2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)

Scheduling Precedence-Constrained Jobs on Related Machines with Communication Delay

Biswaroop Maiti*, Rajmohan Rajaraman*, David Stalfa*, Zoya Svitkina† and Aravindan Vijayaraghavan‡
*Northeastern University, Boston, Massachusetts
m.biswaroop@gmail.com, {r.rajaraman,stalfa.d}@northeastern.edu
†Google Research, Mountain View, California
zoya@google.com
‡Northwestern University, Evanston, Illinois
aravindv@northwestern.edu

***Abstract*—We consider the problem of scheduling precedence-constrained jobs on uniformly-related machines in the presence of an arbitrary, fixed communication delay. Communication delay is the amount of time that must pass between the completion of a job on one machine and the start of any successor of that job on a different machine. We consider a model that allows job duplication, i.e. processing of the same job on multiple machines, which, as we show, can reduce the length of a schedule (i.e., its makespan) by a logarithmic factor. Our main result is an approximation algorithm for makespan with approximation ratio polylogarithmic in the number of machines and the length of the communication delay, assuming the minimum makespan is at least the delay. Our algorithm is based on rounding a linear programming relaxation for the problem, which includes carefully designed constraints capturing the interaction among communication delay, precedence requirements, varying speeds, and job duplication. To derive a schedule from a solution to the linear program, we balance the benefits of duplication in satisfying precedence constraints early against its drawbacks in increasing overall system load. Our result builds on two previous lines of work, one with communication delay but identical machines (Lepere, Rapine 2002), and the other with uniformly-related machines but no communication delay (Chudak, Shmoys 1999).**

We next show that the integrality gap of our mathematical program is polylogarithmic in the communication delay. Our gap construction employs expander graphs and exploits a property of robust expansion and its generalization to paths of longer length, which may be of independent interest. Finally, we quantify the advantage of duplication in scheduling with communication delay. We show that the best schedule without duplication can have a larger makespan than the optimal with duplication by a logarithmic factor. Nevertheless, we present a polynomial time algorithm to transform any schedule to a schedule without duplication at the cost of an increase in makespan polylogarithmic in the number of jobs and machines. Together with our makespan approximation algorithm for schedules allowing duplication, this also yields a polylogarithmic-approximation algorithm for the setting where duplication is not allowed.

***Keywords*-scheduling; approximation algorithms; linear programming; communication delay; duplication**

I. Introduction

As computational workloads get larger and more complex, it becomes necessary to distribute tasks across multiple heterogenous processors. For example, the process of training and evaluating neural network models is often distributed over diverse devices such as CPUs, GPUs, or other specialized hardware; this process, commonly referred to as *device placement* has gained significant interest [1], [2], [3]. This gives rise to a multiprocessor scheduling problem of optimizing both the assignment of tasks to processors and the order of their execution. We address this problem, taking into account several complications that such a distributed setting presents, including job dependencies, heterogeneous machine speeds, and a communication delay between them.

The jobs comprising a workload can have data dependencies between them, where the output of one job serves as the input to another. As is common in scheduling literature, we model these dependencies using a directed acyclic graph (DAG), where a directed edge uv represents that job u must be scheduled before v. However, if these two jobs are executed on different machines, additional time is needed to transfer the data from one machine to the other. We model this time as a communication delay: this delay is zero if the two jobs run on the same machine, and is equal to some value ρ if they run on different machines. Considering that the communication delay can be substantial, another aspect of the problem comes into play. Instead of a machine waiting for the result of some computation to be communicated from another machine, it may be advantageous for it to perform this computation itself, thus *duplicating* work in order to obtain the result sooner (as highlighted in early work [4]). Indeed, the technique of duplication to hide latency has been incorporated in schedulers proposed for grid computing and cloud environments [5], [6], [7], [8]. In addition, jobs may have different processing sizes and the devices may run at different speeds, representing either different types (e.g. CPU, GPU, or TPU), or differences in machine model.

Optimization problems associated with scheduling under communication delays have been studied over the last three decades, but provably good approximation bounds are few and several challenging open problems remain [9], [10], [11], [12], [13], [14], [15], [16], [4], [17], [18]. It is known that scheduling a DAG of uniform size jobs on identical machines with a communication delay is NP-hard, even when the number of machines is infinite [18], [17]. Several inapproximability results have also been derived [10], [12]. However, these results are very limited and the approximability status of scheduling under communication delay is listed as one of the top ten open problems in scheduling surveys [19], [20]. For the special case of uniform speeds and unit jobs, a logarithmic-approximation algorithm is presented in [13]. In recent work [21], a quasi-polynomial time approximation scheme is developed for the problem when the number of machines is $O(1)$, communication delays are $O(1)$, and the machines are identical. Our focus in this paper is on deriving approximation algorithms for scheduling a DAG with *non-uniform size* jobs on an *arbitrary* number of *related machines* (arbitrary speeds) and an *arbitrary communication delay*.

A. Our results and techniques

We study the problem of scheduling a DAG with n jobs of arbitrary sizes on m *related machines*, connected by a network with a fixed communication delay. In the related machines model, machine i has a speed s_i, and the time taken to complete a job v of size p_v on i is given by p_v/s_i. We represent the network communication delay as ρ times the processing time of the smallest job on the fastest machine.

Approximation algorithm for makespan. We focus on the *makespan* objective, which is defined as the time taken by a given schedule to complete the given DAG on the machines. We consider scheduling policies that allow duplication of jobs, which, as we discuss below, can reduce makespan when compared to schedules that do not allow duplication.

Theorem 1 (Makespan approximation)**.** There is a polynomial time algorithm that, given an instance of DAG scheduling with fixed communication delay, computes a schedule whose makespan is $O(\log m \log \rho / \log\log \rho)(OPT + \rho)$, where OPT is the optimal makespan for the given instance.

We thus obtain an $O(\log m \log \rho / \log\log \rho)$-approximation algorithm as long as $OPT \geq \rho$, which is a natural requirement since it takes ρ time to distribute the jobs to the machines at the start of the schedule as well as to synchronize termination at the end of the schedule. We note that the $\log m$ factor in our approximation corresponds to an upper bound on the number of geometrically separated speed groups. This entails that, for the special case of uniform speeds, our algorithm constructs a schedule with makespan upper bounded by $O(\log \rho / \log\log \rho)(OPT + \rho)$, thus extending the result of [13] to non-uniform job sizes.

A central component of our algorithm is a linear programming relaxation. A significant challenge in this regard is to capture the precedence requirement in the presence of communication delays: we would like to determine where and when to schedule individual jobs while, at the same time, adjusting the start time of each job to account for communication delays in relation to *all* its predecessors. In the full version of our paper [22], we consider several related LPs and their natural extensions and show that these approaches are inadequate for our algorithm. To overcome these challenges, we introduce a set of variables that indicate whether a job and its predecessor are scheduled within ρ time of each other, and incorporate these variables into two new sets of constraints. The first enforces the delay requirement on jobs that do not start within ρ time of each other, and the second upper bounds the total size of all predecessors that can be executed within ρ time of their successor. The addition of these constraints exponentially reduces the integrality gap of the program.

Our rounding algorithm has two components. First, we process a fractional solution to our linear relaxation to determine a tentative assignment of jobs to groups of machines, along the lines of [23]. Next, we convert the group assignment to an actual schedule. Unlike in the case of related machines with no communication delay, we cannot invoke a list scheduling type of policy. Furthermore, our algorithm needs to duplicate jobs judiciously so as to hide the communication latency and achieve the desired approximation ratio. The main challenge in this regard is that, in order to make sufficient progress on the LP solution, we must duplicate some jobs on machines much slower than their assigned machine. We overcome this obstacle by upper bounding the total size of any duplicated jobs and structuring the machines such that those with slower speed have, as a whole, higher capacity. Section III gives an overview of our makespan results and full proofs are given in [22].

Integrality gap. We next study the integrality gap of the linear program underlying our approximation algorithm, and its dependence on the communication delay ρ. Previous work of [23] on scheduling on related machines implies an integrality gap of $\Omega(\log m / \log\log m)$ for non-uniform speeds and non-uniform job sizes, but it does not consider communication delays and hence does not yield any gap in terms of ρ.

Theorem 2 (Integrality gap)**.** There is a family of instances with uniform speeds and uniform job sizes such that for any ρ that is at least some sufficiently large constant, our linear programming relaxation has a gap of at least $\Omega(\sqrt{\log \rho})$.

This integrality gap gives the first evidence that constant factor approximations may not be tractable or may be out of reach of existing techniques when the communication delay ρ is super-constant, even with uniform job sizes and identical machines. The integrality gap also extends to variants of time-indexed linear programs and, we suspect, to a wider class of mathematical programming relaxations. Given that without communication delay, the unit speed and unit job size case has an integrality gap of at most 2 by Graham's list scheduling [24], our result suggests a separation in the approximability between the variants of precedence-constrained scheduling with and without communication delays.

Our gap construction consists of a layered DAG with $L = \omega(1)$ layers, where the dependency graph between successive layers corresponds to a random graph. The main technical challenge is to argue that $\Omega(L)$ phases (a phase here corresponds to roughly ρ time units) are needed in order to schedule all the jobs for the optimal integral solution. The expansion of the random graph implies that at most $o(1)$ fraction of the jobs can be scheduled in the first phase. However, in the next phase, the jobs that were completed previously are now available on all the machines; moreover, the remaining graph (on the unscheduled jobs) in subsequent phases is *not random* any longer! To overcome this technical hurdle, we identify and exploit a property of "robust expansion" and its generalization to paths of longer length, which may be of independent interest. Section IV provides an overview of our integrality gap result.

Bounding the duplication advantage. Given the potential of duplication to effectively hide communication latency, a natural question arises: how much smaller can the makespan of a schedule with duplications be, when compared to a *no-duplication* schedule, i.e., a schedule in which each job is processed exactly once? Our final set of results formally quantifies the *duplication advantage*.

Theorem 3 (Bounding the duplication advantage)**. Upper bound:** Given any instance with n jobs, m machines, communication delay ρ, and a schedule with makespan $C^* \geq \rho$, there exists a polynomial-time computable no-duplication schedule with makespan $O(C^* \cdot \log^2 n \log m)$. **Lower bound:** There exists an instance with $n/2 = m = 2^\rho$ for which any no-duplication schedule has makespan at least $\rho/\log\rho$ times the optimal makespan.

Together with our makespan algorithm for general schedules, the algorithm of Theorem 3 yields the following corollary [22].

Corollary 3.1. There is a polynomial time algorithm that, given an instance of DAG scheduling with fixed communication delay, computes a no-duplication schedule whose makespan is $O(\text{polylog}(n, m, \rho)) \cdot OPT$, where OPT is the makespan achieved by an optimal no-duplication schedule.

Note that the approximation ratio of Corollary 3.1 holds even when the makespan of an optimal no-duplication schedule is less than ρ, since that case can be detected and solved within an $O(1)$ factor without any communication. We also note that, since both our LP rounding and the algorithm of Theorem 3 are combinatorial, our results also yield a *combinatorial* approximation algorithm for the uniform speed case, i.e. $P \mid \text{prec}, c \mid \max C_j$. Section V gives an overview of our algorithm that transforms a general schedule to a no-duplication schedule, and [22] contains the full proofs for bounding the duplication advantage.

B. Related work

Scheduling theory has a rich history and there is extensive work on scheduling jobs with precedence constraints dating back over three decades. In the following, we review scheduling work most closely related to this paper: scheduling DAGs on related machines, and scheduling DAGs under communication delays.

Scheduling DAGs on related machines. The problem of scheduling DAGs on related machines (with no communication delays) to minimize weighted completion time was first studied by Jaffe, who gave an $O(\sqrt{m})$ approximation algorithm [25]. This was significantly improved by Chudak and Shmoys who first derived an $O(\log m)$ asymptotic approximation ratio for minimizing makespan [23] and then invoked a general framework due to Hall et al [26] and Queyranne and Sviridenko [27] to convert an approximation algorithm for makespan to an approximation algorithm for weighted completion time. The Chudak-Shmoys algorithm for makespan minimization first solves an LP relaxation for the problem, and then assigns each job to a group of machines whose speeds are within a factor of two of one another. Using Graham's list scheduling [24], they then schedule the jobs within each group of machines. The $O(\log m)$ factor arises due to the number of machine groups. In subsequent work, Chekuri and Bender derived the same $O(\log m)$ approximation via a combinatorial algorithm [28]. In recent work, Shi Li improved the approximation ratio to $O(\log m/ \log\log m)$ by a more careful tradeoff between the factor lost for organizing the machines into groups and the factor lost while assigning jobs to machine groups [29].

With regard to hardness, it is known that the problem is hard to approximate to within a constant factor even for the special case of identical machines, where the particular constant depends on underlying complexity theory assumptions [30], [31], [32]. Recent work has also shown that the problem is hard to approximate to within any constant assuming the hardness of a particular optimization problem on k-partite graphs [33].

Scheduling under communication delays. As discussed above, optimization problems associated with scheduling under communication delays have been studied for three decades since the early work of [18], [4], [34], but provably good approximation bounds are few. All previous work assumes uniform machines and either uniform job sizes or special cases such as $O(1)$ machines and $O(1)$ communication delay. For instance, in the special case of unit-size jobs, identical machines, and unit communication delay, a 7/3-approximation is presented in [15], while [12] shows that it is NP-hard to approximate better than a factor of 5/4. Hardness results are also shown in [10], [17], [18]. To the best of our knowledge, our work is among the first to develop algorithms for scheduling non-uniform jobs with precedence constraints on related machines connected by an arbitrary communication network with fixed delay.

Several recent results have shed some light on these scheduling problems. The work of [21] presents a novel quasi-polynomial time approximation scheme, based on the Sherali-Adams hierarchy framework, for the problem with $O(1)$ identical machines, non-uniform job sizes, and $O(1)$ variable communication delays. In recent independent work, [35] proves an $O(\log\rho \cdot \log m)$-approximation for the problem of minimizing makespan on identical machines with fixed, arbitrary communication delay. This approach also uses a Sherali-Adams hierarchy and a clustering of the resulting semimetric.

Another line of work [36], [37] uses a more general model of communication delay which assigns to each pair of jobs an amount of data that must be transferred if they are executed on different machines, and assigns to each pair of machines a speed at which that data can be transferred. Early work by Hwang et al. [36] focus on the case of identical machines and develop an Earliest Time First heuristic for which they provide bounds on the resulting makespan. Su et al. [37] generalize this result for the case of related machines using a generalization of the Earliest Time First heuristic. However, since neither provides a true approximation, their results do not entail any results for our problem.

The natural idea of duplication to hide communication latency was first studied by Papadimitriou and Yannakakis, who proposed a 2-approximation algorithm for scheduling DAGs on an unbounded number of identical machines with a fixed communication delay [4]. Improved bounds for infinite machines have been given in [9], [11], [38], [16]. For the case of a bounded number of machines, [14], [15] give approximation algorithms under some special cases of either very small or very large communication delay or with the DAG restricted to be a tree-precedence graph. With duplication, the only provable guarantee for a bounded number of machines with an arbitrary communication delay parameter is due to Lepere and Rapine, who present an algorithm for scheduling a DAG of unit-size jobs on identical machines with communication delay of ρ units, which achieves a makespan $O((OPT + \rho)\log\rho/\log\log\rho)$ [13].

II. Problem formulation and notation

An instance of precedence constrained scheduling with fixed communication delay is a triple (G, M, ρ) where G is a directed acyclic graph, M is a set of machines, and ρ is the communication delay. In the graph $G = (V, E)$, the n nodes of V represent jobs and the edges of E represent precedence constraints. Each job v has a *size* $p_v > 0$ and for any subset $U \subseteq V$, we define $p(U) = \sum_{u \in U} p_u$. In the set of machines $M = \{1, \ldots, m\}$, each machine $i \in M$ has a speed s_i. We order the machines such that $s_1 \le s_2 \le \ldots \le s_m$. Processing a job v on a machine i takes p_v/s_i units of time. We normalize these values so that the shortest job has size 1 and the fastest machine has speed 1, in which case one time unit is defined as the time needed to process the shortest job on the fastest machine. Each job may be *duplicated*, i.e. copies of it processed on different machines. Preemption is not allowed, and at most one job can run on a machine at any given time.

m	num. of machines	n	num. of jobs
i, j	machines	v, u	jobs
s_i	machine i's speed	p_v	size of job v
ρ	comm. delay	A_v	predecessors of v

We say that u is a *predecessor* of v, denoted $u \prec v$, if there is some (non-zero length) directed path from u to v in G. We denote the set of all predecessors of v by A_v (note that $v \notin A_v$). The parameter ρ specifies the time needed to communicate the result of a job computed on one machine to a different machine. So if $u \prec v$ and v starts on machine i at time t, then there must be a copy of u that completes either on machine i by time t or on a different machine by time $t - \rho$.

We represent a schedule as a function $\sigma : V \times M \to \mathbb{R} \cup \{\infty\}$ mapping pair (v, i) to the start time of v on i, or to ∞ if v is not scheduled on i. We say that σ is a schedule of (G, M, ρ) if all jobs in G have a finite start time on some machine in M subject to the constraints listed above. The objective is to find a σ with minimum makespan, which is the maximum (finite) completion time in σ of any copy of any job. Since this objective is trivial if there is only one job or one machine, we assume $n, m \ge 2$. In the three field notation, this problem is denoted $Q|\text{duplication, prec}, c|C_{\max}$ where c indicates uniform communication delay.

III. Approximation algorithm for makespan

At a high level, our algorithm finds a fractional solution to the scheduling problem and then, through a series of refinements, constructs a final schedule for the given instance. The various components of the algorithm are highlighted in Figure 1. The first step is a standard preprocessing of the instance, in which we eliminate machines that are slower than the fastest machine by a factor of m or more, while incurring at most a constant factor increase in makespan. We refer the reader to our full paper [22] for details.

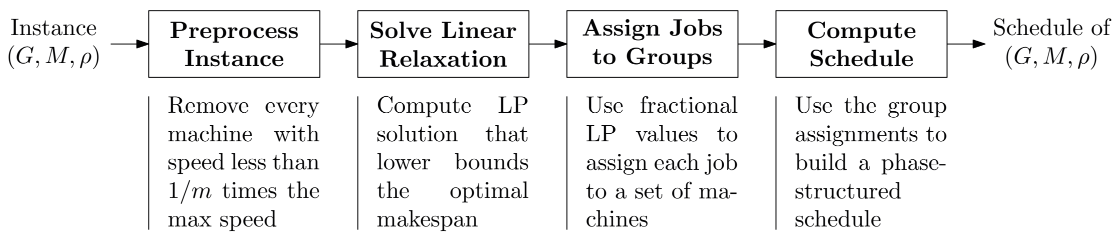
Instance (G, M, ρ)
Preprocess Instance
Solve Linear Relaxation
Assign Jobs to Groups
Compute Schedule
Schedule of (G, M, ρ)
Remove every machine with speed less than $1/m$ times the max speed
Compute LP solution that lower bounds the optimal makespan
Use fractional LP values to assign each job to a set of machines
Use the group assignments to build a phase-structured schedule

Figure 1. A high level view of our algorithm.

Approaches based on previous related work. We briefly review natural approaches to the problem of scheduling on related machines with communication delay, based on previous related work and indicate the ways in which these approaches are inadequate for our setting.

One approach is that taken in [13], which uses a combinatorial algorithm for the case with unit-speed machines, unit-size jobs, communication delay ρ, and duplication allowed. The crux of the algorithm is to repeatedly find jobs that can be completed in ρ steps and schedule them (duplicating their uncompleted predecessors, if necessary) until all such jobs have at least half their remaining predecessors already scheduled. At this point the algorithm introduces a delay on all machines, removes all previously scheduled jobs, and repeats. While this approach may work for arbitrary job sizes, accounting for variable speeds is more difficult. In [22] we show that a natural extension of this combinatorial algorithm fails.

A more effective approach is to develop a suitable LP relaxation. We consider natural variants of two relaxations developed in related work. The first captures precedence constraints and communication delays by relating job-machine assignment variables to job start and completion time variables. In this way, precedence constraints can be addressed effectively, as has been shown by [23], [29], but communication delays are much more challenging to capture. One natural approach is to add same-machine indicator variables $\delta_{u,v,i}$. Intuitively $\delta_{u,v,i} = 1$ if u and v are both scheduled on machine i, otherwise $\delta_{u,v,i} = 0$. We could then add the following constraint, where S_v and C_v represent the start and completion times of job v.

$$S_v \geq C_u + \rho\Big(1 - \sum_i \delta_{u,v,i}\Big) \qquad \forall u, v, i : u \prec v$$

We can think of the constraint as stating that any job v must begin at least ρ steps after any of its predecessors u, if v and u are not executed on the same machine. Unfortunately, a simple instance with a fractional solution that spreads each job among all the machines and sets the δ values to $1/m$ leads to an integrality gap as large as a polynomial in ρ, m, and n. See [22] for details.

A different strategy for constructing a linear relaxation is to use time-indexed job-machine assignment variables $x_{v,i,t}$ to indicate the completion time t of job v on machine i. Indeed, such a program capturing both precedence constraints and communication delays is used in [21] to obtain a quasi-polynomial time approximation scheme when the number of machines m is $O(1)$, the communication delays are $O(1)$, and all machines are identical. Unlike [21], however, we are working with an arbitrary number of machines of arbitrary speeds, and an arbitrarily large communication delay. In this case, the time-indexed relaxation has an integrality gap as large as a polynomial in ρ, m, and n. See [22].

Developing our relaxation. To overcome the challenges mentioned above, we introduce two new sets of constraints – *delay constraints* and *phase constraints* – in addition to the usual related machines scheduling constraints of [23], [29], where a *phase* is any interval of ρ time in a schedule. To build intuition, we introduce these constraints in the setting with unit speeds and unit job sizes. We then provide a natural (but weak) generalization of these constraints to the setting with arbitrary speeds and job sizes which, unfortunately, has a large integrality gap. Finally, we refine the constraints yielding our linear relaxation.

For unit speeds and unit job sizes, the phase constraints require that if a job v is scheduled to start at time t on machine i, then the total number of v's predecessors that are scheduled to start in the interval $[t-\rho, t)$ is at most ρ because they must all be scheduled on the same machine. To capture this property, we introduce *same-phase* variables $y_{u,v}$ for each pair of jobs u, v such that $u \prec v$. We can view $y_{u,v}$ as indicating whether some copy of u is scheduled within ρ steps of the start of v. We can then give the following constraints.

$$\begin{aligned} S_v &\geq S_u + \rho(1 - y_{u,v}) && \forall u, v : u \prec v \\ \rho &\geq \sum_{u \prec v} y_{u,v} && \forall v \end{aligned}$$

The first is the delay constraint and states that the difference in start times for v and u is at least ρ if some copy of u is not scheduled within ρ of the start time of v. The second is the phase constraint and states that the total number of copies of v's predecessors that are scheduled to start within ρ time of v is at most ρ. While this relaxation has a small integrality gap in the unit case, adapting it to non-unit speeds and job sizes is not straightforward.

In the case with arbitrary speeds and job sizes, we would like to capture the property analogous to the one used in the unit case: if a job v is scheduled to start at time t on machine i then the set of all v's predecessors that are scheduled to start in the interval $[t-\rho, t)$ should have total size at most ρs_i. The following relaxation uses a natural extension of the phase constraint to capture this property. It retains the same-phase variables and delay constraint of the unit relaxation, and incorporates $x_{v,i}$ variables to indicate whether job v is exectued on machine i.

$$\rho \sum_i s_i x_{v,i} \geq \sum_{u \prec v} p_u y_{u,v} \qquad \forall v$$

Unfortunately, this constraint has a flaw. If, say, a small fraction of v is placed on the fastest machine and the rest on the slowest, then the left-hand term would allow too many predecessors to be scheduled in the same phase. As shown in [22], this leads to an integrality gap as large as ρ or polynomial in m and n.

A key idea in our linear relaxation is the introduction of *machine-dependent same-phase* variables, which tie the notion of a phase to the speed of a particular machine. Using these variables, we introduce new phase and delay constraints which rely crucially on our ordering of machines by increasing speed. Our linear relaxation LP minimizes C subject to the following constraints.

$$C \geq S_v + p_v \sum_i x_{v,i}/s_i \qquad \forall v \qquad (1)$$

$$S_v \geq S_u + p_u \sum_i x_{u,i}/s_i \qquad \forall u, v : u \prec v \qquad (2)$$

$$S_v \geq S_u + \rho\big(\sum_{j \leq i} x_{v,j} - z_{u,v,i}\big) \qquad \forall u, v, i : u \prec v \qquad (3)$$

$$\sum_{j \leq i} x_{v,j} \geq \sum_{u \prec v} p_u z_{u,v,i}/\rho s_i \qquad \forall v, i \qquad (4)$$

$$C s_i \geq \sum_v p_v x_{v,i} \qquad \forall i \qquad (5)$$

$$\sum_i x_{v,i} = 1 \qquad \forall v \qquad (6)$$

$$S_v \geq 0 \qquad \forall v \qquad (7)$$

$$x_{v,i} \in (0,1) \qquad \forall v, i \qquad (8)$$

$$z_{u,v,i} \in (0,1) \qquad \forall u, v, i \qquad (9)$$

We provide some intuition behind the variables and constraints. We interpret the variables $x_{v,i}$ as giving the "primary" placement of v and S_v as the corresponding start time of v. Then, for any jobs u and v such that $u \prec v$ and for any machine i, we can understand the variable $z_{u,v,i}$ as indicating, first, whether v is executed on a machine indexed i or lower, and second, whether the start time of u is within ρ of the start time of v. The significance of this indication is that, if these conditions are met, then some *copy* of u must execute on the same machine as v within ρ time of v and, therefore, only predecessors of total size at most ρs_i can meet these conditions. We can then think of $z_{u,v,i}$ as giving a "secondary" placement of u in order to finish v as quickly as possible. The remaining variable C captures the makespan of the resulting schedule.

The delay constraint (3) states that if v is scheduled on a machine slower than i, then v should start at least ρ time after any predecessor u unless u is scheduled in the same phase as v. The phase constraint (4) states that if v is scheduled on a machine slower than i, then the total size of v's predecessors scheduled in the same phase is at most ρs_i. The remaining constraints ensure that no job completion time exceeds the makespan (1), that jobs are executed completely and in order (2, 6), and that the total load on any machine does not exceed the makespan (5).

Group assignment. The fractional solution we obtain for LP gives us a fractional assignment of jobs to machines, as well as lower bounds on start times of jobs. The objective function is the maximum over all job completion times as well as over all machine loads, and so it lower bounds the optimal makespan. The next step is to convert this solution into an assignment κ of each job to some set of machines. This assignment will guide our final construction of the schedule. We partition the set of machines into $K \leq \log m$ groups $\Gamma_1, \ldots, \Gamma_K$ of increasing speed and define a job's "median" machine group as the lowest (slowest) one such that the job's total fractional assignment to this and slower groups is at least $1/2$. Our group assignment follows an approach similar to [23], [29]: we assign each job to the highest capacity group that is at least as fast as its median group. Note that, if there are jobs assigned to groups Γ_k and $\Gamma_{k'}$, with $k < k'$, then the minimum speed in group Γ_k is less than that in group $\Gamma_{k'}$, but the capacity of Γ_k is at least that of $\Gamma_{k'}$, since the jobs assigned to Γ_k could have been assigned to group $\Gamma_{k'}$ but were not.

Computing the schedule. Our scheduling algorithm (Algorithm 1) takes the group assignment κ and produces a schedule, with possible duplications, for all jobs. The main challenge in constructing the schedule is balancing two conflicting incentives. On one hand, the more we allow a set of jobs to be duplicated, the faster we can finish any jobs preceded by jobs in this set. On the other hand, if we duplicate too often, then we risk overloading machines with too many jobs to execute. Specifically, we want to avoid scheduling too much load assigned to higher capacity groups on lower capacity (faster speed) groups, even when doing so would allow us to complete some jobs earlier. We strike this balance by allowing a job to be duplicated only in groups with capacity higher than its assigned group. Furthermore, similar to [13], when the scheduler places a set of jobs on a machine, we require that at least a $1/\eta$ fraction of the total size of that set be from jobs that have not yet been placed on any machine, where η will be set later. The algorithim also uses $\overline{s}_k$ to denote the speed of the slowest machine in group Γ_k.

Algorithm 1: Group-Based Scheduling with Duplication and Communication Delay

```
Data: instance $(G, M, \rho)$, assignment $\kappa$ of jobs to
      groups, and overlap parameter $\eta \geq 1$
Result: a schedule $\sigma$ of $G$ on $M$
Initialize: $T \leftarrow 0$;  Placed $\leftarrow \varnothing$;  $\forall j : T_j \leftarrow 0$;
  $\forall v, i : \sigma(v, i) \leftarrow \infty$
while Placed $\neq V$ do
    forall machine groups $k = 1, \dots, K$ do
        forall jobs $v : \kappa(v) = k$ do
            $i \leftarrow \arg\min_{j \in \Gamma_k} \{T_j\}$
            $A \leftarrow (A_v \cup \{v\}) \setminus \{u : \sigma(u, i) + p_u/s_i \leq T_i$
                or $\exists\ j,\ \sigma(u, j) + p_u/s_j \leq T_i - \rho\}$
            if (a) $p(A \setminus \{v\}) \leq 8\rho\overline{s}_k$ and
              (b) $p(A \setminus \text{Placed}) \geq p(A)/\eta$ and
              (c) $A \subseteq \{u : \kappa(u) \geq k\}$ then
                forall $u \in A$ in topological order do
                    $\sigma(u, i) \leftarrow T_i$
                    $T_i \leftarrow T_i + p_u/s_i$
                Placed $\leftarrow$ Placed $\cup A$
    $T \leftarrow \min\{t : t > T$ and either $t = \sigma(v, i) +$
      $p_v/s_i$ or $t = \rho + \sigma(v, i) + p_v/s_i$ for some $v, i\}$
    $\forall j :\ T_j \leftarrow \max\{T, T_j\}$
```

The scheduling algorithm proceeds in a series of rounds. In each round, the algorithm iterates through each machine group Γ_k and considers each job v with $\kappa(v) = k$ that has not yet been scheduled. On a machine $i \in \Gamma_k$ the algorithm schedules v and its predecessors that have not been completed in earlier phases if the following three conditions are satisfied: **(a)** v's incomplete predecessors can be completed on i in time $O(\rho)$; **(b)** the total size of v and its predecessors not already scheduled (on any machine) is at least a $1/\eta$ fraction of the total size of its uncompleted predecessors; and **(c)** all of v's remaining predecessors have been assigned to higher indexed groups. Condition **(c)** ensures that we duplicate jobs only from lower capacity groups to higher capacity groups. Condition **(a)** ensures that any jobs we duplicate from lower capacity, higher speed groups won't take too long on the lower speed group. Condition **(b)** ensures that the resulting schedule has two properties. First, it ensures that the total increase in load from duplication is no more than η. Second it ensures that, when large gaps are introduced in the schedule, all jobs that could have been scheduled in that gap have the size of their predecessor set reduced by a factor of η. The usefulness of these conditions is made more explicit in the analysis section.

Overview of the analysis. For the purposes of analysis, we divide our schedule into phases of length ρ and partition these phases into three types. We then bound the makespan of our schedule by bounding the total number of phases of each type. Our analysis combines elements of the analysis in [23] and [13].

The three types of phases are *chain* phases, *load* phases, and *height* phases. We define a chain of jobs $\mathscr{C}$ such that each element in $\mathscr{C}$ precedes the next, and each element has an instance which takes a sufficiently long time in the schedule. Chain phases are those phases in which some machine spends most of its time working on some chain element. All non-chain phases are divided into load and height phases. Load phases are non-chain phases in which every machine of some group is working on jobs for most of the phase. The remaining phases are height phases. We can think of the three categories more intuitively as follows. Chain phases primarily reduce the remaining execution time of the chain. Load phases primarily reduce the remaining execution time of the set of all jobs. Height phases primarily reduce the amount of time before the next chain phase (or the end of the schedule if the chain has been completed). Figure 2 depicts the relationship between the chain and the sets of jobs on which height phases make progress.

The main lemmas of this section upper bound the number of phases of each type. We present those lemmas and give overviews of their proofs below.

Lemma III.1. There are at most $O(OPT/\rho)$ chain phases.

Since chain jobs take a long time in the schedule, condition **(a)** of our algorithm ensures that every chain job is scheduled only on machines in its assigned group. Since we derived the group assignments from LP, the time spent executing jobs in the chain is at most $O(OPT)$, so the total number of chain phases is at most $O(OPT/\rho)$.

Lemma III.2. There are at most $O(OPT \cdot K\eta/\rho)$ load phases.

Condition **(c)** guarantees that the set of jobs scheduled on groups $\Gamma_k, \dots, \Gamma_K$ is a subset of the jobs assigned to these groups by κ. So, by condition **(b)**, we have that for any k, the total load on groups $\Gamma_k, \dots, \Gamma_K$ is at most an η factor above the total load assigned to those groups by κ. Using a lemma from [23], this entails that the total number of load phases is no more than $O(OPT \cdot K\eta/\rho)$.

Lemma III.3. There are at most $O(K(OPT + \rho) \cdot \log_\eta(\rho)/\rho)$ height phases.

Bounding the number of height phases is more involved as it requires a closer analysis of the linear program as well as a more detailed understanding of the step-by-step operation of the scheduling algorithm. We first partition the jobs into *bands* $B_1, B_2, \dots$ according to their start times as given by LP. We show that, for each job v in a band, the total size

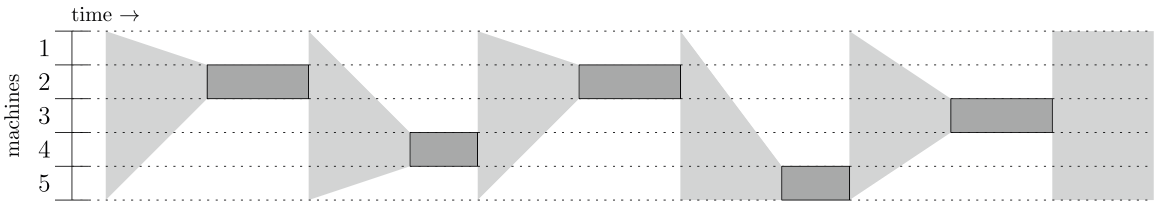

Figure 2. Machines are shown vertically on the left and time increases from left to right. The chain is shown as dark gray boxes. Each light gray, borderless area represents the set of jobs that precede the chain job to its right (if it exists) and complete after the chain job to its left (if it exists).

of v's predecessors in the same band is small enough to be completed in $O(\rho)$ time on v's assigned group. Then, for each height phase τ, we consider the lowest band B_r with some job scheduled after phase τ and the slowest group Γ_k with a job in that band. Let v be some unscheduled job in B_r assigned to group Γ_k. We consider a series of height phases separated by at most $O(1)$ height phases. We show, for each height phase in this series, that there is some iteration of our scheduling algorithm in which the algorithm *considers* placing v with its remaining predecessors on some machine in Γ_k and in which all of v's predecessors that started in the previous height phase in the series have completed with enough time to communicate the results to all machines. Due to our choice of v, we can then infer that, if the algorithm does not place v in this iteration, it is because v's uncompleted predecessor set violates condition **(b)**. This entails that by the next height phase in the series, the size of v's remaining predecessor set is reduced by a factor of η. Since v's predecessors within the band can be completed in $O(\rho)$ time on any machine in group Γ_k, we have that after $O(\log_\eta \rho)$ height phases v's predecessor set is empty. This entails that v is scheduled before (or during) the next height phase in the series. Letting r^* be the total number of bands, this argument upper bounds the number of height phases by $O(Kr^* \log_\eta \rho)$. We then show that the number of bands r^* is $O((OPT + \rho)/\rho)$, which gives the desired bound on the number of height phases.

Finally, we set η to $\log \rho / \log \log \rho$. Summing over the number of phases of each type, we have that the length of our schedule is upper-bounded by $O(K \cdot \log \rho / \log \log \rho)(OPT + \rho)$.

IV. Integrality gap

We construct a new integrality gap instance that achieves a $\omega(1)$ integrality gap in the presence of communication delays. The gap construction consists of a layered DAG with $L = \omega(1)$ layers and n vertices in each layer, where each job in layer ℓ has dependencies on d randomly chosen jobs in $V_{\ell+1}$ as shown in Figure 3. In particular, $\rho = d^L = n^c$ for a small constant $c > 0$. The parameters of the construction are set up in such a way that fractionally all the jobs can be assigned in one phase (hence the LP solution value is at most ρ).

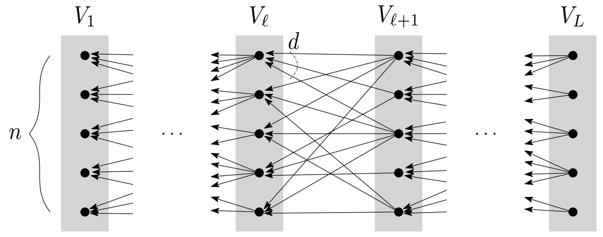

Figure 3. The figure shows the DAG with L layers $V_1, \ldots, V_L$ representing the nL jobs. Each of the n jobs in V_ℓ has dependencies on d randomly chosen jobs in $V_{\ell+1}$. We set $\rho = d^L$, $m = \rho$, and the parameters $L = c_1\sqrt{\log n}$, $d = 2^{c_2\sqrt{\log n}}$ for some appropriate constants $c_1, c_2 > 0$.

The main technical challenge is to argue that $\Omega(L)$ phases are needed to schedule all the jobs in order to get a lower bound of $\Omega(L\rho)$ for the integer solution value. This gives a gap of $\Omega(L) = \Omega(\sqrt{\log \rho})$. From the expansion of the random graph in each layer, it is easy to argue that at most a $o(1)$ fraction of the jobs in layers $\{1, \ldots, L-2\}$ can be scheduled in the first phase (since at most $\rho \ll n$ of the jobs can be on one machine). However, in the next phase the results of all jobs that were scheduled previously are now available to all the machines; moreover the choice of these jobs could depend on the randomness in the DAG. Hence the remaining graph in each layer (after removing vertices that have already been scheduled) in the subsequent phases is *not random* any longer!

To overcome this technical hurdle, we identify and exploit a property of *robust expansion*, which may be of independent interest. The standard vertex expansion property of a random graph says that w.h.p. any subset $S \subset V_\ell$ of size $|S| \leq n/d$ has a neighborhood of size $|\Gamma(S)| = \Omega(d|S|)$. However, random graphs have the stronger property that no subset T of size $o(d|S|)$ can have $\Omega(d|S|)$ of the edges from S incident on it. For our analysis, we need to prove the following generalization for paths of length $\ell < L$.

Lemma IV.1. For any $S \subset V_i$ (of sufficiently small size), there is w.h.p. no subset of size $o(d^\ell|S|)$ that can have $\Omega(d^\ell|S|)$ of the length-ℓ paths going into S.

Each job u in layer i (i.e. a vertex in V_i) has $d^{\ell-i}$ incoming paths from layer V_ℓ, and all of the vertices in these paths need to be scheduled before scheduling u – either in a previous phase, or on the same machine in the current phase. The above robust expansion property is used to upper bound the number of jobs completed in each phase in two different ways: 1) to upper bound the number of jobs in V_i whose dependencies in V_ℓ "mostly" consist of jobs scheduled in previous phases, and 2) to upper bound the number of jobs in V_i such that most of their dependencies in V_ℓ need to be resolved in the current phase. This allows us to prove that we need at least $L/2$ phases before most of the jobs in V_1 can be scheduled.

We believe that our integrality gap argument applies to a wider class of relaxations for the problem. Any program that captures communication delay and precedence requirements through individual constraints for each job and has independent load constraints for each machine is likely to incur a similar gap.

V. Bounding the duplication advantage

The final contribution of this paper is to quantitatively characterize the duplication advantage. While it is easy to construct instances where the makespan of a schedule allowing duplication (which we refer to as a general schedule) is better than that of a no-duplication schedule (one in which all jobs are processed exactly once), our goal is to place upper and lower bounds on the duplication advantage.

Lower bound. We first present a simple family of instances with m identical machines, $n = 2m + 1$ unit jobs, and $\rho = \log m$, for which any no-duplication schedule has makespan $\Omega(\rho^2/\log \rho)$, while the optimal makespan is at most ρ. The DAG for such an instance consists of a rooted binary tree with m leaves and edges directed away from the root, such that an optimal schedule executes each root-leaf path on a separate machine (with necessary duplication), while any no-duplication schedule is essentially forced to decompose the tree into $\rho/(\log \rho)$ phases, interspersed with communication delays. Note that we thus have $\Omega(\log m/ \log\log m)$ and $\Omega(\log n/\log\log n)$ bounds on the duplication advantage.

Upper bound. Our main result in this section is that the duplication advantage is, in fact, also upper-bounded by a polylogarithmic factor of $O(\log^2 n \log m)$. Our proof is through a polynomial-time algorithm that transforms any schedule σ with makespan M^* to a no-duplication schedule with the polylogarithmic factor loss in makespan.

The algorithm processes a given (general) schedule in "phases" of length ρ. Consider the t^{th} phase $[t\rho, (t+1)\rho)$ of σ for integer $t \geq 0$. Let σ_t denote the schedule σ restricted to job executions that begin in the time interval $[t\rho,(t+1)\rho)$. Let G_0 denote the subgraph of G induced by the jobs processed in σ during phase 0. For $t > 0$, let G_t denote the subgraph of G induced by jobs not in $\cup_{\ell<t} G_\ell$ whose first execution in σ begins in $[t\rho, (t+1)\rho)$. For convenience, we use $G_{<t}$ to denote $\cup_{\ell<t} G_\ell$.

The core of the algorithm is transforming each phase into a no-duplication schedule of length $O(\rho \log^2 n \log m)$. There are technical complications since (i) processing of jobs may span multiple phases of the schedule, and (ii) the number of phases may be super-polynomial in the size of the instance. Both these can be handled relatively easily by considering machines that are processing "long" jobs separately, and ignoring phases where no jobs are started or completed. We assume for now that all jobs take at most ρ time in σ. Later we show how to remove this assumption.

With this assumption, our algorithm for transforming an arbitrary schedule σ to a no-duplication schedule $\widehat{\sigma}$ consists of transforming each σ_t, $t \geq 0$, to a no-duplication schedule $\widehat{\sigma}_t$ that completes G_t in $O(\rho \log^2 n \log m)$ time. We then concatenate these schedules, inserting ρ time units before and after each $\widehat{\sigma}_t$. The extra time allows for communication of all jobs from $\widehat{\sigma}_{t-1}$ and for completion of all jobs in σ_t (assuming $p_v \leq \rho, \forall v$). Since each σ_t is of length ρ, it follows that $\widehat{\sigma}$ is of length $O(M^* \cdot \log^2 n \log m)$. From here on, we fix the phase index t and consider σ_t.

The following lemma gives, for any two jobs in G_t that share a predecessor in G_t, a lower bound on the number of machines on which both jobs are executed.

Lemma V.1. Suppose jobs u and v in G_t share a common predecessor p in G_t, and let m_u, m_v, and m_p denote the number of machines that process u, v, and p, respectively, in σ_i. Then, there exist at least $m_u + m_v - m_p$ machines that process both u and v in σ_i.

Proof: Let M_p (resp., M_u and M_v) denote the set of m_p (resp., m_u and m_v) machines processing p (resp., u and v) in σ_i. Since $M_p \supseteq M_u, M_v$, it follows that at most $m_p - m_u$ (resp., $m_p - m_v$) of the machines in M_p do not process u (resp., v). Thus, at least $m_p - (m_p - m_u) - (m_p - m_v) = m_u + m_v - m_p$ machines in M_p process both u and v, yielding the desired claim. ■

With lemma V.1, we can prove the following lemma, which entails the desired bound for the special case in which every job completes in ρ steps in σ.

Lemma V.2. There exists a polynomial-time computable no-duplication schedule that can complete G_t in $O(\rho \log^2 n \log m)$ steps.

We now give an overview of the algorithm's core. Consider the sub-DAG D of the original DAG formed by jobs processed within a particular phase of the general schedule. We face several technical challenges while designing a no-duplication schedule for this sub-DAG. First, we need to determine the relative order between the jobs. On the one hand, if a node serves as a predecessor of many jobs, it could be given higher priority. On the other hand, that same job might be duplicated several times and have successors on many different machines, something not allowed in the no-duplication schedule. Second, if we choose to process two jobs on two different machines in a phase, we have to ensure that they do not share a common predecessor.

To address these challenges, we organize and process the jobs of D as follows. First, we divide them into

$O(\log n \log m)$ groups based on their level of duplication in the general schedule; each group consists of jobs whose duplication level is within a factor of $(1+1/(2\log n))$ of one another. We then process the groups from the highest level of duplication down to the lowest, since the duplication level of a job in D is at least that of any of its successors. Within a given group, we construct an undirected graph H over the sink jobs (which have no successors in the group) in which an edge exists between two sinks if they share a common predecessor. Our key insight about H is that any subset of jobs in H composed of regions of diameter $O(\log n)$ that do not share any common neighbors among them can be processed in a single phase in a no-duplication schedule. We show that using a classic low-diameter decomposition technique from approximation algorithms and distributed computing (e.g., see [39], [40], [41]), we can find a subset of $\Omega(H)$ jobs that has the desired structure. A recursive use of this subroutine, together with the other techniques indicated above, yields the desired no-duplication schedule. We now give a more formal proof of Lemma V.2.

Proof: Recall that σ_t is a one-phase schedule for completing G_t (with possible duplication of jobs). We divide the jobs of G_t into groups based on the number of machines they are duplicated on in σ_t. For any integer $r \geq 0$, let G_{tr} be the set of jobs that are duplicated on at least $(1+\mu)^r$ machines and fewer than $(1+\mu)^{r+1}$ machines, where $\mu = 1/(2\log n)$. Since any job is duplicated on at most m machines, we obtain that the number of subgroups r^* is at most $\log_{1+\mu} m = O(\log m \log n)$.

We first argue that for any r and $r' < r$, any job in G_{tr} has no predecessors in $G_{tr'}$. Recall that any job u in G_{tr} is duplicated on at least $(1+\mu)^r$ machines and suppose v is some predecessor of u in G_t. Since there is no communication in σ_t, it follows that v must be executed on every machine where u is executed, which implies that v is also duplicated on at least $(1+\mu)^r$ machines. Therefore, $v \in G_{tr'}$ for some $r' \geq r$.

Our algorithm computes a no-duplication schedule for G_{tr}, in order from $r = r^*$ to $r = 0$. In the remainder, we show that any G_{tr} can be completed by a no-duplication schedule in $O(\log n)$ phases. Together with the bound on the number of subgroups, this yields the desired bound.

Fix integer $r \in [0, r^*]$. We show how to construct a no-duplication schedule that completes at least $1/4$ of the sinks (jobs with no successors) in G_{tr} in one phase. Repeating this at most $2\log n$ times schedules all the sinks of G_{tr}, and hence also all of G_{tr} in $2\log n$ phases (non-sink jobs are scheduled with sink jobs for which they are required).

We construct an auxiliary undirected graph H over the sinks in G_{tr} as follows: there is an edge between sink job u and sink job v if and only if u and v share a common predecessor in G_{tr}. Using a standard ball-growing technique (or the notion of sparse partitions), we determine a collection $\{S_\ell\}$ of disjoint sets of sinks in H such that (a) in every set S_ℓ, there exists a sink s_ℓ that is within $\log n$ hops of every sink in S_ℓ, (b) for any distinct ℓ, ℓ' and any two sinks $s \in S_\ell$ and $s' \in S_{\ell'}$, s is not adjacent to s'; and (c) the total number of sinks in the collection is at least $|H|/2$. Our algorithm for obtaining a collection $\{S_\ell\}$ is given in Algorithm 2. For any undirected graph K, vertex $v \in K$, and integer $x \geq 0$, let $B_x(K, v)$ denote the ball of radius x around v in K.

Algorithm 2: Find Sinks

```
1 Initialize: $H' \leftarrow H$; $\ell \leftarrow 0$
2 while $H'$ is not empty do
3     $s_\ell \leftarrow$ an arbitrary node in $H'$
4     $x \leftarrow \min\{y : |B_{y+1}(H', s_\ell)| \leq 2|B_y(H', s_\ell)|\}$
5     $S_\ell \leftarrow B_x(H', s_\ell)$
6     $H' \leftarrow H' \setminus B_{x+1}(H', s_\ell)$
```

We now argue the three properties we desire. For (a), we note that in step 4, $x \leq \log n$ since otherwise $|B_{y+1}(H', s_\ell)| > 2|B_y(H', s_\ell)|$ for $0 \leq y < \log n$, implying that $|B_{\log n}(H', s_\ell)|$ exceeds $n \geq |H'|$, a contradiction. For (b), we note that once we include a set S_ℓ, we remove all sinks in $H' \setminus S_\ell$ that are adjacent to a sink in S_ℓ, which ensures that any sink in S_ℓ is not adjacent to any sink in $S_{\ell'}$ for $\ell' > \ell$, thus establishing (b). Finally, for (c), we observe that when S_ℓ is included in the collection, we remove a set of size at most $2|S_\ell|$ from H', implying that the total number of sinks in the collection $\{S_\ell\}$ is at least $|H'|/2$, as desired.

Consider any edge (u, v) in H. By Lemma V.1, since u and v share a predecessor in G_{ir}, it follows that there exist at least $(1+\mu)^r(1-\mu)$ machines that process both u and v. Let u be any job in S_ℓ. By a repeated application of the lemma along the shortest path from s_ℓ to u, we obtain that s_ℓ and u are processed on at least $(1+\mu)^r(1-\mu)^{\log n} \geq (1+\mu)^r/2$ machines. By a standard averaging argument, it follows that there is a machine j_ℓ that processes a subset S'_ℓ of at least $|S_\ell|/2$ of the jobs in S_ℓ in σ_i.

The desired no-duplication schedule, which we denote by $\widehat{\sigma}_H$ then consists of processing S'_ℓ and all of its predecessors in G_{ir} on machine j_ℓ, for every ℓ. Since no two jobs in S_ℓ and $S_{\ell'}$ share any predecessors, it follows that no job is executed on more than one machine, hence ensuring that $\widehat{\sigma}_H$ is indeed a no-duplication schedule. Furthermore, since the jobs scheduled by $\widehat{\sigma}_H$ on a given machine j is a subset of the jobs scheduled by σ_i on j, $\widehat{\sigma}_H$ completes in a phase. Finally, since $|S'_\ell| \geq |S_\ell|/2$ and $|\cup_\ell S_\ell| \geq |H|/2$, it follows that at least $|H|/4$ of the sinks are completed in $\widehat{\sigma}_H$. We thus have obtained a no-duplication schedule that completes at least $1/4$ of the sinks in G_{ir} in one phase, thus completing the proof of the lemma. ■

For the general case where there exist jobs that take more one phase to complete, we extend the above algorithm for scheduling a given G_t as follows. For each t, we maintain a set M_t of machines on which G_t will be scheduled. Initially, M_0 is the set of all machines. When processing G_t on M_t, we first mark the jobs in G_t that begin in a phase but end at a different phase; there is at most one marked job on each machine in a given phase. We then apply the above algorithm to all the jobs in G_t. In our schedule, any marked job, if executed, would be the last job scheduled on the respective machine. We add an additional delay of ρ so that any marked jobs that complete in the following phase in σ are completed in the no-duplication schedule. If a machine works on its marked job for the next $k > 1$ phases in σ, then we remove the machine from consideration for the next k iterations (i.e., remove the machine from M_j for $t + 1 \leq j \leq t + k$) since it is not executing any jobs in G_{t+1} through G_{t+k}. This defines the set M_{t+1} of machines on which G_{t+1} will be scheduled, and we repeat the process. This completes the extension to the general case.

VI. Discussion and Open Problems

We have presented the first approximation algorithms for scheduling precedence-constrained jobs of non-uniform sizes on related machines with a fixed communication delay, with the objective of minimizing makespan. Using standard arguments, we can extend our makespan result Theorem 1 to obtain asymptotic approximations for the objective of weighted completion times [22].

Theorem 4 (Weighted completion time)**.** There is a polynomial time algorithm, that given an instance of DAG scheduling with fixed communcation delay and a weight for each job, computes a schedule with weighted completion time at most $OPT \cdot \text{polylog}(n, m) + W\rho$, where OPT is the optimal weighted completion time and W is the sum of weights of all of the jobs.

We thus obtain a true polylogarithmic-approximation for weighted completion time as long as at least a constant fraction of the jobs, by weight, take at least one communication phase to complete. Using Theorem 3, this approximation ratio result also extends to no-duplication schedules.

Our work leaves several open problems and directions for future research. Can we improve on the approximation factor achieved for general schedules? Is there a $\omega(1)$ hardness of approximation for the problem? We conjecture that the integrality gap of the relaxations is $\Omega(\log \rho / \log \log \rho)$. Improving the current bound and broadening the class of programs is of interest. Also, there is a gap between the lower and upper bounds for the duplication advantage. Narrowing this gap, and finding better approximation algorithms for no-duplication schedules would be useful for scenarios where job duplication is not a viable option.

We believe the most significant direction for future research is to study the scheduling problem under more general communication delay environments. From a practical standpoint, developing algorithms that account for delays in a hierarchical network, which may be modeled for instance by a hierarchically well-separated metric, would be valuable for many datacenter scheduling problems.

References

[1] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017, pp. 2430–2439.

[2] Y. Gao, L. Chen, and B. Li, "Optimizing device placement for training deep neural networks," in *International Conference on Machine Learning*, 2018.

[3] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "Hierarchical planning for device placement," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/pdf?id=Hkc-TeZ0W

[4] C. H. Papadimitriou and M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms," *SIAM journal on computing*, vol. 19, no. 2, pp. 322–328, 1990.

[5] D. Bozdag, F. Ozguner, and U. V. Catalyurek, "Compaction of schedules and a two-stage approach for duplication-based dag scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857–871, 2009.

[6] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Zomaya, "A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems," *Future Generation Computer Systems*, vol. 74, September 2017.

[7] D. Hu and B. Krishnamachari, "Throughput optimized scheduler for dispersed computing systems," in *2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2019, pp. 76–84.

[8] I. Song, W. Yoon, E. Jang, and S. Choi, "Task scheduling algorithm with minimal redundant duplications in homogeneous multiprocessor system," in *Grid and Distributed Computing*, T.-h. Kim, H. Adeli, H.-s. Cho, O. Gervasi, S. S. Yau, B.-H. Kang, and J. G. Villalba, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 238–245.

[9] I. Ahmad and Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 9, pp. 872–892, Sep. 1998.

[10] E. Bampis, A. Giannakos, and J.-C. König, "On the complexity of scheduling with large communication delays," *European Journal of Operational Research*, vol. 94, pp. 252–260, 1996.

[11] S. Darbha and D. P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 87–95, 1998.

[12] J. Hoogeveen, J. Lenstra, and B. Veltman, "Three, four, five, six, or the complexity of scheduling with communication delays," *Operations Research Letters*, vol. 16, no. 3, pp. 129 – 137, 1994.

[13] R. Lepere and C. Rapine, "An asymptotic $\mathcal{O}(\ln \rho / \ln \ln \rho)$-approximation algorithm for the scheduling problem with duplication on large communication delay graphs," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 2002, pp. 154–165.

[14] A. Munier, "Approximation algorithms for scheduling trees with general communication delays," *Parallel Computing*, vol. 25, no. 1, pp. 41–48, 1999.

[15] A. Munier and C. Hanen, "Using duplication for scheduling unitary tasks on m processors with unit communication delays," *Theoretical Computer Science*, vol. 178, no. 1, pp. 119 – 127, 1997.

[16] M. A. Palis, J.-C. Liou, and D. S. L. Wei, "Task clustering and scheduling for distributed memory parallel architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 1, pp. 46–55, 1996.

[17] C. Picouleau, *Two new NP-complete scheduling problems with communication delays and unlimited number of processors*. Inst. Blaise Pascal, Univ., 1991.

[18] V. J. Rayward-Smith, "Uet scheduling with unit interprocessor communication delays," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 55–71, 1987.

[19] N. Bansal, "Scheduling open problems: Old and new," 2017, mAPSP 2017. [Online]. Available: http://www.mapsp2017.ma.tum.de/MAPSP2017-Bansal.pdf

[20] 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.

[21] J. Kulkarni, S. Li, J. Tarnawski, and M. Ye, "Hierarchy-based algorithms for minimizing makespan under precedence and communication constraints," in *Proceedings of the Fortieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, to appear, 2020.

[22] B. Maiti, R. Rajaraman, D. Stalfa, Z. Svitkina, and A. Vijayaraghavan, "Scheduling precedence-constrained jobs on related machines with communication delay," 2020. [Online]. Available: https://arxiv.org/abs/2004.10776

[23] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," *Journal of Algorithms*, vol. 30, no. 2, pp. 323–343, 1999.

[24] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 17, p. 416–429, 1969.

[25] J. M. Jaffe, "Efficient scheduling of tasks without full use of processor resources," *Theoretical Computer Science*, vol. 12, no. 1, p. 1–17, Sep 1980.

[26] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of Operations Research*, vol. 22, no. 3, p. 513–544, Aug 1997.

[27] M. Queyranne and M. Sviridenko, "Approximation algorithms for shop scheduling problems with minsum objective," *Journal of Scheduling*, vol. 5, no. 4, p. 287–305, 2002.

[28] C. Chekuri and R. Motwani, "Precedence constrained scheduling to minimize sum of weighted completion times on a single machine," *Discrete Applied Mathematics*, vol. 98, no. 1-2, pp. 29–38, 1999.

[29] S. Li, "Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations," in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, 2017, pp. 283–294.

[30] J. K. Lenstra and A. H. G. R. Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, no. 1, pp. 22–35, 1978.

[31] N. Bansal and S. Khot, "Optimal long code test with one free bit," *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 453–462, Oct 2009.

[32] O. Svensson, "Conditional hardness of precedence constrained scheduling on identical machines," *Proceedings of the 42nd ACM symposium on Theory of computing - STOC '10*, p. 745–754, 2010.

[33] A. Bazzi and A. Norouzi-Fard, "Towards tight lower bounds for scheduling problems," *Lecture Notes in Computer Science*, p. 118–129, 2015.

[34] B. Veltman, B. J. Lageweg, and J. Lenstra, "Multiprocessor scheduling with com-munication delays.parallel computing," *Parallel Computing*, vol. 16, pp. 173–182, 1990.

[35] S. Davies, J. Kulkarni, T. Rothvoss, J. Tarnawski, and Y. Zhang, "Scheduling with communication delays via LP hierarchies and clustering," 2020. [Online]. Available: https://arxiv.org/abs/2004.09682

[36] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM Journal on Computing*, vol. 18, no. 2, pp. 244–257, 1989.

[37] Y. Su, X. Ren, S. Vardi, A. Wierman, and Y. He, "Communication-aware scheduling of precedence-constrained tasks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 47, pp. 21–23, 12 2019.

[38] A. Munier and J.-C. König, "A heuristic for a scheduling problem with communi-cation delays," *Operations Research*, vol. 45, no. 1, pp. 145–147, 1997.

[39] B. Awerbuch and D. Peleg, "Sparse partitions," in *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, 1990, pp. 503–513.

[40] N. Linial and M. Saks, "Low diameter graph decompositions," *Combinatorica*, vol. 13, pp. 441–454, 1993.

[41] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. Philadelphia, PA: SIAM, 2000.