on each machine are run in the order decided. We will insert idle times on all machines as necessary, if any job j's tasks have an earlier starting time than the completion of tasks of job i for $(i, j) \in E$ (if there is a precedence constraint), and if any jobs are "scheduled" to begin processing ahead of their release times. This procedure ensures that job precedence constraints and job release time constraints are satisfied.

Algorithm 1 Proposed Algorithm, S-PC

```
1: Find optimal $C_j$ by solving LP relaxation problem, denote
   them as $C_j^{LP}$.
2: Sort jobs in ascending order w.r.t $C_j^{LP}$. Denote the ordered
   jobs as $\sigma(1), \cdots, \sigma(N)$.
3: for $j = \sigma(1)$ to $\sigma(N)$ do
4:     Assign different tasks of job $j$ on heterogeneous ma-
       chines using Algorithm 2
5: end for
```

Algorithm 2 Scheduling single job j on heterogeneous machines.

```
1: Initialize $T_l = 0$ for all machines $l = 1, \cdots, m$
2: for tasks $t = 1, \cdots, t_j$ do
3:     $l^* = \arg\min_l \left\{ \max(T_l, r_{\sigma(j)}) + p_{\sigma(j),t}/v_{l^*} \right\}$
4:     Assign task $t$ to machine $l^*$
5:     Update $T_{l^*} \leftarrow \max(T_{l^*}, r_{\sigma(j)}) + p_{\sigma(j),t}/v_{l^*}$
6: end for
```

IV. Proof of S-PC Approximation Ratio

In this section, we will provide approximation guarantees for the proposed algorithm. We first note that since LP-Schedule has the original objective with necessary constraints, we have

$$\sum_j w_j C_j^{LP} \le \sum_j w_j C_j^{OPT}, \tag{5}$$

where C_j^{OPT} is the completion time of job j with optimal scheduling. Let $\widehat{C}_j$ be the completion time of job j using Algorithm 1, and $\widehat{C}_j^l$ be the completion time of all the tasks of job j on machine l using Algorithm 1. We now present the following key result.

Theorem 2. *If all of the jobs are released at time* 0, *the weighted completion time of jobs using S-PC algorithm is upper bounded by $2\left(1 + \frac{m-1}{D}\right)$ times the weighted completion time of jobs under optimal scheduling. Thus,*

$$\sum_j w_j \widehat{C}_j \le 2\left(1 + \frac{m-1}{D}\right) \sum_j w_j C_j^{OPT}. \tag{6}$$

For general release times, the weighted completion time of jobs using S-PC algorithm is upper bounded by $1 + 2\left(1 + \frac{m-1}{D}\right)$ times the weighted completion time of jobs under optimal scheduling. Thus,

$$\sum_j w_j \widehat{C}_j \le \left[1 + 2\left(1 + \frac{m-1}{D}\right)\right] \sum_j w_j C_j^{OPT}. \tag{7}$$

The rest of the section is focused on proving this result. We first note that due to (5), it is enough to show that for zero release times, the following holds for all k,

$$\widehat{C}_{\sigma(k)} \leq 2\left(1+\frac{m-1}{D}\right) C^{LP}_{\sigma(k)}. \tag{8}$$

This, jointly with (5), will prove the desired result. For general release times, it is similarly enough to show for all k that

$$\widehat{C}_{\sigma(k)} \leq \left[1+2\left(1+\frac{m-1}{D}\right)\right] C^{LP}_{\sigma(k)}. \tag{9}$$

We will first show the result in (8), and then show the extensions for (9). The result in (8) will be shown in three steps. The first step evaluates $\widehat{C}^{l}_{\sigma(k)}$ in terms of $\widehat{C}_{\sigma(k-1)}$. The second step extends this further to write $\widehat{C}_{\sigma(k)}$ in terms of $\widehat{C}_{\sigma(k-1)}$. The third step uses the result from the second step and performs algebraic manipulations to obtain the result.

A. Step 1 for the Proof of Theorem 2 for Zero Release Times

In the first step, we find the time needed to process job $\sigma(k)$ on machine l once job $\sigma(k-1)$ is completed on every machine. We let $\widehat{C}_{\sigma(0)} = 0$. In this subsection, we will show the following result.

Lemma 2. *The difference between the completion time of all tasks of job $\sigma(k)$ on machine l and the completion time of all tasks of job $\sigma(k-1)$ is bounded by $\frac{p_{\sigma(k)}}{\mu}(1+\frac{m-1}{D})$. More precisely, we have*

$$\widehat{C}^{l}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)} \leq \frac{p_{\sigma(k)}}{\mu}\left(1+\frac{m-1}{D}\right). \tag{10}$$

We note that $\frac{p_{\sigma(k)}}{\mu}$ gives the processing time of job $\sigma(k)$ on all machines considered together as a single machine. This result shows an additional factor loss as compared to scheduling on a single machine.

Proof. Since the tasks of job $\sigma(k)$ do not have precedence constraints among each other, every task can be started as soon as the previous ones scheduled on a machine are complete. Suppose the task of job $\sigma(k)$ that finishes the last is t^*. Let the total size of all tasks in job $\sigma(k)$ except t^* that are assigned on machine l be denoted by $p^{l}_{\sigma(k)}$. Then,

$$p_{\sigma(k)} - p_{\sigma(k),t^*} = \sum_{l=1}^{m} p^{l}_{\sigma(k)}. \tag{11}$$

The load for job $\sigma(k)$ on machine l is at most $p^{l}_{\sigma(k)} + p_{\sigma(k),t^*}$. The completion time for job $\sigma(k)$ on server l will be highest if task t^* is assigned to this machine. Since the task t^* is assigned to the machine that has the lowest completion time, the load for any machine l under the proposed algorithm is upper bounded by $(p^{l}_{\sigma(k)} + p_{\sigma(k),t^*})/v_l$. Thus we have

$$\max_{l\in\{1,\cdots,m\}}\left(\widehat{C}^{l}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)}\right) \leq \frac{p^{l}_{\sigma(k)} + p_{\sigma(k),t^*}}{v_l}, \quad \forall l. \tag{12}$$

From (12), we have

$$v_l\left(\max_{l\in\{1,\cdots,m\}}\widehat{C}^{l}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)}\right) \leq p^{l}_{\sigma(k)} + p_{\sigma(k),t^*}, \quad \forall l. \tag{13}$$

Adding this over all l, we obtain

$$\begin{aligned}
&\max_{l\in\{1,\cdots,m\}}\left(\widehat{C}^{l}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)}\right)\\
\leq\ & \frac{p_{\sigma(k)} + (m-1)p_{\sigma(k),t^*}}{\mu}\\
\leq\ & \frac{p_{\sigma(k)} + (m-1)p_{\sigma(k),1}}{\mu}\\
\leq\ & \frac{p_{\sigma(k)}}{\mu}\left(1+\frac{m-1}{D}\right),
\end{aligned} \tag{14}$$

where $p_{\sigma(k),t^*} \leq p_{\sigma(k),1}$, and $D \leq p_{\sigma(k)}/p_{\sigma(k),1} \leq p_{\sigma(k)}/p_{\sigma(k),t^*}$ by definition. □

B. Step 2 for the Proof of Theorem 2 for Zero Release Times

In the second step, we extend the result in Lemma 2 to get a bound in terms of $\widehat{C}_{\sigma(k)}$ rather than $\widehat{C}^{l}_{\sigma(k)}$, as given in the following result.

Lemma 3. *The difference between the completion times of jobs $\sigma(k)$ and $\sigma(k-1)$ using S-PC algorithm is bounded as follows.*

$$\widehat{C}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)} \leq \left(1+\frac{m-1}{D}\right)\frac{p_\sigma(k)}{\mu}. \tag{15}$$

Proof. Note that $\widehat{C}_j = \max_l \widehat{C}^{l}_{j}$. Thus,

$$\begin{aligned}
\widehat{C}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)} &= \max_{l\in\{1,\cdots,m\}}[\widehat{C}^{l}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)}]\\
&\leq \left(1+\frac{m-1}{D}\right)\frac{p_\sigma(k)}{\mu},
\end{aligned} \tag{16}$$

where the inequality follows from (10) in Lemma 2. □

C. Step 3 for the Proof of Theorem 2 for Zero Release Times

In this subsection, we will show the result in (8). We first show a bound on the job completion times for the LP-Schedule in the following lemma.

Lemma 4. *Let $\sigma(1),\cdots,\sigma(N)$ be the schedule of the jobs in S-PC algorithm. Then,*

$$\sum_{k=1}^{j} p_{\sigma(k)} \leq 2\mu\cdot C^{LP}_{\sigma(j)}. \tag{17}$$

Proof.

$$\begin{aligned}
C^{LP}_{\sigma(j)}\sum_{k=1}^{j} p_{\sigma(k)} &\geq \sum_{k=1}^{j} p_{\sigma(k)} C^{LP}_{\sigma(k)}\\
&\geq \frac{(\sum_{k=1}^{j} p_{\sigma(k)})^2}{2\mu} + \sum_{k=1}^{j}\frac{(p_{\sigma(k)})^2}{2\mu_j}\\
&\geq \frac{(\sum_{k=1}^{j} p_{\sigma(k)})^2}{2\mu},
\end{aligned}$$

where the first inequality is from $C^{LP}_{\sigma(1)} \le C^{LP}_{\sigma(2)} \le \cdots \le C^{LP}_{\sigma(N)}$, and the second inequality is from (2). Since every term is non-negative, we further have

$$C^{LP}_{\sigma(j)} \ge \frac{\sum_{k=1}^{j} p_{\sigma(k)}}{2\mu}. \tag{18}$$

□

We will now describe the steps to prove (8), which proves Theorem 2 for zero release times.

Proof of (8). From Lemma 3, we have

$$\widehat{C}_{\sigma(k)} - \widehat{C}_{\sigma(k-1)} \le \left(1 + \frac{m-1}{D}\right) \frac{p_{\sigma(k)}}{\mu}. \tag{19}$$

Summing this over k from 1 to j, we have

$$\widehat{C}_{\sigma(j)} \le \left(1 + \frac{m-1}{D}\right) \frac{\sum_{k=1}^{j} p_{\sigma(k)}}{\mu}. \tag{20}$$

Further, using Lemma 4, we have

$$\widehat{C}_{\sigma(j)} \le 2\left(1 + \frac{m-1}{D}\right) C^{LP}_{\sigma(j)}. \tag{21}$$

□

D. Extension to General Release Times

When release times are not zero, (3) indicates that for any job j, $C^{LP}_j \ge r_j$. Therefore, if the job $\sigma(k)$ starts after $r_{\sigma(k)}$ in addition to the completion of previous jobs, then

$$\widehat{C}_{\sigma(k)} \le r_{\sigma(k)} + 2\left(1 + \frac{m-1}{D}\right) C^{LP}_{\sigma(k)}. \tag{22}$$

Using $C^{LP}_j \ge r_j$, we have the result as in (9), which proves Theorem 2 for general release times.

V. Implementation

We implement our proposed scheduler and evaluate it in Hadoop, whose map and reduce tasks satisfy a complete bipartite dependence graph. Our implementation consists of three key modules: a *job scheduler* that implements Algorithm 1 to determine the scheduling order of different jobs, a *task scheduler* that is responsible for scheduling map and reduce tasks on different machines, and an *execution database* that stores statistics of previously executed jobs/tasks for estimating task completion times. A key feature of our implementation is its ability to adapt to system dynamics and task interference at runtime, which introduce additional sources of processing-time uncertainty. In particular, S-PC's task scheduler not only computes an optimal schedule of map and reduce tasks according to Algorithm 1, but also has the ability to re-optimize the schedule on the fly based on available task progress and renewed completion time estimates, as it continuously monitors progress of map/reduce tasks through collaboration with Hadoop's Application Master module and updates estimates of the remaining completion time. Further, to cope with potential desynchronization and disconnection issues in Hadoop (which may cause task execution to deviate from the optimal schedule), we also modify the Application Master (AM) and Resource Manager (RM) in Hadoop, which work collectively with task scheduler to ensure the execution (and resumption) of tasks in the desired order.

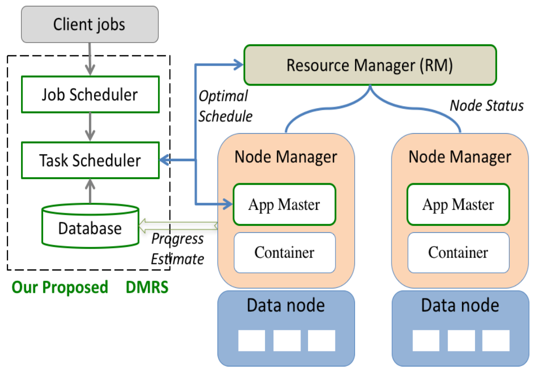


Fig. 2: System diagram of our proposed scheduler implementation.

More precisely, our scheduler works as follows. First, the job scheduler loads necessary job parameters and queries the execution database for estimated machine speeds (speed of machine l is v_l), to formulate and solve the LP-Schedule problem. The optimal job schedule is input to the task scheduler to find the schedule and placement of every map and reduce task according to Algorithm 1. Next, based on the task schedule and placement, the RM assigns a queue to each machine to store all map and reduce tasks that are scheduled to run on it. Tasks in each machine l's queue are then processed in a FIFO manner, guaranteeing the execution of jobs/tasks under our proposed algorithm. In particular, each task is given a unique ID. When resources become available on machine l, the RM launches a container and associates it with the head-of-line task. The container and task-ID pair are sent to the AM for launching the desired task.

Next, before launching each (map or reduce) task t, the task schedule estimates the completion time t_l of all jobs/tasks scheduled before t on each machine $l = 1, 2, \ldots, m$. The time t_l is obtained by combining known task completion times (which are available from execution database) and estimating the remaining times of active tasks (which are calculated by each AM using the remaining data size divided by machine speed). In particular, we continuously monitor task/job progress through AMs, refines the estimate of completion time, and if necessary, re-optimizes task schedules on the fly. By default, Hadoop reports progress scores for each task and provides estimated task execution time, derived as the time elapsed since task launching divided by the current reported progress score. In Figure. 3, we run 1000 tasks on Default Hadoop, and plot the progress score from Hadoop Reporter every 3 seconds against the (normalized) actual task completion times. It can be seen that the default progress score and actual task completion times do not follow a linear relationship, leading to high inaccurate time estimates. To mitigate this issue, we recognize that task completion time estimation error is mainly caused by Hadoop's assumption that a task starts running right after it is launched. However, due to highly contended resource-sharing environments, JVM startup time is significant, and we need to take into consideration the time to launch a JVM when estimating task progress and completion time. In particular, we calculate the time for launching JVM by finding the difference between first progress report time (t_{FP}) and launching time (t_{lau}). Therefore, the new estimated completion time is given by

$$t_{\mathrm{ect}} = t_{\mathrm{lau}} + (t_{\mathrm{FP}} - t_{\mathrm{lau}}) + (t_{\mathrm{now}} - t_{\mathrm{FP}})(CP - FP), \quad (23)$$

where $\frac{t_{\mathrm{now}} - t_{\mathrm{FP}}}{CP - FP}$ is time for processing workload, and FP and CP are first reported progress value and current reported progress value, respectively. Figure. 3 shows that our proposed estimator can eliminate the bias and provide more reliable estimation of task progress and required completion time, further boosting the performance of our scheduling algorithm.

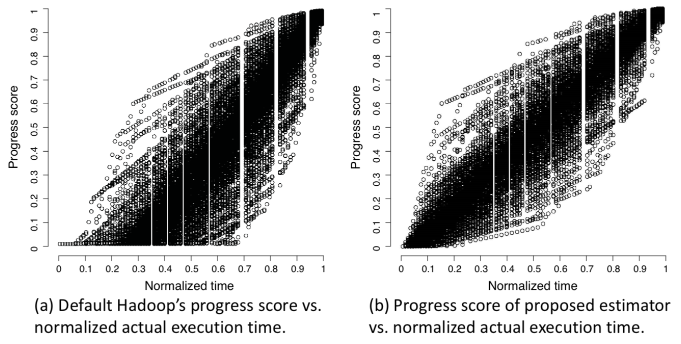


Fig. 3: The proposed estimator significantly improves progress estimates: (a) Default Hadoop's progress score and (b) Proposed estimator's progress score vs. normalized actual execution time.

Then, the optimization in Algorithm 1 is repeated at runtime to find the optimal machine l^* for task t. A new optimization of all remaining tasks by the task scheduler is triggered if l^* is different from the previous solution. This makes our S-PC schedule robust to any possible execution uncertainty and estimation errors. We also implement additional features in both AM and RM to make them fault tolerant. The container and task-ID pairs are duplicated at each AM in advance (after an optimal schedule is computed by the job and task schedulers). If RM accidentally sends an incorrect container that is intended for application (*e.g.,* due to lack of synchronization), AM will detect such inconsistency and immediately release the container back to RM. Further, a mechanism to handle occasional disconnection is implemented in both AM and RM, allowing them to buffer current containers/tasks and attempt reconnection.

VI. Evaluation

A. Evaluations of S-PC.

We evaluate our scheduler on a Hadoop cluster with three benchmarks, viz., WordCount, Sort, and TeraSort. We compare S-PC with FIFO, Identical-machine, and Map-only schedulers. FIFO is provided by Hadoop, and FIFO schedules jobs based on the jobs' releasing order. Within a job, FIFO schedules a task to the first available machine, and reduce tasks are scheduled after the map tasks of the job are completed. Identical-machine assumes all machines are identical, and applies Algorithm 1 to schedule jobs and tasks. Map-only considers the map phase is the most critical, and employs Algorithm 1 to schedule jobs and tasks without considering the reduce phase.

Experiment setup: We set up a heterogeneous cluster. The cluster contains 12 (virtual) machines, and each machine consists of a physical core and 8GB memory. Each machine can process one task at a time. In the cluster, machines are connected to a gigabit ethernet switch and the link bandwidth is 1Gbps. The heterogeneous cluster contains two types of machines, fast machines and slow machines. The processing speed ratio between a fast machine and a slow machine is 8. We evaluate our scheduler by using three benchmarks – WordCount, Sort, and TeraSort. WordCount is a CPU-bound application, and Sort is an I/O-bound application. TeraSort is CPU-bound for map phase, and I/O bound for reduce phase. We download workload for WordCount from Wikipedia, and generate workloads for Sort and TeraSort by using RandomWriter and TeraGen applications provided by Default Hadoop. The number of reduce tasks per job is set based on workload of the reduce phase. We set the number of reduce tasks per job in WordCount to be 1, and in Sort and TeraSort to be 4. All jobs are associated with weights, which are uniformly distributed between 1 and 5. Also, all jobs are partitioned into two releasing groups, and each group contains the same number of jobs. The releasing time interval between the two groups is 60sec.

Experiment results: In the first set of experiments, each experiment contains 20 jobs, and the workload of a job is 1GB. The task sizes of all jobs are the same, and equal 64MB. Figure 4(a) shows that our scheduler outperforms FIFO, Identical-machine, and Map-only by up to 62%, 68%, and 45%, respectively. Identical-machine has the largest total weighted completion time (TWCT). The reason is that it distributes the same number of tasks to each machine. A job's completion time is dominated by the tasks' completion time running on slow machines. Also, Identical-machine results in a large amount of cluster resources being wasted, since fast machines need to wait for slow machines to finish their tasks. FIFO schedules jobs based on the jobs' release order. Jobs with high weights (time-sensitive jobs) cannot be scheduled first, so time-sensitive jobs cannot be completed in time. For task scheduling, FIFO does not consider the heterogeneous cluster environment, and tasks are scheduled to the first available container. Such a task scheduling scheme can increase the completion time of tasks, since a container which becomes available later might be launched on a fast machine and be able to complete a task faster. Also, FIFO might schedule reduce tasks soon after the job is scheduled, and before the last map task is scheduled. Even though such scheme leaves more time for reduce tasks to fetch data from map tasks' outputs, given the large available network bandwidth nowadays, reduce tasks only need a little time for fetching data from all map tasks' outputs. Map-only schedules jobs and tasks without considering the reduce phase. Under benchmarks with light workloads in the reduce phase, *e.g.,* under WordCount benchmark, Map-only can achieve comparable performance as our scheduler. However, if the workload of

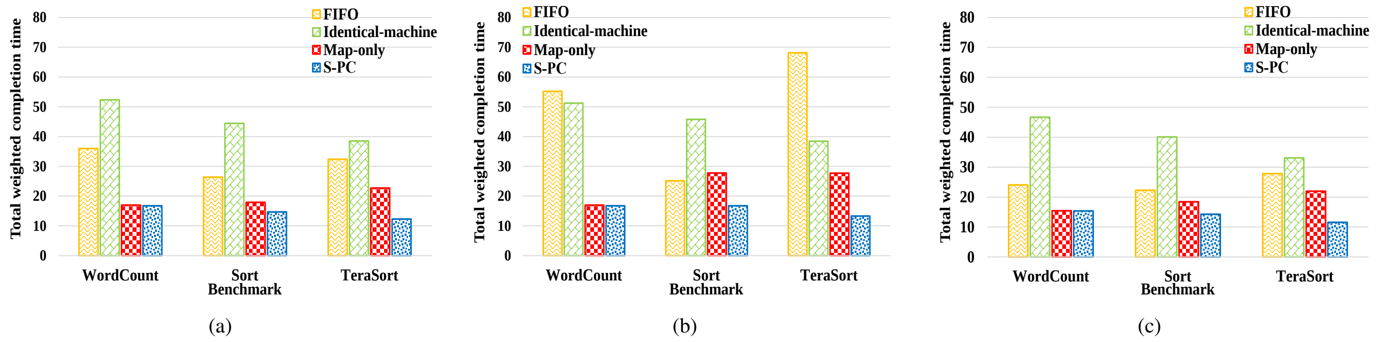


Fig. 4: (a) Total weighted completion time of jobs with identical task sizes and workloads. (b) Total weighted completion time of jobs with different task sizes and workloads. (c) Total weighted completion time of jobs with different task sizes and workloads.

the reduce phase is comparable with the map phase's, *e.g.,* under Sort, scheduling jobs without considering the workload of reduce phases and scheduling reduce tasks to random machines result in performance degradation and increase in TWCT. Furthermore, under a benchmark with heavy workload in the reduce phase, *e.g.,* TeraSort, TWCT is dominated by the completion time of reduce tasks, and Map-only increases TWCT by 85%, compared with our scheduler.

Figure 4(b) shows the results of employing jobs with different task sizes and different amount of workloads. In the set of experiments, whose results are shown in Figure 4(b), each experiment contains 20 jobs. Of these 20 jobs, 12 jobs need to process 1GB data each, and the task size is 64MB; 4 jobs need to process 0.5GB each, with a task size of 32MB; and the remaining 4 jobs need to process 2GB, with a task size of 128MB. Figure 4(b) shows that S-PC outperforms FIFO, Identical-machine, and Map-only by up to 80%, 65%, and 52%, respectively. Under WordCount and TeraSort, the TWCT of FIFO is much larger than TWCT of other schedulers in Figure 4(b) and TWCT of FIFO in Figure 4(a). This is because several jobs schedule reduce tasks soon after the jobs are scheduled; those reduce tasks occupy all fast machines, and since they cannot start to process data until all map tasks complete, all map tasks end up being scheduled on slow machines. Even though jobs' completion time of FIFO has large variation, based on our results, our scheduler can outperform FIFO by at least 36%. Also, introducing jobs with large workloads (*large jobs*) increases the TWCT of FIFO, since it does not consider jobs' and tasks' workloads in job scheduling. Jobs with small workloads (*small jobs*) might be scheduled after large jobs, and this makes small jobs suffer from the starvation problem.

We further increase the number of large jobs and small jobs, and set the number of jobs in each experiment to be 18 to make the total workload of jobs in each experiment be roughly the same as the first two sets of experiments'. Of the 18 jobs, 6 jobs need to process 1GB data each, with a task size of 64MB; another 6 jobs need to process 0.5GB each, with a task size of 32MB; and the remaining 6 jobs need to process 2GB each, with a task size of 128MB. Figure 4(c) shows that as the number of large jobs increases, our scheduler has low TWCT, since small jobs with large weights do not suffer from the starvation problem.

In the final experiment, we fix the number of TeraSort jobs to be 18, and change the number of elephant jobs. We set the task size to be 64MB. An elephant job needs to process 2GB data, and a mice job needs to process 0.5GB data. Figure 5 shows that our scheduler outperforms FIFO, Identical-machine, and Map-only by up to 82%, 66%, and 61%, respectively. As the number of elephant jobs increases, mice jobs with large weights might be scheduled after more elephant jobs, and this results in long waiting times for mice jobs, causing large increase in TWCT. Also, as the number of elephant jobs increases, the total workload increases, and the long time occupied on fast machines by reduce tasks before all map tasks finish increases TWCT greatly, since more map tasks have to process data on slow machines. By comparing TWCT of Identical-machine, Map-only, and our scheduler, we observe that as the number of elephant jobs increases, the TWCT increases for Identical-machine and Map-only are much larger than for our scheduler's. For Identical-machine, increasing the number of elephant jobs means the difference of the amount of time used to complete all assigned tasks on fast machines and on slow machines increases. Without considering the scheduling of reduce tasks, as the number of elephant jobs increases, more workloads of reduce tasks are assigned to slow machines, and this results in a large increase in TWCT.

B. Large-scale simulations

Evaluation of online re-optimization. We conduct a large-scale, trace-driven simulation (using the Google trace [24]) to evaluate S-PC's ability to re-optimize job schedules on the fly. More precisely, upon each job arrival or departure, S-PC is employed to re-optimize and update the schedules of all existing jobs in the system. For each job, we extract the arrival time, the number of associated tasks, and the workload of each task from Google trace [24]. Each job is assigned a weight uniformly distributed between 1 and 10, and its tasks are randomly partitioned into map and reduce phases (with 60% and 40% probabilities, respectively). We simulate a system with $m = 50, 100, 150$ machines, each with a different speed-up ranging from 1 to 3. Figure 6(a) shows the total weighted completion time of all jobs and compares S-PC

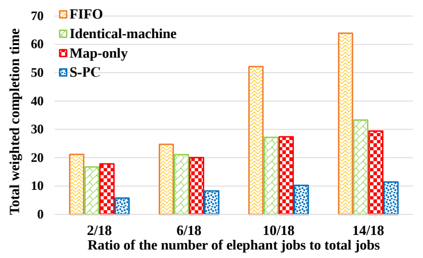
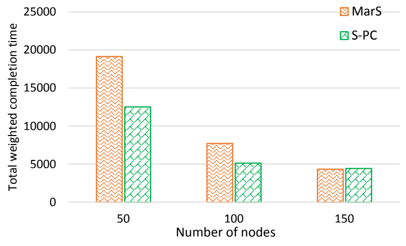
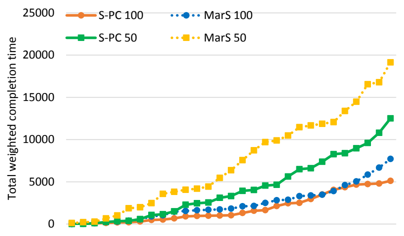


Fig. 5: Total weighted completion time under different ratios of the number of elephant jobs to total jobs..

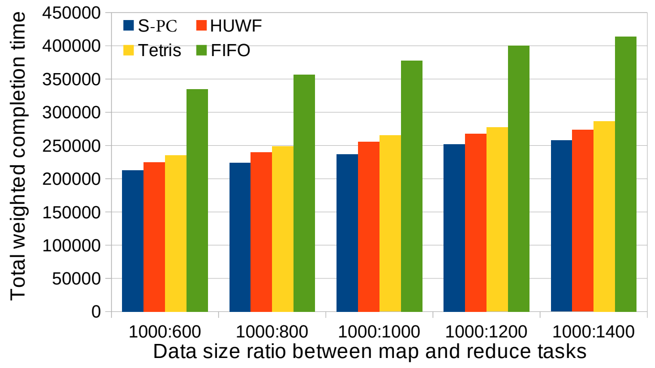


(a) Comparing weighted completion time



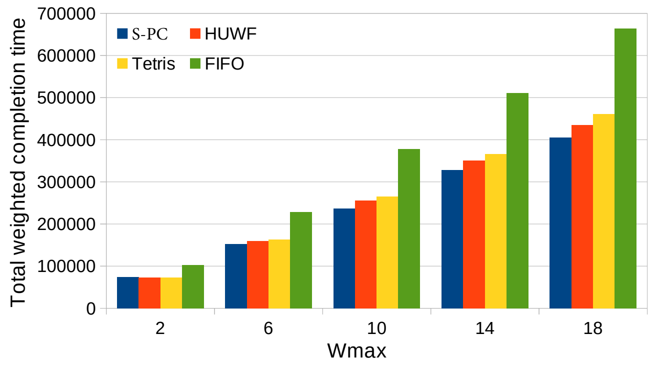
(b) Accumulative completion time over t

Fig. 6: Comparing S-PC and MarS via Google-trace simulation and online re-optimization.



	1000:600	1000:800	1000:1000	1000:1200	1000:1400
S-PC	212997	224317	237050	251986	257677
HUWF	224689	239676	255599	267546	273419
Tetris	235362	249202	265137	277193	286683
FIFO	334947	356623	377699	400461	413924

Fig. 7: Comparisons of S-PC, HUWF, Tetris, and FIFO in terms of total weighted job completion time with different data size ratios between map and reduce tasks.



	2	6	10	14	18
S-PC	74446	152420	237050	328727	405319
HUWF	73295	159819	255599	350878	434418
Tetris	73333	162568	265137	366656	461547
FIFO	103014	228727	377699	511059	664199

Fig. 8: Comparisons of S-PC, HUWF, Tetris, and FIFO in terms of total weighted job completion time with different W_{max}.

with an existing online scheduler, MarS [25], which considers joint map and reduce scheduling for the special case with identical machine speeds. We observe that S-PC outperforms MarS by up to 34%, when the system load is high (i.e., a large number of tasks to be processed by each machine as $m = 50$). This is because S-PC is able to jointly schedule map and reduce tasks of heterogeneous jobs with respect to the precedence constraints. Furthermore, the evolution of accumulative weighted completion time (over time t) is shown in Figure 6(b), for different numbers of machines. We note that as the accumulative weighted completion time of MarS grows rapidly with more jobs arriving over time, the benefit of S-PC becomes more substantial with online re-optimization.

Evaluation of scalability. In each evaluation, we simulate 100 jobs, with multiple rounds of dependent tasks, whose numbers are generated uniformly between 1 and 50. Weights of jobs are generated uniformly between 1 and W_{max}, and the default value of W_{max} is 10. Jobs are scheduled in a 100-machine cluster. Data processing speed of machines follows Gaussian distribution with mean=50 MB/sec and standard deviation=10. A machine can process one task at a time. Each data point in the following figures is the average value over 20 evaluations. Since existing scheduling algorithms fail to optimize under such dependence, we compare our scheduler with a few heuristics: High Unit Weight First (HUWF), Tetris [26], and FIFO algorithms. In HUWF, Unit Weight (UW) of a job equals job's weight divided by job execution time. Jobs are sorted based on UW in descending order, and map and reduce tasks are scheduled one by one. A task is assigned to a machine that produces the earliest completion time. In Tetris, resource usage score of a job equals the number of tasks (machines) multiplied by job execution time. Jobs are sorted based on resource usage scores, and map and reduce tasks are scheduled one by one. A task is assigned to a machine that produces the earliest completion time. Finally, FIFO sorts jobs and schedules tasks one by one to the first available machine.

Figure 7 compares S-PC with HUWF, Tetris and FIFO, in terms of total weighted completion time, by changing data size ratio between map and reduce tasks. As data size of a reduce task increases from 600 MB to 1400 MB, the total weighted job completion time of S-PC, HUWF, Tetris and FIFO increases by 44680, 48730, 51321, and 78977 (20.98%, 21.69%, 21.8%, and 23.58%), respectively. The results show that by considering precedence constraints between map and reduce tasks in job scheduling, S-PC has smaller increase in total weighted completion time as data sizes of reduce tasks increase.

Figure 8 compares our proposed scheduler with HUWF, Tetris and FIFO, in terms of TWCT, by changing maximum job weights. When job weights are small, TWCT of our scheduler, HUWF, and Tetris are almost the same. As maximum job weight increases from 2 to 18, the weighted job completion time of our scheduler, HUWF, Tetris and FIFO increases by 330873, 361123, 388214 and 561185, respectively. Because Tetris and FIFO do not consider weight in job scheduling, as job weight increases, Tetris and FIFO have larger increase in TWCT than our scheduler and HUWF. Results also show that scheduling jobs based on UW is not efficient. To achieve smaller TWCT, we also need to consider precedence between different phases and different machine processing speeds.

C. *Stochastic Task Execution Times*

We now evaluate the proposed S-PC algorithm under stochastic task execution times. Although tasks are often designated to process fixed and equal-size data splits, their processing times can still vary significantly in practice, even if machines are assumed to have deterministic speed. For example, programs like *WordCount* and *WordMean* need to iteratively store and process unique words from text files. The number of such operations (and thus processing time) required by each individual task depends heavily on the text content. Similarly, the processing times of *Sort* and *K-mean* are highly dependent on initial data distribution. Figure 9 shows the distribution of map-task processing times on real-world datasets (including Facebook [27], Twitter [28], Linux source files [29], News [30], Stack Overflow [31], and Wikipedia [32]) with a fixed split size of 128MB. Not only do we observe high processing-time uncertainty among different datasets (with a mean of 28 seconds and a standard deviation of 7 seconds, or 25% of the mean, in the aggregate distribution), but there is also substantial randomness when processing the same dataset.

We show that applying the S-PC algorithm with respect to mean task size $\bar{p}_{j,t} = \mathbb{E}[p_{j,t}]$, we can obtain a robust task schedule significantly reducing the total weighted completion time. The S-PC algorithm is evaluated against four baselines: FIFO, Identical-machine, Map-only, LATE [33] and MarS [25] schedulers, with respect to stochastic task execution times and three benchmarks, *viz.*, WordCount, WordMean, and Sort. In particular, LATE is a popular scheduler for speculatively scheduling jobs in heterogeneous environment.

Evaluation setup: We set up a heterogeneous cluster. The cluster contains 12 (virtual) machines, and each machine consists of a physical core and 8GB memory. Each machine can process one task at a time. In the cluster, machines are connected to a gigabit ethernet switch and the link bandwidth is 1Gbps. The heterogeneous cluster contains two types of machines, fast machines and slow machines. The processing speed ratio between a fast machine and a slow machine is 8. We evaluate S-PC by using three benchmarks – WordCount, WordMean, and Sort. Both WordCount and WordMean are a CPU-bound application, and Sort is an I/O-bound application.

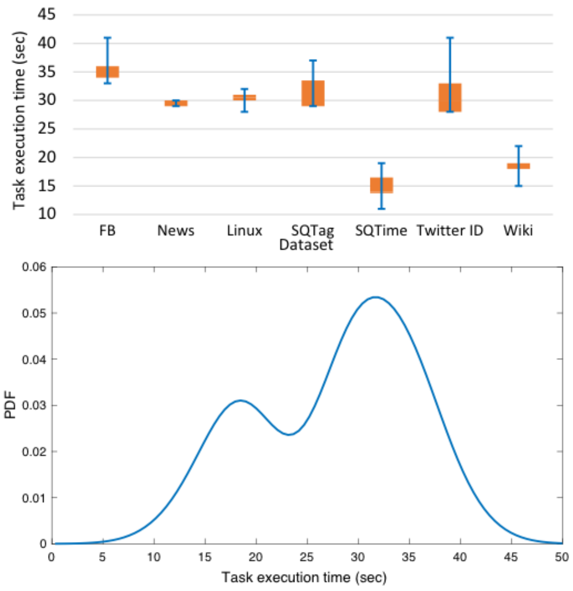


Fig. 9: Empirical distribution of map task processing time for real-world datasets (using *WordMean* and 128MB split size): (top) box-and-whisker plot for each dataset, and (bottom) aggregate *cdf* of task processing time.

Our input data is extracted from Facebook dataset. As shown in Figure 9, the difference between maximum task execution time and minimum task execution time is 8 sec, and it is 24% of the mean. The number of reduce tasks per job is set based on workload of the reduce phase. We set the number of reduce tasks per job in WordCount and WordMean to be 1, and in Sort to be 4. All jobs are associated with weights, and values of weights are uniformly distributed between 1 to 5. Also, all jobs are partitioned into two releasing groups, and each group contains the same number of jobs. The releasing time interval between two groups is 60sec. The completion time of a job is measured by the hour.

Experiment results: In the first set of experiments, each experiment contains 20 jobs, and the workload of a job is 2GB. The task sizes of all jobs are the same, and equal 64MB. Figure 10 shows that S-PC outperforms FIFO, LATE, Identical-machine, and Map-only by up to 37%, 38%, 28%, and 16% respectively. In particular, LATE launches extra copies for tasks whose progress speed is much slower than others. Even though introducing extra copies can reduce task completion time, they also consume extra cloud resources, which otherwise can be assigned to other tasks or other jobs. Furthermore, LATE records machines' processing speeds, and tries to avoid scheduling tasks on slow machines, so resources on slow machines are often under-utilized. LATE might schedule reduce tasks soon after the job is scheduled, and before the last map task is scheduled. Even though such scheme leaves more time for reduce tasks to fetch data from map tasks' outputs, reduce tasks can start to process data until all map tasks finish, and reduce tasks might occupy containers on fast machines, which can be assigned to other map tasks. Sort benchmark has heavy reduce workload, and reduce tasks in LATE waste more cluster resources. Map-only

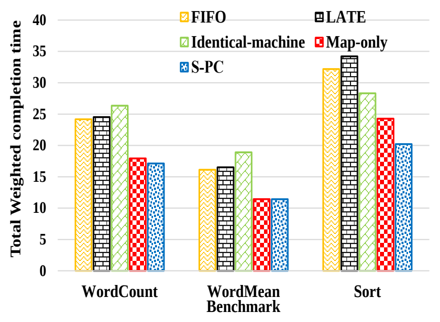


Fig. 10: Total weighted completion time of jobs with identical task sizes and workloads.

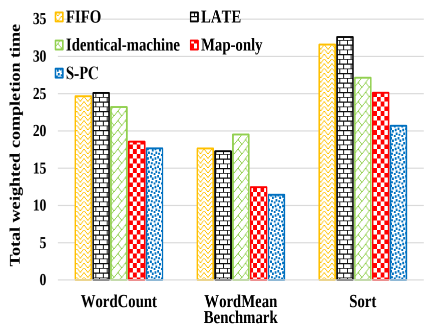


Fig. 11: Total weighted completion time with heterogeneous task sizes and workloads.

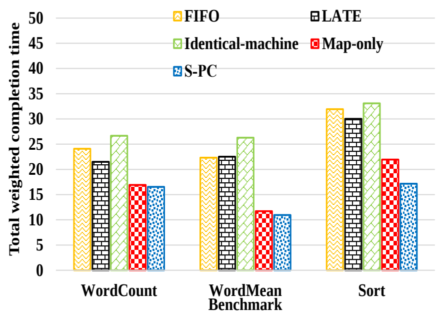


Fig. 12: Total weighted completion time with increased task heterogeneous task sizes.

schedules jobs and tasks without considering the reduce phase. Under benchmarks with light workloads in the reduce phase, *e.g.,* under WordCount and WordMean benchmarks, Map-only can achieve comparable performance as S-PC. However, under benchmarks with heavy workloads in reduce phase, *e.g.,* under Sort, scheduling reduce tasks to random machines result in performance degradation and increase total weighted completion time. S-PC schedules jobs based on their weights, average task completion time, and machine speeds to minimize total weighted completion time. Also, S-PC assigns a task to a machine, which can finish the task earliest, and such task scheduling scheme can evenly distribute workloads to all machines and fully utilize cluster resources.

Figure 11 shows the results of evaluating S-PC, in terms of total weighted completion time, by employing jobs with different task sizes and different amount of workloads. In the set of experiments, each experiment contains 20 jobs. Of these 20 jobs, 12 jobs need to process 2GB data each, and the task size is 64MB; 4 jobs need to process 1GB each, with a task size of 32MB; and the remaining 4 jobs need to process 4GB, with a task size of 128MB. Figure 11 shows that S-PC outperforms FIFO, LATE, Identical-machine, and Map-only by up to 35%, 36%, 40%, and 18%, respectively. FIFO and LATE schedule reduce tasks soon after jobs are scheduled, and those reduce tasks occupy fast machines, which can be assign to map tasks. So, S-PC outperforms FIFO and LATE by up to 35% and 36%, respectively. Also, FIFO and LATE do not try to minimize weighted completion time, jobs with small workloads and large weights might be scheduled after jobs with large workloads and small weights.

We further increase the number of large jobs and small jobs, and set the number of jobs in each experiment to be 18 to make the total workload of jobs in each experiment be roughly the same as the first two sets of experiments'. Of the 18 jobs, 6 jobs need to process 2GB data each, with a task size of 64MB; another 6 jobs need to process 1GB each, with a task size of 32MB; and the remaining 6 jobs need to process 4GB each, with a task size of 128MB. Figure 12 shows that total weighted completion time of S-PC does not grow much as the number of large jobs increases, since large jobs with small weights can be scheduled after jobs with small workloads and large weights.

Next, we fix the number of WordCount jobs, *i.e.,* 18 jobs,

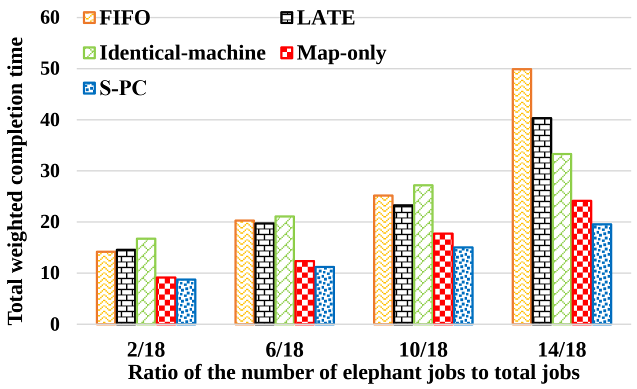


Fig. 13: Weighted completion time with different ratios of large/small jobs.

and change the number of large jobs. We set the task size to be 128MB. An large job needs to process 4GB data, and a small job needs to process 1GB data. Figure 13 shows that S-PC outperforms FIFO, LATE, Identical-machine, and Map-only by up to 60%, 50%, 37%, and 20% respectively. As the number of large jobs increases, the total weighted completion time increases. Under FIFO and LATE, as the number of large jobs increases, small jobs with large weights might be scheduled after more large jobs, and this results in long waiting times for small jobs, causing large increase in total weighted completion time. Also, as the number of large jobs increases, the increments of Identical-machine and Map-only are much larger than S-PC's. For Identical-machine, the difference of the amount of time used to complete all assigned tasks on fast machines and on slow machines increases, as the number of large jobs increases. For Map-only, as total workload in reduce phase increases, the scheduling of reduces tasks is more important for minimizing total weighted completion time. We observe that in heterogeneous environment, S-PC can significantly improve the total weighted completion time by optimally placing and scheduling all tasks with respect to their processing time and machine speeds, while simply launching extra copies for slow-running tasks (similar to LATE), or considering only map phase (similar to Map-only) are inadequate.

VII. Conclusions and Future Work

This paper considers the related machine scheduling problem for minimizing weighted sum completion time under arbitrary precedence constraints and on heterogeneous machines with different processing speeds. The precedence between any pair of tasks is modeled through a DAG, and an efficient algorithm is proposed to solve the optimization with an approximation ratio of $2(1+(m-1)/D)$ for zero release time and $2(1+(m-1)/D)+1$ for general release times. Our results significantly improve prior work - $O(\log m/\log\log m)$ in [21] (which is only for zero release times) - and achieves nearly optimal performance when the number of tasks to schedule is sufficiently larger than the number of machines available. We implement the proposed scheduling algorithm and evaluate its performance in Hadoop. The numerical results show up to 82% improvement over several baselines in terms of total weighted completion time.

We note that this paper did not account for the communication delay time when two jobs with precedence constraints are run on different machines like in [34]; such an extension is an important future direction. Further, the proposed analysis does not lead to effiicient bound for small D, and having better analytical result that achieves the bound in [21] for small D and our bound of $O(1)$ for large D is an open problem. Online algorithms akin to that in [35] are another future direction.

References

[1] R. Cadene, H. Ben-Younes, N. Thome, and M. Cord, "Murel: Multimodal Relational Reasoning for Visual Question Answering," in *IEEE Conference on Computer Vision and Pattern Recognition CVPR*, 2019. [Online]. Available: http://remicadene.com/pdfs/paper_cvpr2019.pdf

[2] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[3] ——, "Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment," *Information Systems Frontiers*, vol. 20, no. 2, pp. 373–399, 2018.

[4] M. Masdari and M. Zangakani, "Efficient task and workflow scheduling in inter-cloud environments: challenges and opportunities," *The Journal of Supercomputing*, vol. 76, no. 1, pp. 499–535, 2020.

[5] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics (NRL)*, vol. 3, no. 1-2, pp. 59–66, 1956.

[6] H. Chang, M. S. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in mapreduce-like systems for fast completion time," in *IEEE INFOCOM*, 2011, pp. 3074–3082.

[7] M. Skutella, "Convex quadratic and semidefinite programming relaxations in scheduling," *Journal of the ACM (JACM)*, vol. 48, no. 2, pp. 206–242, 2001.

[8] P. Schuurman and G. J. Woeginger, "Polynomial time approximation algorithms for machine scheduling: Ten open problems," *Journal of Scheduling*, vol. 2, no. 5, pp. 203–213, 1999.

[9] S. Im and S. Li, "Better unrelated machine scheduling for weighted completion time via random offsets from non-uniform distributions," in *IEEE (FOCS)*. IEEE, 2016, pp. 138–147.

[10] M. Sviridenko and A. Wiese, "Approximating the configuration-lp for minimizing weighted sum of completion times on unrelated machines," in *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 2013, pp. 387–398.

[11] R. Murray, S. Khuller, and M. Chao, "Scheduling distributed clusters of parallel machines: Primal-dual and lp-based approximation algorithms [full version]," *arXiv preprint arXiv:1610.09058*, 2016.

[12] N. Bansal, A. Srinivasan, and O. Svensson, "Lift-and-round to improve weighted completion time on unrelated machines," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 156–167.

[13] F. Chen, M. S. Kodialam, and T. V. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in *IEEE INFOCOM*, 2012, pp. 1143–1151.

[14] Y. Yuan, D. Wang, and J. Liu, "Joint scheduling of mapreduce jobs with servers: Performance bounds and experiments," in *IEEE INFOCOM*, 2014, pp. 2175–2183.

[15] M. Queyranne, "Structure of a simple scheduling polyhedron," *Mathematical Programming*, vol. 58, no. 1, pp. 263–285, 1993.

[16] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[17] D. Fotakis, I. Milis, O. Papadigenopoulos, V. Vassalos, and G. Zois, "Scheduling mapreduce jobs under multi-round precedences," in *European Conference on Parallel Processing*. Springer, 2016, pp. 209–222.

[18] D. Fotakis, I. Milis, O. Papadigenopoulos, E. Zampetakis, and G. Zois, "Scheduling mapreduce jobs and data shuffle on unrelated processors," in *International Symposium on Experimental Algorithms*. Springer, 2015, pp. 137–150.

[19] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós, "On scheduling in map-reduce and flow-shops," in *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, 2011, pp. 289–298.

[20] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '97. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997, pp. 581–590. [Online]. Available: http://dl.acm.org/citation.cfm?id=314161.314393

[21] S. Li, "Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations," *58th Annual IEEE Symposium on Foundations of Computer Science*, 2017.

[22] J. Y.-T. Leung, H. Li, and M. Pinedo, "Scheduling orders for multiple product types to minimize total weighted completion time," *Discrete Applied Mathematics*, vol. 155, no. 8, pp. 945–970, 2007.

[23] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Operations Research Letters*, vol. 38, no. 5, pp. 390–395, 2010.

[24] "Google trace," https://github.com/google/cluster-data, 2011.

[25] Y. Yuan, D. Wang, and J. Liu, "Joint scheduling of mapreduce jobs with servers: Performance bounds and experiments," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2175–2183.

[26] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2015.

[27] Facebook, "http://facebook.com/," 2018.

[28] Twitter, "www.twitter.com," 2018.

[29] L. kernel source tree, "https://github.com/torvalds/linux," 2018.

[30] CNN, "https://www.cnn.com/," 2018.

[31] Stackoverflow, "https://stackoverflow.com/," 2018.

[32] Wikipedia, "www.wikipedia.org," 2018.

[33] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," ser. OSDI'08, 2008.

[34] S. Davies, J. Kulkarni, T. Rothvoss, J. Tarnawski, and Y. Zhang, "Scheduling with communication delays via lp hierarchies and clustering ii: Weighted completion times on related machines," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2021, pp. 2958–2977.

[35] V. Gupta, B. Moseley, M. Uetz, and Q. Xie, "Greed works—online algorithms for unrelated machine stochastic scheduling," *Mathematics of Operations Research*, vol. 45, no. 2, pp. 497–516, 2020.

Vaneet Aggarwal (S'08 - M'11 - SM'15) received the B.Tech. degree in 2005 from the Indian Institute of Technology, Kanpur, India, and the M.A. and Ph.D. degrees in 2007 and 2010, respectively from Princeton University, Princeton, NJ, USA, all in Electrical Engineering.

He is currently an Associate Professor at Purdue University, West Lafayette, IN, where he has been since Jan 2015. He was a Senior Member of Technical Staff Research at AT&T Labs-Research, NJ (2010-2014), Adjunct Assistant Professor at Columbia University, NY (2013-2014), and VAJRA Adjunct Professor at IISc Bangalore (2018-2019). His current research interests are in communications and networking, cloud computing, and machine learning.

Dr. Aggarwal received Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the AT&T Vice President Excellence Award in 2012, the AT&T Key Contributor Award in 2013, the AT&T Senior Vice President Excellence Award in 2014, and Purdue University's Most Impactful Faculty Innovator in 2020. He also received the 2017 Jack Neubauer Memorial Award recognizing the Best Systems Paper published in the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the 2018 Infocom Workshop HotPOST Best Paper Award. He was on the Editorial Board of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING from 2017-2020. He is currently on the Editorial Board of the IEEE TRANSACTIONS ON COMMUNICATIONS, and the IEEE/ACM TRANSACTIONS ON NETWORKING.

Tian Lan received the B.A.Sc. degree from the Tsinghua University, China in 2003, the M.A.Sc. degree from the University of Toronto, Canada, in 2005, and the Ph.D. degree from the Princeton University in 2010. Dr. Lan is currently an Associate Professor of Electrical and Computer Engineering at the George Washington University. His research interests include network optimization, machine learning, and network security. Dr. Lan received the SecureComm Best Paper Award in 2019, the SEAS Faculty Recognition Award at GWU in 2018, the Hegarty Faculty Innovation Award at GWU in 2017, AT&T VURI Award in 2014, the INFOCOM Best Paper Award in 2012, the IEEE GLOBECOM Best Paper Award in 2009, and the IEEE Signal Processing Society Best Paper Award in 2008.

Suresh Subramaniam (S'95-M'97-SM'07-F'15) received the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1997. He is Professor and Chair of Electrical and Computer Engineering at the George Washington University, Washington DC, where he directs the Lab for Intelligent Networking and Computing. His research interests are in the architectural, algorithmic, and performance aspects of communication networks, with current emphasis on optical networks, cloud computing, data center networks, and IoT. He has published over 230 peer-reviewed papers in these areas.

Dr. Subramaniam is a co-editor of three books on optical networking. He has served in leadership positions for several top conferences including IEEE ComSoc's flagship conferences of ICC, Globecom, and INFOCOM. He serves/has served on the editorial boards of 7 journals including the IEEE/ACM Transactions on Networking and the IEEE/OSA Journal of Optical Communications and Networking. During 2012 and 2013, he served as the elected Chair of the IEEE Communications Society Optical Networking Technical Committee. He has received 5 Best Paper Awards, and received the 2017 SEAS Distinguished Researcher Award from George Washington University. He is an IEEE Distinguished Lecturer during 2018-2021. He is a Fellow of the IEEE.

Maotong Xu received the B.S. degree from the Northwestern Polytechnical University, China in 2012, and the M.S. and the Ph.D. degree from the George Washington University in 2014 and in 2019, respectively. Dr. Xu is currently a senior research scientist in Facebook. His interests include real-time processing system development and optimization, and cloud computing.