Performing Partially Ordered Sets of Jobs on a MAC in Presence of Adversarial Crashes

Marek Klonowski
Faculty of Fundamental Problems of Technology
Wrocław University of Science and Technology
Wrocław, Poland
marek.klonowski@pwr.edu.pl

Dariusz R. Kowalski
School of Computer and Cyber Sciences
Augusta University
Augusta, US
SWPS University of Social Sciences and Humanities
Warsaw, Poland
dariusz.kowalski@swps.edu.pl

Jarosław Mirek
Department of Computer Science
University of Liverpool
Liverpool, UK
j.mirek@liverpool.ac.uk

Prudence W. H. Wong
Department of Computer Science
University of Liverpool
Liverpool, UK
pwong@liverpool.ac.uk

***Abstract*—We study the problem of scheduling n similar jobs on m machines, with respect to the fact that jobs are dependent and some of them must be performed before others. Dependencies between jobs are modeled as a partial order relation. Machines are prone to crashes, induced by an *Adaptive f-Bounded* adversary who can fail up to f machines, where $0 \leq f < m$. Communication takes place via a Multiple-Access Channel (MAC), which restricts simultaneous transmissions. We show an optimal solution (with respect to total work of all machines) for partially ordered sets of jobs forming chains and an algorithm and lower bound for trees.**

I. Introduction

We focus on performing n jobs on m machines reliably. This means that we expect all jobs to be done, as a result of executing an algorithm. We examine jobs that are in a partial order relation and the resulting task dependencies have to be preserved (aka precedence constraints) during the execution. Partially ordered sets of our particular interest consist of sets of independent chains of arbitrary lengths and some types of trees.

The communication medium that we analyze is a Multiple-Access Channel (also called a shared channel or a MAC for short) without collision detection. This means that simultaneous transmissions have no effect and only single broadcasts at a particular time bring legible messages heard on the channel. Furthermore, we assume that the system may suffer from adversarial crashes performed by an *Adaptive f-Bounded* adversary. Crashes take place arbitrarily with respect to the bound f with $0 \leq f < m$ i.e., at most f crashes can occur. The measure of performance of protocols is the total *work*, i.e., sum of time units of machines allocated for doing all jobs. Note that this measure also accrues *idle* time units, where a machine does not perform any job and waits for jobs to be performed by other machines. Work is a standard complexity measure used for research considerations on the Do-All problem in various models [11].

First, we show that the simplest case of ordered jobs forming independent chains can be effectively solved by adopting techniques used for scheduling arbitrary length jobs on a Multiple-Access Channel, described in [14]. Then, we consider more complex types of orders. In particular, we investigate different types of trees-shaped orders.

A. Previous and related results

The topic in this paper lies on an intersection of two different streams of research and combines a classic scheduling problem of performing jobs with the problem of performing jobs considered in distributed computing, where machines cooperate in order to perform all the jobs, communicating with each other via a medium with limited communication capacities and may often suffer congestion.

Scheduling with precedence constraints was studied on different objectives for decades [15], [16], [18]. The study started on a single machine [2], [15], [16], [18], [19], [24] and then extended to multiple machines [12], [17], [21], [23]. More complex settings were studied, under various assumptions, e.g., preemption has been considered in [17], [21]. Recently, precedence constraints were also studied on machines with different speeds by Chudak and Shmoy [7] and variable speeds [1], [22]. More details on this line of research can be found in [20].

On the other hand, Chlebus et al. [6] were the first who considered performing jobs on a Multiple-Access Channel. Their results originated from the seminal work by Dwork et al. [9], who analyzed the problem of performing jobs in the context of a message-passing model with processor crashes. The problem was studied in a number of follow-up papers [3]–[5], [8], [10] in the context of a message-passing model, in

This work is supported by the Polish National Science Center (NCN) grant UMO-2017/25/B/ST6/02553, and by Networks Sciences & Technologies (NeST), School of EEECS, University of Liverpool.

978-1-7281-2522-0/19/$31.00 ©2019 IEEE

which every node can send a message to any subset of nodes in one round.

Considerations about performing jobs on a MAC originated by Chlebus et al. [6] were continued in [13] in the context of randomized solutions for performing jobs on a Multiple Access Channel under ordered and delayed adversaries. Work by Klonowski et al. [14] analyzed the problem of performing arbitrary length jobs with preemption on a MAC. To best of our knowledge this is the first work to combine scheduling jobs with constraints and communication on a shared channel.

II. Technical preliminaries

We focus on performing n jobs on m machines communicating via a shared channel. In this section we describe the model for the considered problem.

Machines: We assume that there are m machines, with unique identifiers from the set $\{1, \ldots, m\}$. They are synchronized with a global clock, and time is divided into synchronous slots, called *rounds*. All the machines start simultaneously at a certain moment and each machine may halt voluntarily. A machine that halted is considered to be non-faulty. The complexity measure that we use in our considerations is *work* i.e. the total number of available machines steps.

Communication: Machines communicate over a Multiple Access Channel (MAC), where individually transmitted messages reach every operational machine. Thus, a broadcast is successfully received by all stations if exactly one machine transmits at a particular round. Simultaneous transmissions result in background noise heard on the channel, which is distinguishable from any meaningful message.

Adversary: Machines are prone to crashes because of an adversary that interferes with the system. The adversary is characterized by its power f and this defines how many crashes it may enforce, where $0 \leq f \leq m-1$. Thus, at least one machine remains operational in an arbitrary execution. Crashes are permanent, so machines do not restart.

The adversary of our interest is the *Adaptive f-Bounded* adversary. It may decide arbitrarily which machines will be crashed at any given moment.

Complexity measure: The complexity measure that is used throughout this paper is *work*. It is the number of available machine steps for computations. This means that each operational machine that did not halt or crash contributes a unit of work even if it is idling.

Precisely: consider algorithm A and assume that execution $\mathcal{E}$ starts when all the machines begin simultaneously in some fixed round r_0. Let r_v be the round when machine v halts or is crashed. Then its work contribution is equal $r_v - r_0$. In what follows, work accrued by A in execution $\mathcal{E}$ is the sum of such expressions over all machines, i.e.: $\sum_{1 \leq v \leq m}(r_v - r_0)$. The work complexity of A is the maximum over all possible executions of A.

Work is a standard complexity measure for analyzing the Do-All problem in various models [11].

Jobs and reliability: Machines have entire knowledge about the jobs, their ID's and their dependencies. Jobs are assumed to be *similar* (require the same number of rounds to be performed), *idempotent* (each can be performed many times, even concurrently by different machines) and *dependent* (the order in which they should be performed is described by a partial order relation).

The relation of our particular interest is the precedence relation $\prec$. If $a \prec b$ in the partial order, then we say that job a precedes job b and hence job a must be done before job b. If it holds that $a \prec b$ or $b \prec a$, we say that a and b are comparable. A subset of jobs where each pair of jobs is comparable is called a *chain*. A subset of jobs where no two different jobs are comparable is called an anti-chain.

A job is considered *done*, when it is performed by a machine and it is *confirmed*, when a machine broadcasts the fact of it being *done*. Consider some job a. Before a particular machine begins working on a all jobs preceding a must be done and confirmed on the channel.

Performing jobs in a certain order and confirming them via the channel is motivated by the need of obtaining the outputs of previous jobs as the input for the following ones. Hence, we assume that machines transmit results of certain jobs in a confirmation broadcast and this opens access to the proceeding jobs for all machines. If some machine is working on a particular chain of jobs, then it may be doing consecutive jobs in this chain without confirming them on the channel until the last job in the chain is done, as it has required inputs for consecutive jobs computed locally.

We assume that our algorithms are *reliable* i.e., in any execution: all the jobs are eventually performed, if at least one machine remains non-faulty. Moreover each machine eventually halts, unless it has crashed.

III. Sets of chains

Considerations in [14] include the problem of performing n preemptive jobs of an arbitrary length, by m machines. If performing some of the jobs takes more than one computational time unit, then a natural question arises, whether such jobs have to be performed fully by a specific machine (non-preemptive model), or any intermediate progress is remembered and jobs may be reclaimed by some other machine (preemptive model).

By preemption we define the option of performing jobs partially. Consider job a of length l_a. If machine v is to perform job a and performs x units of this job, then the remaining part of $l_a - x$ units of job a may be done by some other machine w.

Because time is divided into rounds, we can assume that there is some minimal length of a job that may be processed in one step in the preemptive setting. Thus, all jobs can be presented as a multiplicity of the minimal length. Hence, jobs are assumed be built with minimal length units, called *tasks* and in order to perform a job all its tasks have to be performed consecutively one by one. Such view was the basis for solving jobs of arbitrary lengths for the preemptive setting in [14].

Having explained preemptive jobs of arbitrary lengths, the correspondence to partially ordered sets of chains is simple.

Tasks from the preemptive model will now represent *jobs* and respectively *jobs* will be substituted by *chains*.

The described adjustments are actually only syntactic changes, that allow us to improve the presentation and we denote the resulting algorithm for solving chains of jobs ALGORITHM A. Consequently, all the results translate straightforwardly, so we can state the lower bound for our problem:

Theorem 1: ([14] Theorem 1) The *Adaptive f-Bounded* adversary, for $0 \leq f < m$, can force any reliable, possibly randomized and centralized algorithm and partially ordered sets of chains of jobs, to perform work $\Omega(n+m\sqrt{n}+m\min\{f,n\}+mH)$, where H represents the longest chain of jobs.

Theorem 2: ([14], Theorem 2) ALGORITHM A performs work $\mathcal{O}(n+m\sqrt{n}+m\min\{f,n\}+mH)$ against the adaptive adversary and partially ordered sets of chains of tasks, where H represents the longest chain.

IV. TREES

In this section we analyze trees "growing upwards", making the root the least node. Observe that in such a critical part of the partial order, as near the root, it is better to direct all effort to perform all initial jobs, in order to open access to a wider range of jobs which can be performed in parallel.

Our idea is based on decomposing the tree into a number of layers and then performing them in consecutive phases. In general machines can start performing jobs from layer i, when all jobs from layer $i-1$ are done and confirmed.

Lemma 1: A reliable algorithm, possibly centralized and randomized, performs work $\Omega(mH)$ even in an execution in which no failures occur, on a partial order of jobs forming an upwards growing tree, where H is the height of the tree.

Combining Theorem 1 with the above leads us to the lower bound for work in the assumed model.

Corollary 1: A reliable algorithm on a tree-shaped partial order of jobs has to perform work $\Omega(n+m\sqrt{n}+m\min\{f,n\}+mH)$ against the *Adaptive f-Bounded* adversary.

A. ALGORITHM B*: construction*

We begin with a construction allowing to decompose an arbitrary tree into a set of disjoint components.

Tree decomposition. Consider the following construction. We label each node in the tree by a pair (x,c) of natural numbers, representing its *colour* and *counter*, respectively. Each leaf is labelled $(1,1)$, i.e., its colour equals 1 and its counter is set to 1 as well. All non-leaf nodes are labelled recursively according to the procedure described below.

Let us consider node v. Let x be the **maximal** colour over v's children.

- If exactly one child of v has colour x and its counter is c, then v is labelled as $(x, c+1)$.
- If $l > 1$ children of v have colour x, let $(x,c_1),(x,c_2),\ldots,(x,c_l)$ denote their labels. We consider two cases dependently on the value $c=\sum_{i=1}^{l} c_i$.
  - If $c \geq 2n/m$, then v is labelled as $(x+1,1)$. (We reset the counter and start a group with a new colour).
  - Otherwise, if $c < 2n/m$, then v is labelled as $(x, c+1)$. (We add v to the group marked with colour x).

This procedure partitions the tree into subtrees labelled with different colours. Let us consider the properties of this partition.

Lemma 2: Let $\mathcal{T}$ be a tree labelled according to the procedure described above with n nodes and let H be the length the longest path in $\mathcal{T}$.

1) None of the nodes will have colour greater than $\log m+1$.
2) Every connected monochromatic component (i.e., connected set of nodes, labelled with the same colour) has at most $2n/m+H$ nodes.

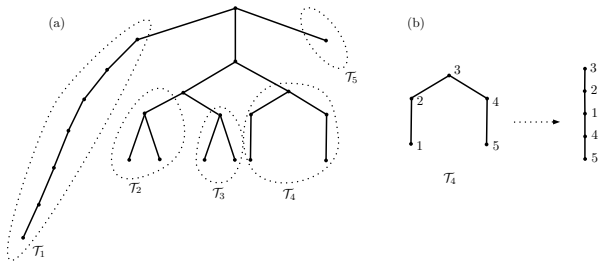

Fig. 1. (a) Decomposition of a tree. Layer 1 has 5 trees which can be performed in parallel. (b) Subtree into chain translation.

ALGORITHM B design. Let us recall, that machines have entire knowledge about the partial order of jobs. In what follows machines may perform local computations regarding the partial order and can follow the construction above.

Using the aforementioned construction we perform tree decomposition as follows. Let the i-th *layer* be the set of all nodes with colour i. Clearly each layer consist of monochromatic disjoint trees of size at most $\frac{2n}{m}+H$ each, by Lemma 2. We refer to this procedure as DECOMPOSE-TREE in the pseudo-code of Algorithm 1.

Each such tree in a given layer can be *translated* into a chain as follows: having a tree we construct a chain of the same nodes that preserves the order of the tree, i.e., if $a \prec b$ in the tree then $a \prec b$ in the chain as well. Note that it is possible that $a \prec b$ in the chain, while a and b are incomparable in the original tree. Clearly, such translation is not unique, except the case when the tree is a chain.

Consequently, each layer is mapped to a set of independent chains that can be performed by ALGORITHM A. In order to preserve the clarity of the algorithm, in the pseudo-code we simply execute SUBTREES-INTO-CHAINS to perform the above mentioned operation. To complete ALGORITHM B we simply execute ALGORITHM A for each of the l consecutive layers.

We end with a statement about the work performance of ALGORITHM B.

Theorem 3: ALGORITHM B performs work $\mathcal{O}(m\sqrt{n\log m}+m\min\{f,n\}+n\log m+mH\log m)$ for m machines and n jobs ordered in a tree of height H in the presence of an *Adaptive f-Bounded* adversary.

Algorithm 1: ALGORITHM B: machine v

initialize MACHINES to a sorted list of all m names of machines;
initialize JOBS to a sorted list of all n names of jobs;
initialize CHAINS to an empty list;
initialize LAYERS to an empty list;
$i := 0$;
LAYERS := DECOMPOSE-TREE;
repeat
  CHAINS = SUBTREES-INTO-CHAINS(LAYERS$_i$);
  execute ALGORITHM A(v, CHAINS, JOBS, MACHINES);
  $i++$;
until $|JOBS| = 0$;

Proof 1: ALGORITHM B decomposes the partial order using the technique described in Section IV-A. All jobs having a particular colour are grouped into a single layer. Hence, the total work depends on the number of layers and the work that is necessary in order to perform all the jobs within a single layer.

Let n_i be the number of jobs in the i-th layer and f_i be the number of machines crashed when the i-th layer is being performed. H_i denotes the largest subtree within layer i. Since the i-th layer consists of n_i jobs, which are translated into chains, so running ALGORITHM A on this layer performs all the jobs with work W_i which does not exceed $O(n_i + m\sqrt{n_i} + m\min\{f_i, n_i\} + mH_i)$, according to Theorem 2.

On the other hand, we know that the longest chain in the i-th layer was constructed from a single subtree. Hence by Lemma 2 we know that $H_i < 2n/m + H$.

The total work, calculated over all l layers, does not exceed $\sum_{i=1}^{l} W_i \leq \sum_{i=1}^{l} O(n_i + m\sum_{i=1}^{l}\sqrt{n_i} + m\sum_{i=1}^{l}\min\{f_i, n_i\} + m\sum_{i=1}^{l}(2n/m + H)) \leq O(n + m\sum_{i=1}^{l}\sqrt{n_i} + m\min\{f, n\} + l \cdot n + lmH)$.
To estimate this further we need the following fact, that can be easily proved using the Lagrange multipliers method:

Fact 1: Let $n_1 + n_2 + \ldots + n_l = n$ and $n_i > 0$ for all i. Then $\sum_{i=1}^{l}\sqrt{n_i} \leq \sqrt{l \cdot n}$.
Applying this fact to the estimate gives us that $O(n + m\sum_{i=1}^{l}\sqrt{n_i} + m\min\{f, n\} + l \cdot n + lmH) = O(m\sqrt{ln} + m\min\{f, n\} + l \cdot n + lmH)$.
According to the first point of Lemma 2 we know that $l \leq \log m + 1$, so we conclude that the overall work of ALGORITHM B is $O(m\sqrt{n\log m} + m\min\{f, n\} + n\log m + mH\log m)$.

V. CONCLUSIONS

The tree decomposition construction in IV-A is independent of the direction the tree grows, so ALGORITHM B can be directly applied on a downwards growing tree with the root being the greatest node and a generalized tree combining a tree growing upwards and a tree growing downwards with the same asymptotic work complexity.

An interesting open direction is to investigate whether there are some classes of adversaries that can make an explicit use of the impediment in the form of job dependencies and impose greater work, even for some simple partial orders. Furthermore, it is interesting to investigate whether one can show a universal solution for arbitrary partially ordered sets of jobs.

REFERENCES

[1] E. Bampis et al. A note on multiprocessor speed scaling with precedence constraints. In *SPAA*, 2014.
[2] C. Chekuri et al. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *D. Appl. Maths.*, 98(1):29–38, 1999.
[3] B. Chlebus et al. Performing tasks on synchronous restartable message-passing processors. *Distrib. Comput.*, 14(1):49–64, Jan. 2001.
[4] B. Chlebus et al. Bounding work and communication in robust cooperative computation. In *DISC*, 2002.
[5] B. Chlebus et al. Randomization helps to perform independent tasks reliably. *Rand. Struct. & Algo.*, 24(1), 2004.
[6] B. Chlebus et al. Performing work in broadcast networks. *Distributed Computing*, 18(6):435–451, 2006.
[7] F. Chudak et al. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *J. of Algo.*, 30(2):323 – 343, 1999.
[8] R. De Prisco et al. Time-optimal message-efficient work performance in the presence of faults. In *PODC*, 1994.
[9] C. Dwork et al. Performing work efficiently in the presence of faults. *SIAM J. Comput.*, 27(5):1457–1491, 1998.
[10] Z. Galil et al. Resolving message complexity of byzantine agreement and beyond. In *FOCS*, page 724, 1995.
[11] C. Georgiou and A. A. Shvartsman. *Cooperative Task-Oriented Computing: Algorithms and Complexity*. 2011.
[12] J. Hurink and S. Knust. List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, 29(5):231 – 239, 2001.
[13] M. Klonowski et al. Ordered and delayed adversaries and how to work against them on a shared channel. *Dist. Comp.*, Sep 2018.
[14] M. Klonowski et al. Fault-tolerant parallel scheduling of arbitrary length jobs on a shared channel. In *FCT*, 2019.
[15] E. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Manage. Sci.*, 19(5), 1973.
[16] E. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In *Algorithmic Aspects of Combinatorics*, volume 2. 1978.
[17] E. L. Lawler. Preemptive scheduling of precedence-constrained jobs on parallel machines. In *Deterministic and Stochastic Scheduling*, pages 101–123, 1982.
[18] J. Lenstra et al. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, Feb. 1978.
[19] J. Leung et al. Minimizing total tardiness on a single machine with precedence constraints. *ORSA J. on Comp.*, 2(4), 1990.
[20] J. Leung et al. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. 2004.
[21] I. N. Lushchakova. Two machine preemptive scheduling problem with release dates, equal processing times and precedence constraints. *Euro. J. Op. Res.*, 171(1), 2006.
[22] K. Pruhs et al. Speed scaling of tasks with precedence constraints. *TOCS*, 43(1), Jul 2008.
[23] M. Skutella et al. Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *SODA*, 2001.
[24] G. Woeginger. On the approximability of average completion time scheduling under precedence constraints. In *ICALP*, 2001.