Fault-tolerant parallel scheduling of arbitrary length jobs on a shared channel

Marek Klonowski* Dariusz R. Kowalski† Jarosław Mirek†
Prudence W.H. Wong†

October 24, 2017

Abstract

We study the problem of scheduling jobs on fault-prone machines communicating via a shared channel, also known as multiple-access channel. We have n arbitrary length jobs to be scheduled on m identical machines, f of which are prone to crashes by an adversary. A machine can inform other machines when a job is completed via the channel without collision detection. Performance is measured by the total number of available machine steps during the whole execution. Our goal is to study the impact of preemption (i.e., interrupting the execution of a job and resuming later in the same or different machine) and failures on the work performance of job processing. The novelty is the ability to identify the features that determine the complexity (difficulty) of the problem. We show that the problem becomes difficult when preemption is not allowed, by showing corresponding lower and upper bounds, the latter with algorithms reaching them. We also prove that randomization helps even more, but only against a non-adaptive adversary; in the presence of more severe adaptive adversary, randomization does not help in any setting. Our work has extended from previous work that focused on settings including: scheduling on multiple-access channel without machine failures, complete information about failures, or incomplete information about failures (like in this work) but with unit length jobs and, hence, without considering preemption.

*Faculty of Fundamental Problems of Technology, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland. Email: `Marek.Klonowski@pwr.edu.pl`

†Department of Computer Science, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, UK. Email: `{d.kowalski,j.mirek,pwong}@liverpool.ac.uk`

1 Introduction

We examine the problem of performing n jobs by m machines reliably on a multiple-accessed shared channel. This problem in such model, originated by Chlebus, Kowalski and Lingas [5], has already been studied for unit length jobs, whereas this paper extends it by considering jobs with arbitrary lengths, studying the impact of features including preemption, randomization, and severity of failures on the work performance.

The notion of preemption may be understood as the possibility of not performing a particular job in one attempt. This means that a single job may be interrupted halfway through performing it and then be resumed later by the same machine or even by another machine. Intuitively, the model without preemption is a more general configuration, yet both have subtleties that distinguish them noticeably. In general, algorithms exploiting randomization may improve the performance and we aim to investigate when randomization actually helps.

The communication takes place on a multiple access channel without collision detection. We consider the impact of the severity of adversaries (measuring the severity of machine failures) on the performance. In particular, we distinguish between adaptive and non-adaptive adversaries. An *adaptive f-bounded* adversary may crash up to f machines online at any time of the execution. A *non-adaptive* adversary has to additionally declare in the beginning of the execution which, up to f, machines and when to crash. The former adversary corresponds to a malicious opponent infecting the system and disturbing the computation online, while the latter depicts the scenario when all machines have their natural life cycles unknown to the scheduling algorithm. We use work as the effectiveness measure, i.e., the total number of machine steps available for computations. This measure is also related (after division by m) to the average time performance of available machines.

1.1 Previous and related results

Our work can be seen as an extension of *Do-All* problem defined in the seminal work by Dwork, Halpern and Waarts [7] followed up by several papers [2, 3, 4, 6, 8] considered for the message-passing model with point-to-point communication. In all those papers the authors assumed that the performance of a single job contributes a unit to the work complexity. Paper [6] introduced a model, wherein the measure of work for solutions of *Do-All* is the available machine steps (i.e., including idle rounds). They developed an algorithm which has work $\mathcal{O}(n + (f + 1)m)$ and message complexity $\mathcal{O}((f + 1)m)$. We also adopt this measurement in our paper.

In [8] the authors improved the message complexity to $\mathcal{O}(fm^{\varepsilon} + \min\{f+1, \log m\}m)$, for any $\varepsilon > 0$ keeping the same work complexity. *Do-All* problem with failures controlled by a weakly-adaptive linearly-bounded adversary was studied in [4], wherein the authors demonstrated randomized algorithm with expected work $\mathcal{O}(m \log^* m)$.

The authors of [3] developed a deterministic algorithm with effort $\mathcal{O}(n + m^a)$, for a specific constant $1 < a < 2$, against the unbounded adversary. They presented the first algorithm for this type of adversary with both work and communication $o(n+m^2)$. They also gave an algorithm achieving both work and communication $\mathcal{O}(n+m\log^2 m)$ against a strongly-adaptive linearly-bounded adversary.

In [10] an algorithm based on gossiping protocol with work $\mathcal{O}(n+m^{1+\varepsilon})$, for any fixed constant ε, is demonstrated. In [14] *Do-All* is studied for an asynchronous message-passing mode when executions are restricted such that every message delay is at most d. The authors proved $\Omega(n+md\log_d m)$ on the expected work. They developed several algorithms, among them a deterministic one with work $\mathcal{O}((n+md)\log m)$. Further developments and a more comprehensive related literature can be find in the book by Georgiou and Shvartsman [11].

The work closest to ours is *Do-All* problem of unit length jobs on a multiple access channel with no collision detection, which was first studied in [5]. The authors have showed a lower bound of $\Omega(n+m\sqrt{n})$ on the work when there is no crash, and $\Omega(n+m\sqrt{n}+m\min\{f,n\})$ in the presence of crashes against an adaptive f-bounded adversary. They also proposed an algorithm, called Two-Lists, with optimal performance $\Theta(n+m\sqrt{n}+m\min\{f,n\})$. The paper also studied other variants that are out of the scope of this paper. A recent paper [12] discusses different models of adversaries on a multiple access channel in the context of performing unit length jobs.

Extending the study of unit length jobs to arbitrary length jobs has taken place in other context, c.f., [15] for recent results and references to such extension in the context of fault-tolerant centralized scheduling. There has been a long history of studying preemption vs non-preemption, e.g., [17, 16], to which our work also contributes.

1.2 Our results

In this paper, we consider scheduling n arbitrary length jobs on m machines reliably over a multiple accessed channel. In the setting with job preemption, we prove a lower bound $L+\Omega(m\sqrt{L}+m\min\{f,L\}+m\alpha)$ on work, where L is the sum of lengths of all jobs and α is the maximum length, which holds for both deterministic solutions and randomized algorithms against an adaptive *f-bounded* adversary. We design a corresponding deterministic distributed algorithm, called ScaTri, achieving work $L+\mathcal{O}(m\sqrt{L}+m\min\{f,L\}+m\alpha)$, which matches the lower bound asymptotically with respect to the overhead. In the model without job preemption, we show slightly higher lower bound $L+\Omega\left(\frac{L}{n}m\sqrt{n}+\frac{L}{n}m\min\{f,n\}+m\alpha\right)$ on work, implying a separation between the two settings with and without preemption. Similarly as in the previous setting, it holds for both deterministic solutions and randomized algorithms against an adaptive *f-bounded* adversary, thus showing that randomization does not help against adaptive adversary regardless of (non-)preemption setting. We develop a corresponding deterministic distributed algorithm, called DefTri, achieving work $L+\mathcal{O}(\frac{L}{n}m\sqrt{n}+\alpha m\min\{f,n\})$. We then investigate the scenario with randomization and derive a randomized distributed preemptive algorithm, called RanScaTri, against non-adaptive adversaries. This algorithm achieves work $\mathcal{O}(L+m\sqrt{L}+m\alpha)$, and thus shows that randomization helps against non-adaptive adversary, lowering the performance to the absolute lower bound $L+\Omega(m\sqrt{L}+m\alpha)$ for the model. The results are discussed and compared in details in Section 6. Figure 1 illustrates the complexity for different variants of the considered problem.

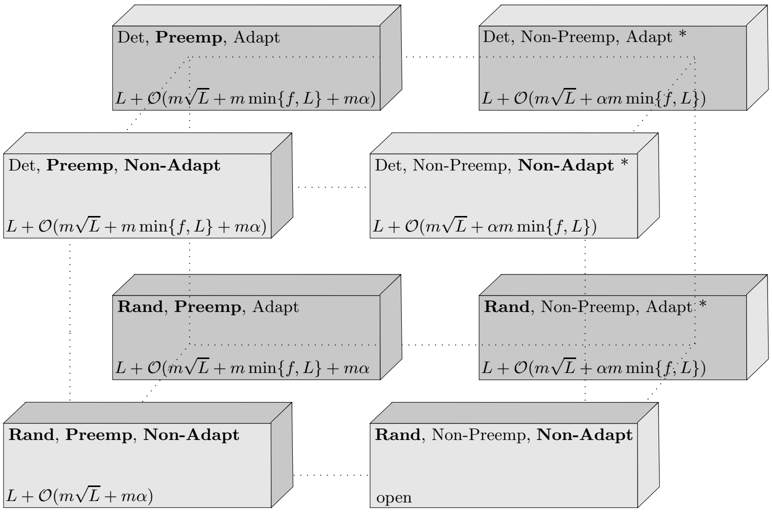

Figure 1: Illustration of the complexity of scheduling on mac. Legend: Rand - randomized algorithm, Det - deterministic algorithm, Preemp - preemptive setting, Non-Preemp - non-preemptive setting, Adapt - adaptive adversary, Non-Adapt - non-adaptive adversary.

1.3 Document structure

Section 2 describes the model and technical preliminaries. In Section 3 we prove a lower bound and provide an algorithm matching its performance formula (asymptotically with respect to an overhead) in the model with preemption. Section 4 provides similar study for the model without preemption. Section 5 is dedicated to a randomized algorithm solving the problem with minimal work complexity. The results in various settings are discussed and compared in Section 6, and final conclusions given in Section 7.

2 Technical preliminaries

In this section we describe the formal model i.e.: machine capabilities, communication model, failure model and the concept of adversary, the complexity measure used for benchmarking our algorithms and a high-level specification of a black-box procedure TaPeBB that we will be using while designing our algorithms. Additionally we present definitions of performing jobs and the technicalities connected with preemption.

2.1 Machines

In our model we assume having m machines, with unique identifiers from the set $\{1, \ldots, m\}$. The distributed system of those devices is synchronized with a global clock, and time is divided into synchronous time slots, called *rounds*. All the machines start simultaneously at a certain moment. Furthermore every machine may halt voluntarily. A halted machine does not contribute to work, in contrast to an operational machine that contributes to the work performance measure. Precisely, every operational machine generates a unit of work per round, that has to be included while computing an algorithm's complexity. In this paper by M we will denote the number of operational, i.e., not crashed, machines.

2.2 Communication model

The communication channel for machines is the, widely considered in literature, *multiple-access channel* [1, 9], also called a *shared channel*, where a broadcasted message reaches every operational device. We do not allow simultaneous successful transmissions of several messages. In what follows our solutions work on a channel without collision detection, hence when more than one message is transmitted at a certain round, then the devices hear a signal indifferent from the background noise.

2.3 Adversarial model

Machines may fail, which happens because of an adversarial activity. One of the factors that describes the adversary is her power f, that is, the total number of failures that may be enforced. We assume that $0 \leq f \leq m-1$, thus at least one machine remains operational until an algorithm terminates. Machines that were crashed neither restart nor contribute to work.

The adversarial model of our particular interest is the *adaptive f-bounded* — the adversary can observe the execution and make decisions about up to f crashes online. Another adversarial model that we will consider in this paper is the *non-adaptive f-bounded* — apart from choosing the faulty subset of f machines, the adversary has to declare in the beginning of the computation in which rounds those crashes will occur, i.e., for every machine declared as faulty, there must be a corresponding number of round in which the fault will take place. It is worth noticing that in the context of deterministic algorithms such an adversary is consistent with the adaptive adversary that may decide online which machines will be crashed at any time, as the algorithm may be simulated by the adversary before its execution, providing knowledge about the best decisions to be taken. Finally, an $(m-1)$-bounded adversary is also called an *unbounded* adversary.

2.4 Complexity measure

The complexity measure that will be mainly used in our analysis is, as mentioned before, *work*, also called *available processor steps*. It is the number of available machine steps for computations. This means that each operational machine that did not halt contributes a unit of work even if it is idling.

In order to precisely describe this complexity measure let us assume that an execution 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 $r_v - r_0$. Consequently the algorithm complexity is the sum of such expressions over all machines i.e.: $\sum_{1 \leq v \leq p}(r_v - r_0)$.

2.5 Jobs and reliability

We expect that machines will perform all n jobs as a result of executing an algorithm. We assume that *jobs have arbitrary lengths*, they are *independent* (they may be performed in any order) and *idempotent* (every job may be performed many times, even concurrently by different machines).

Furthermore a *reliable* algorithm satisfies the following conditions in any execution:

1. *All the jobs are eventually performed, if at least one machine remains non-faulty.*

2. *Each machine eventually halts, unless it has crashed.*

Let us assume that jobs have some minimal (atomic or unit) length. For simplicity, we can also think that all the lengths are a multiple of the minimal length. The model that we consider is synchronous, so this minimal length may be justified by the round duration - required for local computations for each machine. By ℓ_a we denote the length of job a. We will also use L for describing the sum of all the lengths of jobs i.e. $L = \sum_i \ell_i$. Finally, by α let us denote the length of the longest job.

2.6 Preemptive vs Non-Preemptive Model

By the means of *preemption* we define the possibility of performing jobs in several pieces. Precisely, let us consider that there is some job a, of length ℓ_a (for simplicity we assume that ℓ_a is divisible by 2) and machine v is scheduled to perform this job at some time of the algorithm execution. Assume that v performs $\ell_a/2$ units of job a and then reports such progress. When *preemption* is available the following $\ell_a/2$ units of job a may be performed by some machine w where $w \neq v$.

Let us consider a particular job a od length ℓ_a. This means that job a consists of ℓ_a time units needed to perform it fully. Such view allows to think that the job is a chain of ℓ_a *tasks*, and hence all jobs form a set of chains formed by unit length tasks.

Having defined the abovementioned, we may now use a_k for denoting task k of job a. However, when we would like to refer to a single job, disregarding its tasks, we will refer simply to job a.

We define two types of jobs within this model:

- **Oblivious** — it is sufficient to have the knowledge that previous tasks were done, in order to perform others from the same job. In other words, any information from the in between progress does not have to be announced.

- **Non-Oblivious** — a job for which any intermediate progress needs to be reported through the channel when passing the job to another machine.

In the preemptive oblivious model a job may be abandoned by machine v on some task k without confirming progress up to this point on the channel and then continued from the same task k by some other machine w.

As an example, let us consider a scenario that there is a job that a shared array of length x needs to be erased. If a machine stopped performing this job at some point, another one may reclaim that job without the necessity of repetition by simply reading to which point it has been erased.

For the preemptive non-oblivious model let us consider that a machine executes Dijkstra's algorithm for finding the shortest path. If it becomes interrupted, then another machine cannot reclaim this job otherwise than by performing the job from the beginning, unless intermediate computations have been shared. In other words preemption is available with respect to maintaining information about tasks.

On the contrary to the preemptive model, we also consider the model without preemption i.e. where each job can be performed by a machine in one piece - when a machine is crashed while performing a long job, the whole progress is lost, even if it was reported on the channel before the crash took place.

2.7 The Task-Performing Black Box (TaPeBB)

When considering jobs that have different lengths, we are rather interested in the matter of how to insert a chunk of input (jobs forming chains of consecutive tasks) to a procedure, which as a result would perform them reliably or inform that something went wrong. For this reason and for clarity of presentation, we concentrate on the most important ideas regarding our algorithms and treat the parts where and how the jobs are actually performed as a black box procedure. In what follows, we specify this procedure, called *Task-Performing Black Box* (TaPeBB for short), and argue that it can be implemented.

General properties of TaPeBB. A synchronous system is characterized by time being divided into synchronous slots, that we already called rounds. In what follows, each round is a possibility for machines to transmit. The nature of the problem of jobs with different lengths leads however to a concept that the time between consecutive broadcasts needs to be adjusted. Especially that when a job is long — then for some configurations it may be better to broadcast the fact of performing the job fully, rather than broadcasting semi-progress multiple times. Therefore, we assume that TaPeBB has a feature of changing the duration between consecutive broadcasts and we will call this parameter a *phase*. Let us denote the length of a phase by ϕ. Nevertheless, if not stated differently we assume that $\phi = 1$, i.e., the duration of a phase and a round is consistent, thus machines may transmit in any round.

Input. $\texttt{TPBB}(v, d, \texttt{JOBS}, \texttt{MACHINES}, \phi)$

- TaPeBB takes a list of machines `MACHINES` and a list of jobs `JOBS` as the input, yet from the task perspective, i.e., the jobs are provided as a chain of tasks and are performed according to their ids. It may happen that a job will not be done fully — just the initial segment of tasks forming that job may be performed.
- Additionally, the procedure takes an integer value d. It specifies the number of machines that will be used in the procedure for performing provided pieces of jobs, which is related to the amount of work accrued during a single execution. The procedure works in such a way that each working machine is responsible for performing a number of jobs.
- We call a single execution an *epoch* and the parameter d will be called the length of an epoch (i.e. the number of phases that form an epoch).
- Input of TaPeBB includes machine id v on which it is executed.
- For clarity we assume that TaPeBB always uses the initial d machines from the list of machines.

Output.

- Having explained what is the length of an epoch in TaPeBB the capability of performing tasks in a single epoch (understood as the maximal number of jobs that may be confirmed in an epoch, when it is executed fully without any adversarial distractions) is at least $\sum_{j=1}^{d} j$ which is the sum of an arithmetic serie with common difference equal 1. We will sometimes refer to this capability as a *triangle*, as if we take a geometric approach and draw lines or boxes of increasing lengths, then drawing this sequence would form a triangle.
- As a result of running a single epoch we have an output information about which tasks were actually done and were there any machines identified as crashed, when machines were communicating progress (machines identify which jobs were done after an epoch by comparing information about performed tasks).

One may ask whether such an approach is realistic and whether there are algorithms fulfilling all the abovementioned requirements. The answer is positive and as an example of an algorithm that may serve as TaPeBB is the Two-Lists algorithm from [5], which was an inspiration for this paper and so details are to be seen therein.

Fact 1. *TPBB can be implemented to an algorithm in such a way so that its each execution performs at most $\sum_{j=1}^{d} j$ tasks while generating $md\phi$ work.*

Proof. We have at most m machines working for d phases, and the length of each phase is set to ϕ rounds, hence this gives $md\phi$ units of work in total. □

3 Preemptive model

In this section we consider the scheduling problem in the model with preemption, which is intuitively an easier model to tackle. As mentioned before, the multiple-access channel has no collision detection. We first show a lower bound for oblivious jobs (Section 3.1) and then present and analyze our algorithm for non-oblivious jobs with matching performance (Section 3.2). Notice that a lower bound for oblivious jobs is a stronger lower bound as it also applies for non-oblivious jobs and similarly an upper bound for non-oblivious jobs is a stronger upper bound. The matching bounds imply that with preemption, the distinction between oblivious jobs and non-oblivious jobs does not matter.

3.1 Lower bound

We first recall the minimal work complexity introduced in [5].

Lemma 1 ([5], Lemma 2)**.** *A reliable, distributed algorithm, possibly randomized, performs work $\Omega(n + m\sqrt{n})$ in an execution in which no failures occur.*

As jobs are built from unit length tasks, we can look at this result from the task perspective. Precisely, if we consider that L represents the number of tasks needed to be performed by the system, then the lower bound for our model translates in a straightforward way. Furthermore, because factor n appeared in the formula above with factor 1, then we may take it out of the asymptotic notation.

Furthermore in our model there is a certain bottleneck dictated by the longest job. After all there may be an execution with one long job, in comparison to others being short. This gives the magnitude of work in the complexity formula.

Lemma 2. *A reliable, distributed algorithm, possibly randomized, performs work $L + \Omega(m\sqrt{L} + m\alpha)$ in an execution in which no failures occur on any set of oblivious jobs with arbitrary lengths with preemption.*

Proof. $L + \Omega(m\sqrt{L})$ follows from Lemma 1 trivially. Let us therefore concentrate on the remaining part. When considering jobs that have arbitrary lengths there is always a possibility that some additional work must be done. Especially in the case when there are a few jobs much longer than the overall average length of jobs. In particular, the worst case is when there is one such long job. For this reason the best solution is when all the machines perform the job simultaneously (otherwise some machines would be idle and when adversarial crashes take place then it could prolong the execution). This gives us $m\alpha$ work. The lemma then follows. □

In the presence of failures, we extend the following result.

Lemma 3 ([5], Theorem 2)**.** *The f-bounded adversary, for $0 \le f < m$, can force any reliable, possibly randomized, algorithm to perform work $\Omega(n + m\sqrt{n} + m\min\{f, n\})$.*

Combining all the results above, we conclude with the following theorem, setting the lower bound for the considered problem.

Theorem 1. *The adaptive f-bounded adversary, for $0 \leq f < m$, can force any reliable, possibly randomized and centralized algorithm on the channel without collision detection and a set of oblivious jobs of arbitrary lengths with preemption, to perform work $L + \Omega(m\sqrt{L} + m\min\{f, L\} + m\alpha)$*

Proof. Because now the magnitude of work concentrates on the overall number of tasks L, yet the number of crashes remains the same, the result follows explicitly by combining Lemma 2 and Lemma 3. $\square$

3.2 Algorithm ScaTri

In this section, we present our algorithm Scaling-Triangle MAC Scheduling (ScaTri for short). Intuitively, we use TaPeBB which treats the capability of performing jobs as a triangle (see Section 2). This capability may vary through the execution (because of machine crashes and job completion), and effectively this is like rescaling the triangle. The most important assumption for ScaTri is that machines have access to all the jobs and the corresponding tasks that build those jobs. This means that they know all the jobs (and tasks) and their id's as well as their lengths.

In what follows we assume that each machine maintains three lists: `MACHINES`, `TASKS` and `JOBS`. The first list is a list of operational machines and is updated according to the information broadcasted through the communication channel. If there is information that some machine was recognized as crashed, then this machine is removed from the list. In the context of the TaPeBB procedure, let us remind that this is realized as the output information. TaPeBB returns the list of crashed machines so that operational machines may update their lists.

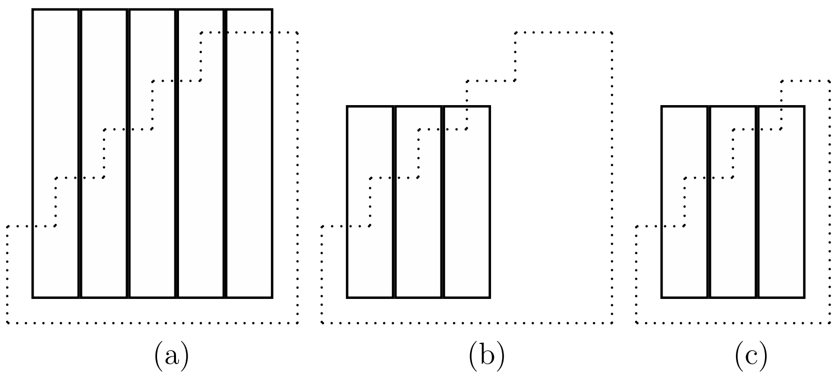

Figure 2: A very general idea about how ScaTri works. The vertical stripes represent jobs and the dotted-line triangle is the capability of the algorithm to perform jobs in a single epoch (or parts of them i.e. some consecutive tasks). Consequently, if there are enough jobs to pack into an epoch, then it is executed (a). Otherwise, if there are not enough jobs for the current length of an epoch (b), then the length of the epoch is reduced (c). This helps preventing excessive idle work.

List TASKS represents all the tasks, that are initially computed from the list of jobs. Every task has its unique identifier, what allows to discover to which job it belongs and what is its position in a certain job. This allows to preserve consistency and coherency: task k cannot be performed before task $k-1$. If some tasks are performed then this fact is also updated on the list.

List JOBS represents the set of jobs — it is a convenient way to know what the consecutive parts of input are for the TaPeBB procedure. Pieces of information on list JOBS may be updated strictly from the ones from list TASKS. TaPeBB also returns information about which jobs were actually done. Speaking about lists, we denote by $|\texttt{XYZ}|$ the length of the list XYZ and $M = |\texttt{MACHINES}|$ the actual number of operational machines.

Jobs are assigned to each machine at the beginning of an epoch (TaPeBB execution). We assume that machines have instant access to the lengths of jobs. While introducing the TaPeBB procedure we already mentioned the triangle-shaped capacity of the procedure to perform jobs in a single execution. Consequently, providing the input efficiently is consistent with being able to fill in such a triangle entirely.

We assume d is a parameter describing the actual length of an epoch. Initially it is set to m, but it may be reduced while the algorithm is running. In what follows assuming that the length of the epoch d is set to $m/2^i$ for some i, and we need to fill in a triangle of size $\sum_{j=1}^{\frac{m}{2^i}} j$, initially we need to provide $m/2^i$ jobs, that will form the base of the triangle. jobs are provided as the input in such a way that the shortest ones are preferred and if there are several jobs with the same length, then those with lower id's are preferred. After having the base filled, it is necessary to look whether there are any gaps in the triangle. If so, another layer of jobs is placed on top of the base layer, and the procedure is repeated. Otherwise, we are done and ready to execute TaPeBB.

Such an approach allows to "trim" longer jobs preferably — these will have more tasks completed after executing TaPeBB than shorter ones. One should observe that performing a transmission is an opportunity to confirm on the channel that the tasks that were assigned to a machine are done. Additionally this confirms that a certain machine is still operational. In what follows these two types of aggregated pieces of information: which jobs were done, and which machines were crashed, are eventually provided as the output of TaPeBB.

When $d = m/2^i$ for some $i = 1, \ldots, \log m$ it may happen that there are not enough tasks (jobs) to fill in a maximal triangle. If this happens we shall, in some cases, reduce the job-schedule triangle (and simultaneously the length of the epoch) to $m/2^{i+1}$ and try to fill in a smaller triangle.

Finally, we shall use $\texttt{TPBB}(v, d, \texttt{JOBS}, \texttt{MACHINES}, \phi)$ to denote which machine executes the procedure, the size of the schedule and the length of the epoch, the list of jobs from which the input will be provided, the list of operational machines, and the phase duration. Let us emphasize that we described the process of performing jobs from the active perspective, i.e., the system needs to provide input to TaPeBB. Nevertheless, for the sake of clarity we assume that TaPeBB collects the appropriate input by itself according to the rules described above.

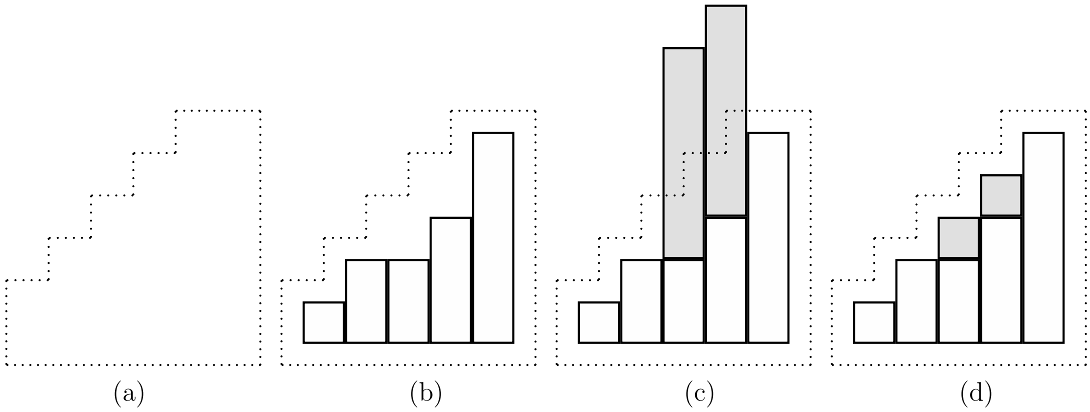

Figure 3: An illustration of job input. (a) Initially there is a certain capability of TaPeBB (triangle size) for performing jobs. (b) The initial layer is filled with jobs (white blocks), yet there are still some gaps. (c) An additional layer (gray blocks) is placed to fill in the triangle entirely. (d) In fact those additional jobs will only be done partially.

We can now analyze the work performance of our algorithm in the following theorem:

Theorem 2. ScaTri *performs work* $L + \mathcal{O}(m\sqrt{L} + m\min\{f, L\} + m\alpha)$ *against the adaptive adversary and a set of non-oblivious jobs with arbitrary lengths with preemption.*

Proof. For clarity, we divide the proof into several cases and compute work accordingly.

- Case 1: $|\texttt{JOBS}| \geq d$ and $|\texttt{TASKS}| \geq d(d+1)/2$

  If the algorithm reaches such a condition, it means that there is a significant number of jobs and tasks, thus we may assure that there is neither redundant nor idle work. That is why work accrued in phases with successful transmissions accounts to L and work in phases with failing transmissions may be estimated as $m\min\{f, L\}$. Precisely, there are clearly f crashes that may occur resulting from the adversary and because the longest length of an epoch is m, then the amount of wasted work for each crash is at most m. On the other hand the number of crashes cannot exceed the overall number of tasks, hence the minimum factor.

- Case 2: $|\texttt{JOBS}| < d$ and $|\texttt{TASKS}| \geq d(d+1)/2$

  If we fall into such case, there is a significant number of tasks, but they all form few long jobs. This means that the size of an epoch has to be reduced significantly and those jobs need to be trimmed by few initial machines in a certain number of epochs. Thus the magnitude of work for such case is $\mathcal{O}(m\alpha)$.

- Case 3: $|\texttt{TASKS}| < d(d+1)/2$

Algorithm 1: ScaTri; code for 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 TASKS to a sorted list of all tasks according to the information from
  JOBS;
- initialize variable d representing the length of an epoch;
- initialize variable φ := 1 representing the length of a phase;
- initialize i := 0;
repeat
    d := m/2^i;
    if |JOBS| ≥ d(d + 1)/2 then
        execute TaPeBB(v, d, JOBS, MACHINES, φ);
        update JOBS, MACHINES, TASKS according to the output information from
          TaPeBB;
    end
    else
        i := i + 1;
    end
until |JOBS| > 0;
```

The algorithm begins with the length of the epoch set as m. When the number of tasks is relatively small (no matter what is their distribution over jobs) this means that the algorithm should reduce the size of the schedule triangle to $m/2$ in order to reduce the number of idling rounds of machines that do not have their jobs assigned. Of course the productive part of work after rescaling the triangle, generated by machines that transmitted successfully is accounted to L and the wasted part produces $m\min\{f, L\}$ work.

Let us now compute the idling work of remaining machines. The ratio between the initial triangle size and the reduced one is

$$\frac{\frac{\frac{m}{2}(\frac{m}{2}+1)}{2}}{\frac{m(m+1)}{2}} = \frac{1}{4}\,\frac{m+2}{m} \geq \frac{1}{4}\ ,$$

hence we conclude that $\mathcal{O}(1)$ epochs is enough for a single case.

Having in mind that we are computing the overall work in the considered case as a union bound, let us examine the inequality $m(m+1)/2 > L$. It states that there are not enough jobs to fill in the initial triangle — the size of the triangle is therefore reduced to $m/2$. But because its size is reduced to $m/2$ it means that now $m/2 = \mathcal{O}(\sqrt{L})$ is the estimate for the number of idling phases within this case. Together with the fact that the number of operational machines is at most m and that the number of epochs sufficing to perform all the jobs within a single case is constant we compute the total work as follows. Following Fact 1 we have that $\mathcal{O}(1)$ epochs $\times$ m machines idling at most $\times$ $\mathcal{O}(\sqrt{L})$ phases of an epoch is the idling work accrued in a single case.

Because there is a logarithmic number of such cases and the reasoning for each of them is consistent with the abovementioned (there are not enough jobs to fill in the triangle of size $m/2^i$, hence the size is reduced to $m/2^{i+1}$), basing once again on Fact 1 we compute the total idle work as a union bound of all the cases, having the following:

$$cm \sum_i \frac{\sqrt{L}}{2^i} = \mathcal{O}(m\sqrt{L}) \ ,$$

where c is the constant number of epochs required to perform everything within a single case.

Let us note that we considered four types of work: idling, wasted, the one following from a bottleneck scenario, where there are few very long jobs and productive. Idling work has been calculated as a union bound of several cases when the length of an epoch is reduced, yet there are still some operational machines that may generate idle work.

Wasted work is a result of adversary and is clearly connected with crashes that may occur. However, if some task k is scheduled to be done in an epoch by machine v, then no other machine apart from v has task k scheduled in this particular epoch. In what follows there are two possibilities: either task k will be done by v, and the generated productive work will be accounted to L (only once for task k in the entire execution), or machine v will be crashed and the work accrued by v will be categorized as wasted (factor $m \min\{f, L\}$) and k will be scheduled once again in the following epoch.

Consequently, considering all the cases we have that ScaTri has $L + \mathcal{O}(m\sqrt{L} + m \min\{f, L\} + m\alpha)$ work complexity against the Adaptive adversary, on channel without collision detection. □

As mentioned in the beginning of Section 3, the lower bound here was proved for oblivious jobs in the preemptive model, while the algorithm works reliably for non-oblivious jobs in the same model, because TaPeBB procedure provides any intermediate job performing progress as the output information and all the information is updated sequentially after each epoch. This implies that the distinction between oblivious and non-oblivious jobs in the preemptive model and against an adaptive adversary does not matter, thus we finish this section with the following corollary:

Corollary 1. ScaTri *is optimal in asymptotic work efficiency for jobs with arbitrary lengths with preemption for both oblivious and non-oblivious jobs.*

4 Non-preemptive model

In this section we consider a complementary problem of performing jobs when preemption is not available, thus every job needs to be performed in one attempt by a single machine in order to be confirmed. Again, we will begin with the lower bound and then proceed to the algorithm and its analysis.

4.1 Lower bound

The non-preemptive model is characterized by the fact that each job can be performed by a machine in one piece. This means that a machine cannot make any intermediate progress while performing a job. If it starts working on a job then either it must be done entirely or it is abandoned without any tasks performed at all.

In what follows any intermediate broadcasts while performing a job are not helpful for the system. The only meaningful transmissions are those which allow to confirm certain jobs being done. This is the key reason to consider the notion of a phase, introduced in the model section — it seems that broadcasts regarding confirming jobs are worth doing only after entire jobs are done. Let us begin the proof of lower bound by estimating how many jobs we may be able to confirm in a certain period basing on the average length of a job. Precisely, we will show that the number of jobs that lengths are twice the average does not exceed half of the total number of jobs.

Fact 2. *Let n be the number of jobs, L be the sum of all the lengths of jobs and let $\frac{L}{n}$ represent the average length of a job. Then we have that $\left|\left\{a : \ell_a > 2\frac{L}{n}\right\}\right| \leq \frac{n}{2}$.*

Proof. We have that $L = \sum_{i \leq n} \ell_i$, so

$$L = \sum_{i \leq n} \ell_i \geq \sum_{\left\{a : \ell_i > 2\frac{L}{n}\right\}} \ell_i \geq 2\frac{L}{n} \left|\left\{a : \ell_a > 2\frac{L}{n}\right\}\right|,$$

hence $\frac{n}{2} \geq \left|\left\{a : \ell_a > 2\frac{L}{n}\right\}\right|$. □

Let us now state a general lower bound for the model even if the adversary does not distract the system.

Lemma 4. *A reliable algorithm, possibly randomized, performs work $L + \Omega(\frac{L}{n} m\sqrt{n})$ in an execution in which no failures occur.*

Proof. It is sufficient to consider the channel with collision detection in which machines could possibly have more information. Let $\mathcal{A}$ be a reliable algorithm. Let us also assume that $\mathcal{A}$ has the phase parameter set to the average length of the set of jobs. This is the duration between two consecutive transmissions, yet a particular broadcast may be omitted, if necessary.

The part L of the bound follows from the fact that all the jobs are built from L tasks, thus all the tasks need to be done at least once in any execution of $\mathcal{A}$. In other words, there are at least L units of work needed to dedicate in order to perform them all.

Job a is *confirmed* at phase γ of an execution of algorithm $\mathcal{A}$, if either a machine broadcasts successfully and it has performed a by the end of phase γ, or at least two machines broadcast simultaneously and all of them, with a possible exception of one machine, have performed job a by the end of phase γ of the execution. All of the machines broadcasting at phase γ and confirming job a have performed it by then, so at most 2γ jobs can be confirmed at phase γ. This is possible according to Fact 2, because there is a significant number of jobs with lengths that fit to the phase duration and may be confirmed in the initial phases and the longer ones may be confirmed in phases occurring later — thus the machines may always confirm a job in a phase.

Let $\mathcal{E}_1$ be an execution of the algorithm when no failures occur. Let machine v come to a halt at some phase γ' in $\mathcal{E}_1$.

Claim. *The jobs not confirmed by the end of phase γ' were performed by v itself in $\mathcal{E}_1$.*

Proof. Suppose, to the contrary, that this is not the case, and let b be such a job. Consider an execution, say $\mathcal{E}_2$, obtained by running the algorithm and crashing any machine that performed job j in $\mathcal{E}_1$ just before the machine finishes performing job b in $\mathcal{E}_1$, and all the remaining machines, except for v, crashed at phase γ'. The broadcasts on the channel are the same during the first γ' phases in $\mathcal{E}_1$ and $\mathcal{E}_2$. Hence all the machines perform the same jobs in $\mathcal{E}_1$ and $\mathcal{E}_2$ till the beginning of phase γ'. The definition of $\mathcal{E}_2$ is consistent with the power of the *unbounded* adversary. The algorithm is not reliable because job b is not performed in $\mathcal{E}_2$ and machine v is operational. We obtain contradiction that justifies the claim. □

We estimate the contribution of the machine v to work. The total number of jobs confirmed in $\mathcal{E}_1$ is at most

$$2(1+2+\ldots+\gamma') = \mathcal{O}((\gamma')^2) \ .$$

Suppose some n' jobs have been confirmed by the end of phase γ'. Each job contributes $\frac{L}{n}$ units of work because of the phase parameter. The remaining $n-n'$ jobs have been performed by v and each of them required $\frac{L}{n}$ units of work to be done as well. The work of v is therefore

$$\Omega\left(\frac{L}{n}\sqrt{n'} + \frac{L}{n}(n-n')\right) = \Omega\left(\frac{L}{n}\sqrt{n}\right) \ ,$$

which completes the proof. □

Lemma 5. *A reliable, distributed algorithm, possibly randomized, performs work $L + \Omega\left(\frac{L}{n}m\sqrt{n} + \frac{L}{n}m\min\{f,n\} + m\alpha\right)$ in an execution with at most f failures against an adaptive adversary.*

Proof. We consider two cases, depending on which term dominates the bound. If it is $L + \Omega\left(\frac{L}{n}m\sqrt{n} + m\alpha\right)$, then the bound follows from combining Lemma 4 with Lemma 2 — if the longest job was a bottleneck in the preemptive model, then certainly it is a bottleneck in the non-preemptive model.

Consider the case when $\Omega(\frac{L}{n}m\min\{f,n\})$ determines the magnitude of the bound. Denote $g = \min\{f,n\}$. As for now, we may assume that the duration of a phase, i.e., the duration between consecutive broadcasts, is set to some value ϕ and that it may even change multiple times during any execution. Let $\mathcal{E}_1$ be an execution obtained by running the algorithm and crashing any machine that wants to broadcast as a single one during the first $g/4$ phases. Denote as A the set of machines failed in $\mathcal{E}_1$. The definition of $\mathcal{E}_1$ is consistent with the power of the f*-bounded* adversary, since $|A| \leq g/4 \leq f$.

Claim. *No machine halts by phase $g/4$ in execution $\mathcal{E}_1$.*

Proof. Suppose, to the contrary, that some machine v halts before phase $g/4$ in $\mathcal{E}_1$. We show that the algorithm is not reliable. To this end we consider another execution, say, $\mathcal{E}_2$ that can be made to happen by the *unbounded* adversary. Let γ be a job which is performed in $\mathcal{E}_1$ by at most

$$\frac{mg}{4(n-g/4)} \leq \frac{mg}{3n}$$

machines, except for machine v, during the first $g/4$ phases. It exists because $g \leq n$. Let B be this set of machines. Size $|B|$ of set B satisfies the inequality

$$|B| \leq \frac{mg}{3n} \leq \frac{m}{3} \ .$$

We define operationally a set of machines, denoted C, as follows. Initially C equals $A \cup B$. Notice that the inequality $|A \cup B| \leq 7m/12$ holds. If there is any machine that wants to broadcast during the first $g/4$ phases in $\mathcal{E}_1$ as the only machine not in the current C, then it is added to C. At most one machine is added to C for each among the first $g/4 \leq m/4$ phases of $\mathcal{E}_1$, so $|C| \leq 10m/12 < m$.

Let an execution $\mathcal{E}_2$ be obtained by failing all the machines in C at the start and then running the algorithm. The definition of $\mathcal{E}_2$ is consistent with the power of the *unbounded* adversary. There is no broadcast heard in $\mathcal{E}_2$ during the first $g/4$ phases. Therefore each machine operational in $\mathcal{E}_2$ behaves in exactly the same way in both $\mathcal{E}_1$ and $\mathcal{E}_2$ during the first $g/4$ phases. Job γ is not performed in execution $\mathcal{E}_2$ by the end of phase $g/4$, because the machines in B have been failed and the remaining ones behave as in $\mathcal{E}_1$.

Machine v is not failed in $\mathcal{E}_2$ and so it performs the same actions in both $\mathcal{E}_1$ and $\mathcal{E}_2$. Consider a new execution, denoted $\mathcal{E}_3$. This execution is like $\mathcal{E}_2$ till the beginning of phase $g/4$, then all the machines, except for v, are failed. The definition of $\mathcal{E}_3$ is consistent with the power of the *unbounded* adversary. Machine v is operational but halted and job γ is still outstanding in $\mathcal{E}_3$ at the end of phase $g/4$. We conclude that the algorithm is not reliable. This contradiction completes the proof of the claim. □

Let us consider the original execution $\mathcal{E}_1$ again. It follows from the claim that there are at least $m - g/4 = \Omega(m)$ machines still operational and non-halted by the end of phase $g/4$ in execution $\mathcal{E}_1$. The duration of a phase is dictated by lengths of jobs. Hence even if ϕ was changed n times, it lasted L/n on average. Thus we may bound the magnitude of ϕ by the initial average length of jobs L/n and consequently machines have generated work $\Omega\left(\frac{L}{n} \cdot m \cdot \min\{f, n\}\right)$ by the end of phase $g/4$, what ends the proof. □

Corollary 2. *There are non-preemptive job inputs for which a reliable, distributed algorithm, possibly randomized, performs work $L + \Omega\left(\frac{L}{n} m\sqrt{n} + \alpha m \min\{f, n\}\right)$ against adaptive f-bounded adversary.*

4.2 Algorithm DefTri

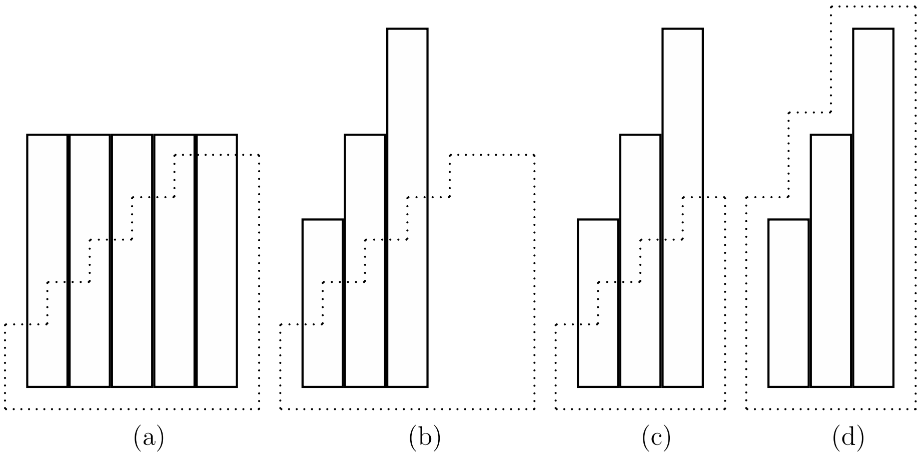

Figure 4: Most important features of DefTri. Similarly to ScaTri we apply the method of reducing idle work by reducing the input size (epoch length) for the task performing procedure ((a), (b), (c)). Additionally we pump phases (the duration between consecutive broadcasts) in order to be able to fill in jobs entirely - in the model without preemption jobs cannot be done partially (d).

We now design a scheduling algorithm in the non-preemptive model of jobs. Its work complexity is $L + \mathcal{O}(\frac{L}{n}m\sqrt{n} + \alpha m \min\{f, n\})$ against the adaptive f*-bounded* adversary. At first glance, algorithm Deforming-Triangle MAC Scheduling (DefTri for short) is similar to ScaTri introduced in the previous section and on the top of its design there is the TaPeBB procedure for performing jobs. Roughly speaking, the algorithm, repetitively, tries to choose tasks that could be packed into a specific triangle (parameters of which are controlled by the algorithm) and feed them to TaPeBB. Furthermore, once again the main goal of the algorithm is to avoid redundant or idle work. Hence, the main feature is to examine whether there is an appropriate number of jobs in order to fill in an epoch of TaPeBB. However, as we are dealing with the non-preemptive model, jobs cannot be done in small pieces (tasks).

Consequently, apart from checking the number of jobs (and possibly reducing the size of an epoch) the algorithm also changes the phase parameter, i.e., the duration between consecutive broadcasts — this is somehow convenient in order to know how the input jobs should be feeded into TaPeBB. It settles a "framework" (epoch) with "pumped rounds" that should be filled in appropriately.

Using Fact 2 assures that if the size of an epoch is reduced accordingly to the number of jobs and the phase parameter is set accordingly to the actual average length of the current set of jobs (i.e., $|\texttt{TASKS}|/\texttt{JOBS}|$ in the code of Algorithm 2), then we are able to provide the input for TaPeBB appropriately and expect that at least $\sum_{j=1}^{\frac{m}{2^i}} j$ jobs will

Algorithm 2: DefTri; code for 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 `TASKS` to a sorted list of all tasks according to the information from `JOBS`;
- initialize variable d representing the length of an epoch;
- initialize variable ϕ representing the length of a phase;
- initialize $i := 0$;

repeat
    $d := m/2^i$;
    if $|\mathit{JOBS}| \geq d(d+1)/2$ **then**
        `// set` ϕ `to the current average length of a job`
        $\phi := |\texttt{TASKS}|/|\texttt{JOBS}|$;
        execute TaPeBB$(v, d, \texttt{JOBS}, \texttt{MACHINES}, \phi)$;
        update `JOBS`, `MACHINES`, `TASKS` according to the output information from TaPeBB;
    end
    else
        if $|\mathit{TASKS}| < d(d+1)/2$ **then**
            $i := i + 1$;
        end
        else
            $\phi := 1$;
            execute TaPeBB$(v, \min\{|\texttt{JOBS}|, |\texttt{MACHINES}|\}, \texttt{JOBS}, \texttt{MACHINES}, \phi)$;
            update `JOBS`, `MACHINES`, `TASKS` according to the output information from TaPeBB;
        end
    end
until $|\mathit{JOBS}| > 0$;

be performed for some size parameter i as a result of executing TaPeBB, without yet taking into account crashes.

Finally, there is a special and quite pessimistic case, that makes a difference between ScaTri and DefTri. Specifically, if there are not many jobs left, but they are very long, then the algorithm needs to switch to a slightly different TaPeBB phase parameter ϕ, mainly $\phi = 1$, so that transmissions are enabled in any given round. This is in order not to waste too much idle work because of the jobs being significantly long for the considered case.

Theorem 3. DefTri *performs work* $L+\mathcal{O}(\frac{L}{n}m\sqrt{n}+\alpha m\min\{f,n\})$ *against the adaptive* f*-bounded adversary.*

Proof. We analyze similar cases as in the proof of Theorem 2 wrt ScaTri, using same methodology.

- Case $|\texttt{JOBS}| \geq d$ and $|\texttt{TASKS}| \geq d(d+1)/2$.

  For this case we have a great number of jobs to perform and the only subtlety is to take care of the appropriate phase length, that is adjusted before each epoch. Similarly as in Theorem 2, productive work resulting from successful job performance accrues L units of work. If a machine crashes, its work in the epoch is at most ϕm, which is equal to the maximal length of the epoch rescaled by ϕ that is the phase parameter. Because we have f possible crashes and ϕ is always adjusted to the current average, then we may estimate the upper bound for this kind of work as $\frac{L}{n}m\min\{f,n\}$, taking into consideration that the number of jobs n may be less than the adversary's possibilities of crashes.

- Case $|\texttt{JOBS}| < d$ and $|\texttt{TASKS}| \geq d(d+1)/2$.

  In this case there are not too many jobs, but they are long. Consequently, DefTri changes the phase parameter of TaPeBB to 1 so that transmissions are available at any given time, hence the schedule of transmissions is fixed according to list `MACHINES` and `JOBS`. In this case each machine has barely one job to take care of and machines with shorter jobs and/or lower job ID's have the opportunity to confirm their jobs before machines having longer jobs scheduled. Obviously the only transmissions that make sense are those that confirm performing a particular job.

  The length of an epoch is now set to the number of jobs, unless the number of jobs is greater than the number of currently operational machines — for such case the length of the epoch is simply $M \leq m$. In this case several epochs may be needed to perform all the jobs.

  The execution for such case is likely to be consistent with the scenario where machines will work silently for some time (long jobs) and then start communicating progress one after another. Hence if now a machine performs a successful transmission, it means that it performed a job and the work accrued by this fact accounts to factor L. Otherwise, if there was a crash caused by the adversary then the maximum number of wasted work units accrued is α. While there are as many as f crash possibilities, we have that αf is upper bound on the amount of wasted work.

Finally, if there is a number of machines idling because some others are performing few long jobs, then this process generates the amount of αf work multiplied by m. We therefore conclude that the overall amount of work accrued for the considered case is $m\alpha \min\{f, n\}$, given the sub-case that the number of jobs is less than the number of crashes.

- Case $|\texttt{TASKS}| < d(d+1)/2$.

  When there are not enough jobs to provide the input to TaPeBB, then the size of a single epoch of TaPeBB is reduced in order to prevent excessive idle or redundant work. Obviously, the productive and wasted work is L and $\frac{L}{n} m \min\{f, n\}$ respectively, however we would like to estimate the idle work for the considered case.

  Likewise in the proof of Theorem 2, we use a union bound over a logarithmic number of cases for the idle kind of work. Let us assume that the length of the epoch is set to $\frac{m}{2^i}$, for some i, and there are not enough tasks to fully fill the input of TaPeBB, hence the length of the epoch is reduced with parameter $i+1$. The ratio between both of these cases is

  $$\frac{\frac{\frac{m}{2^{i+1}}(\frac{m}{2^{i+1}}+1)}{2}}{\frac{\frac{m}{2^i}(\frac{m}{2^i}+1)}{2}} = \frac{1}{4}\,\frac{m+2^{i+1}}{m+2^i} \geq \frac{1}{4}\ ,$$

  thus we conclude that of the reduced epoch is about 4 times smaller than the initial one. In what follows, the number of reduced length epochs sufficing to perform all the tasks is constant. What is more, the phase parameter will be set appropriately to the average length of the current set of tasks, hence, according to Fact 2, this will be a correct setting to fill in the input for TaPeBB.

  Taking these facts altogether and a similar reasoning as in Theorem 2 over the logarithmic number of cases, we have that

  $$cm \sum_i \frac{\sqrt{n}}{2^i} = \mathcal{O}(m\sqrt{n})$$

  phases of work are needed, where c is the constant number of epochs required for performing all the jobs. Once again: we have a sum over a logarithmic number of cases characterized by the epoch length, where at most m machines are operational, and they are working for c executions of TaPeBB, hence the estimate above.

  The final question is however how to bound the duration of all the phases from above for the idle work estimation?

  Once again, we calculate a union bound over a logarithmic number of cases. The initial arithmetic average length of a job is L/n. It is set for the time of performing at most $n/2$ jobs, because if we fell into the condition that $|\texttt{TASKS}| < d(d+1)/2$, then at most two epochs suffice to perform all the jobs. And because we change the average length of a phase after each epoch, hence we argued that the initial average is set for $n/2$ jobs.

The next step is that we fall into the case when an average of $L_2/(n/2)$ is set for at most $n/4$ jobs, and other cases follow similarly. The key observation is that the consecutive average phase durations are set for monotonically decreasing periods, where each one is set for at most half the "time" than the preceding one. Of course $L = L_1 \geq L_2 \geq \ldots L_{\log n}$. In what follows we have that the initial average length of a job dominates the weighted average of all the averages that occur over all the cases:

$$\frac{L}{n} \geq \frac{L/2 + L/4 + \cdots + L/2^{\log n}}{n} \geq \frac{L_1/2 + L_2/4 + \cdots + L_{\log n}/2^{\log n}}{n}.$$

We therefore showed that the idle work factor equals $\mathcal{O}\left(\frac{L}{n} m \sqrt{n}\right)$.

As in Theorem 2, the analysis of DefTri consists of idle, wasted and productive work. Likewise, if a job is performed and it has been confirmed, the work accounts to L. Otherwise, it accounts to $\alpha m \min\{f, L\}$. In what follows, factor L appears in the complexity formula with constant 1, similarly as it was in ScaTri analysis. We conclude that DefTri has $L + \mathcal{O}(\frac{L}{n} m \sqrt{n} + \alpha m \min\{f, L\})$ work complexity. □

5 Towards randomized solutions

We already checked that preemption is helpful when considering arbitrary lengths of jobs that need to be done by a distributed system. However, a natural question appears whether we may perform even better when randomization is available for the considered problem and its corresponding model. The answer seemed to be positive for the non-adaptive adversary, hence we investigate this issue by adjusting an algorithm from [12] and analyzing it for the configuration focusing on jobs with preemption. We provide an improved algorithm, some intuitions about it and a detailed analysis. We send the reader to [12] for any details regarding the original solution.

5.1 Algorithm RanScaTri

The name of RanScaTri follows from ScaTri and that in fact the actual job performing remains the same. What distinguishes both algorithms is the use of randomization. Precisely, RanScaTri changes the order of machines on list `MACHINES` by moving machines that performed a succesful broadcast to front of that list. This has dual significance - on one hand it prevents flexible adversarial crashes but on the other it also allows to predict to which interval $n \in (\frac{m}{2^i}, \frac{m}{2^{i-1}}]$ for $i = 1, \cdots, \lceil \log_2(m) \rceil$ does the current number of operational machines belong what helps us with the analysis.

Algorithm 3: MIX-AND-TEST, code for machine v

Input: i, L, m

`coin` $:= \frac{m}{2^i}$;
for $\sqrt{L} + \log m$ *times* **do**
  if *v has not been moved to front of list* `MACHINES` *yet* **then**
    toss a coin with the probability `coin`$^{-1}$ of heads to come up
  end
  if *heads came up in the previous step* **then**
    broadcast v via the channel and attempt to receive a message
  end
  if *some machine w was heard* **then**
    move machine w to the front of list `MACHINES`;
    decrement `coin` by 1;
  end
end
if *at least $\sqrt{L}$ broadcasts were heard* **then**
  return `true`;
end
else
  return `false`;
end

Algorithm 4: RanScaTri, code for machine v

if $m^2 \leq L$ **then**
  execute ScaTri;
end
else
  initialize `MACHINES` to a sorted list of all m names of machines;
  initialize variable $\phi := 1$ representing the length of a phase;
  if $\log_2(m) > e^{\frac{\sqrt{L}}{32}}$ **then**
    execute a L-phase epoch where every machine has all tasks assigned (without transmissions);
    execute Confirm-Work;
  end
  else
    $i = 0$;
    repeat
      if $(\frac{m}{2^i} \leq \sqrt{L})$ **then**
        Execute ScaTri;
      end
      if Mix-And-Test(i, L, m) **then**
        repeat
          Execute TaPeBB$(v, \sqrt{T}, \texttt{TASKS}, \texttt{STATIONS}, \phi)$;
        until *less than* $\frac{1}{4}\sqrt{L}$ *broadcasts are heard*;
      end
      increment i by 1;
    until $i = \lceil \log_2(m) \rceil$;
  end
end

Algorithm 5: Confirm-Work, code for machine v

```
i := 0 ;
repeat
    coin := m/2^i ;
    toss a coin with the probability coin^{-1} of heads to come up;
    if heads came up in the previous step then
        broadcast v via the channel and attempt to receive a message;
    end
    if some machine w was heard then
        clear list JOBS;
        break;
    end
    else
        increment i by 1 ;
        if i = ⌈log_2(m)⌉ + 1 then
            i := 0;
        end
    end
until a broadcast was heard;
```

Machines moved to front of list `MACHINES` are called *leaders*. Because they are chosen randomly, we expect that they will be uniformly distributed in the adversary order of crashes. This allows us to assume that a crash of a *leader* is likely to be preceeded by several crashes of other machines.

Let us consider Mix-And-Test at first. If p is the previously predicted number of operational machines, then each of the machines tosses a coin with the probability of success equal $1/p$. When only one machine did successfully broadcast it is moved to front of list `MACHINES` and the procedure starts again with a decreased parameter. Otherwise, when none or more than one of the machines broadcasts then silence is heard on the channel, as there is no collision detection. Machines that have already been moved to front do not take part in the following iterations of the procedure.

TaPeBB together with ScaTri are used as subprocedures for RanScaTri. The length parameter for TaPeBB is now set to $\sqrt{L}$, as this allows us to be sure that the total work accrued in each epoch does not exceed $m\sqrt{L}\phi$ and simultaneously it is enough to perform all the tasks in a constant number of epochs.

Before the execution the *Non-Adaptive* adversary has to choose f machines prone to crashes and declare in which rounds they will be crashed what is in fact consistent with declaring in what order those crashes may happen. As a result when some of the *leaders* are crashed, we may be approaching the case when $M \leq \sqrt{L}$ and it is possible to execute ScaTri that complexity is $L + m\alpha$ for such parameters. Otherwise the adversary spends the majority of his possible crashes and the machines may finish all the jobs without being distracted.

The design of RanScaTri is quite simple: if some specific conditions are not satisfied (see Algorithm 4 lines 1-10), then Mix-And-Text is executed which purpose is to select a number of leaders. Because they are working for $\sqrt{L}$ rounds then, if no crashes occur, all the jobs will be done in a constant number of epochs ($\sqrt{L}$ machines perform $1+\cdots+\sqrt{L} = \mathcal{O}(L)$ tasks in a single epoch). In case of crashes occurring we may need to repeat Mix-And-Test and draw a new set of leaders. Whenever $M < \sqrt{L}$ is satisfied, ScaTri is executed that work complexity is $\mathcal{O}(L + m\alpha)$ for such parameters.

We will begin the analysis od RanScaTri with the fact stating that whenever $\frac{m}{2^i} \leq \sqrt{L}$ holds, ScaTri is executed, with $\mathcal{O}(L + m\alpha)$ work.

Fact 3. *Let m be the number of operational machines, and L be the number of outstanding tasks. Then for $m \leq \sqrt{L}$ ScaTri work complexity is $\mathcal{O}(L + m\alpha)$.*

Proof. If $m \leq \sqrt{L}$, then the outstanding number of crashes is $f < m$, hence $f < \sqrt{L}$. ScaTri has $L + \mathcal{O}(m\sqrt{L} + m \min\{f, L\} + m\alpha)$ work complexity. In what follows the complexity is $L + \mathcal{O}(\sqrt{L}\sqrt{L} + \sqrt{L} \min\{\sqrt{L}, L\} + m\alpha) = \mathcal{O}(L + m\alpha)$. □

Lemma 6. *Let us assume that we have M operational machines at the beginning of an epoch, where $\sqrt{L}$ were chosen leaders. If the adversary crashes $M/2$ machines, then the probability that there were $3/4$ of the overall number of leaders crashed in this group does not exceed $e^{-\frac{1}{8}\sqrt{L}}$.*

Proof. We have M machines, among which $\sqrt{L}$ are leaders. The adversary crashes $M/2$ machines and our question is how many leaders where in this group?

The hypergeometric distribution function with parameters N - number of elements, K - number of highlighted elements, l - number of trials, k - number of successes, is given by:

$$\mathbb{P}[X = k] = \frac{\binom{K}{k}\binom{N-K}{l-k}}{\binom{N}{l}}.$$

The following tail bound from [13] tells us, that for any $t > 0$ and $p = \frac{K}{N}$:

$$\mathbb{P}[X \geq (p + t)l] \leq e^{-2t^2 l}.$$

Identifying this with our process we have that $K = M/2$, $N = m$, $l = \sqrt{L}$ and consequently $p = 1/2$. Placing $t = 1/4$ we have that

$$\mathbb{P}\left[X \geq \frac{3}{4}\sqrt{L}\right] \leq e^{-\frac{1}{8}\sqrt{L}}.$$

□

Lemma 7. *Let us assume that the number of operational machines is in $(\frac{m}{2^i}, \frac{m}{2^{i-1}}]$ interval. Then procedure Mix-And-Test(i, L, m) will return true with probability $1 - e^{-c\,(\sqrt{L}+\log_2(m))}$, for some $0 < c < 1$.*

Proof.

Claim 1. *Let the current number of operational machines be in $(\frac{x}{2}, x]$. Then the probability of an event that in a single iteration of* Mix-And-Test *exactly one machine will broadcast is at least $\frac{1}{2\sqrt{e}}$ (where the* $\texttt{coin}^{-1}$ *parameter is $\frac{1}{x}$).*

Proof. Let us consider a scenario where the number of operational machines is in $(\frac{x}{2}, x]$ for some x. If every machine broadcasts with probability of success equal $1/x$ then the probability of an event that exactly one machine will transmit is $(1-\frac{1}{x})^{x-1} \geq 1/e$. Estimating the worst case, when there are $\frac{x}{2}$ operational machines (and the probability of success remains $1/x$) we have that

$$\frac{1}{2}\left(1-\frac{1}{x}\right)^{x\cdot\frac{x-2}{2}} \geq \frac{1}{2\sqrt{e}}.$$

$\square$

According to the Claim the probability of an event that in a single round of Mix-And-Test exactly one machine will be heard is $\frac{1}{2\sqrt{e}}$.

We assume that $M \in (\frac{m}{2^i}, \frac{m}{2^{i-1}}]$. We show that the algorithm confirms appropriate i with probability $1-e^{-c\,(\sqrt{L}+\log_2 m)}$. For this purpose we need $\sqrt{L}$ transmissions to be heard.

Let X be a random variable such that $X = X_1 + \cdots + X_{\sqrt{L}+\log_2(m)}$, where $X_1, \cdots, X_{\sqrt{L}+\log_2(m)}$ are Poisson trials and

$$X_k = \begin{cases} 1 & \text{if machine broadcasted,} \\ 0 & \text{otherwise.} \end{cases}$$

We know that

$$\mu = \mathbb{E}X = \mathbb{E}X_1 + \cdots + \mathbb{E}X_{\sqrt{L}+\log_2(m)} \geq \frac{\sqrt{L}+\log_2(m)}{2\sqrt{e}}.$$

To estimate the probability that $\sqrt{L}$ transmissions were heard we will use the Chernoff's inequality.

We want to have that $(1-\epsilon)\mu = \sqrt{L}$. Thus $\epsilon = \frac{\mu-\sqrt{L}}{\mu} = \frac{\sqrt{L}+\log_2(m)-2\sqrt{e}\sqrt{L}}{\sqrt{L}+\log_2(m)}$. In what follows $0 < \epsilon^2 < 6$ i.e. it is bounded. Hence

$$\mathbb{P}[X < \sqrt{L}] \leq e^{-\frac{\left(\frac{\sqrt{L}+\log_2(m)-2\sqrt{e}\sqrt{L}}{\sqrt{L}+\log_2(m)}\right)^2}{2}\frac{\sqrt{L}+\log_2(m)}{2\sqrt{e}}} = e^{-c\,\sqrt{L}+\log_2(m)},$$

for some bounded constant c. We conclude that with probability $1-e^{-c\,\sqrt{L}+\log_2(m)}$ we shall confirm the correct i which describes and estimates the current number of operational machines. $\square$

Lemma 8. Mix-And-Test(i, L, m) *will* ***not be*** *executed if there are more than* $\frac{p}{2^{i-1}}$ *operational machines, with probability not less than* $1-(\log_2(m))^2 \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\sqrt{L}+\log_2(m)}\}$.

Proof. Let A_i denote an event that at the beginning of and execution of the Mix-And-Test(i, L, m) procedure there are no more than $\frac{m}{2^{i-1}}$ operational machines.

The basic case when $i = 0$ is trivial, because initially we have m operational machines, thus $\mathbb{P}(A_0) = 1$. Let us consider an arbitrary i. We know that

$$\mathbb{P}(A_i) = \mathbb{P}(A_i|A_{i-1})\mathbb{P}(A_{i-1}) + \mathbb{P}(A_i|A_{i-1}^c)\mathbb{P}(A_{i-1}^c) \geq \mathbb{P}(A_i|A_{i-1})\mathbb{P}(A_{i-1}).$$

Let us estimate $\mathbb{P}(A_i|A_{i-1})$. Conditioned on that event A_{i-1} holds, we know that after executing Mix-And-Test$(i-1, L, m)$ we had $\frac{m}{2^{i-2}}$ operational machines. In what follows if we are now considering Mix-And-Test(i, L, m), then we have two options:

1. Mix-And-Test$(i-1, L, m)$ returned *false*,
2. Mix-And-Test$(i-1, L, m)$ returned *true*.

Let us examine what these cases mean:

Ad 1. If the procedure returned *false* then we know from Lemma 7 that with probability $1 - e^{-c\sqrt{L}+\log_2(m)}$ there had to be no more than $\frac{m}{2^{i-1}}$ operational machines. If that number would be in $(\frac{m}{2^{i-1}}, \frac{m}{2^{i-2}}]$ then the probability of returning *false* would be less than $e^{-c\sqrt{L}+\log_2(m)}$.

Ad 2. If the procedure returned *true*, this means that when executing it with parameters $(i-1, L, m)$ we had no more than $\frac{m}{2^{i-1}}$ operational machines. Then the internal loop of RanScaTri was broken, so according to Lemma 6 we conclude that the overall number of operational machines had to reduce by half with probability at least $1 - e^{-\frac{1}{8}\sqrt{L}}$.

Consequently, we deduce that $\mathbb{P}(A_i|A_{i-1}) \geq (1 - \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\sqrt{L}+\log_2(m)}\})$. Hence $\mathbb{P}(A_i) \geq (1 - \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\sqrt{L}+\log_2(m)}\})^i$. Together with the fact, that $i \leq \log_2(m)$ and the Bernoulli inequality we have that

$$\mathbb{P}(A_i) \geq 1 - \log_2(m)\ \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\sqrt{L}+\log_2(m)}\}.$$

We conclude that the probability that the conjunction of events $A_1, \cdots, A_{\log_2(p)}$ will hold is at least

$$\mathbb{P}(A_i) \geq 1 - (\log_2(m))^2\ \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\sqrt{L}+\log_2(m)}\}.$$

□

Theorem 4. RanScaTri *performs* $\mathcal{O}(L + m\sqrt{L} + m\alpha)$ *expected work against the non-adaptive adversary.*

Proof. In the algorithm we are constantly controlling whether condition $\frac{m}{2^i} > \sqrt{L}$ holds. If not, then we execute ScaTri which complexity is $\mathcal{O}(L + m\alpha)$ for such parameters.

If this condition does not hold initially then we check another one i.e. whether $\log_2(m) > e^{\frac{\sqrt{L}}{32}}$ holds. For such configuration we assign all the tasks to every machine. The work accrued during such a procedure is $\mathcal{O}(mL)$. However when $\log_2(m) > e^{\frac{\sqrt{L}}{32}}$ then together with the fact that $e^x < x$ we have that $\log_2(m) > L$ and consequently the total complexity is $\mathcal{O}(m\log(m))$.

Finally, the successful machines, that performed all the jobs have to confirm this fact. We demand that only one machine will transmit and if this happens the algorithm terminates. The expected value of a geometric random variable lets us assume that this confirmation will happen in expected number of $\mathcal{O}(\log(m))$ rounds, generating $\mathcal{O}(m\log(m))$ work.

When none of the conditions mentioned above hold, we proceed to the main part of the algorithm. The testing procedure by Mix-And-Test for each of disjoint cases, where $M \in (\frac{m}{2^i}, \frac{m}{2^{i-1}}]$ requires a certain amount of work that can be estimated by $\mathcal{O}(m\sqrt{L} + m\log(m))$, as there are $\sqrt{L} + \log_2(m)$ testing phases in each case and at most $\frac{m}{2^i}$ machines take part in a single testing phase for a certain case.

In the algorithm we run through disjoint cases where $M \in (\frac{m}{2^i}, \frac{m}{2^{i-1}}]$. From Lemma 6 we know that when some of the leaders were crashed, then a proportional number of all the machines had to be crashed. When leaders are crashed but the number of operational machines still remains in the same interval, then the lowest number of jobs will be confirmed if only the initial segment of machines will transmit. As a result, when half of the leaders were crashed, then the system still confirms $\frac{L}{8} = \Omega(L)$ tasks. This means that even if so many crashes occurred, $\mathcal{O}(1)$ epochs still suffice to do all the jobs. Summing work over all the cases may be estimated as $\mathcal{O}(m\sqrt{L})$.

By Lemma 8 we conclude that the expected work complexity is bounded by:

$$\Big((\log(m))^2 \ \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\,(\sqrt{L}+\log(m))}\}\Big)\, \mathcal{O}(mL + m\sqrt{L}\log^2(m))$$

$$+\Big(1 - (\log(m))^2 \ \max\{e^{-\frac{1}{8}\sqrt{L}}, e^{-c\,(\sqrt{L}+\log(m))}\}\Big)\, \mathcal{O}(m\sqrt{L} + m\log(m)) = \mathcal{O}(m\sqrt{L}),$$

where the first expression comes from the fact, that if we entered the main loop of the algorithm then we know that we are in a configuration where $\log_2(m) \le e^{\frac{\sqrt{L}}{32}}$. Thus we have that

$$\frac{mL + m\sqrt{L}\log^2(m)}{e^{\frac{\sqrt{L}}{8}}} \le \frac{mL + mL\log^2(m)}{e^{\frac{\sqrt{L}}{16}} e^{\frac{\sqrt{L}}{16}}} \le \frac{m + m\log^2(m)}{e^{\frac{\sqrt{L}}{16}}} \le m + m\log(m) = \mathcal{O}(m\log(m)).$$

Altogether, we have $\mathcal{O}(L + m\alpha)$ work, resulting from reaching condition $\frac{m}{2^i} > \sqrt{L}$ and $\mathcal{O}(m\log(m))$ work resulting from reaching $\log_2(m) \le e^{\frac{\sqrt{L}}{32}}$. Additionally, there is $\mathcal{O}(L + m\sqrt{L})$ work, because of the MIX-AND-TEST procedure and the corresponding effort of the leaders, so overall the work complexity of RANSCATRI is $\mathcal{O}(L+m\sqrt{L}+m\alpha)$, what ends the proof. □

6 Comparison

In this section we carry out a brief comparison of the formulas that we managed to prove in previous sections, i.e., the upper bound for preemptive scheduling from Theorem 2 vs the lower bound in the model of jobs without preemption from Corollary 2. This comparison shows the range of dependencies between model parameters for which both models are separated, i.e., the upper bound in the model with preemption is asymptotically smaller than the lower bound in the model without preemption. We settle the scope on the, intuitively greater, bound for the model without preemption and examine how the ranges of the parameters influence the magnitude of the formulas. Let us recall both formulas:

Preemptive, upper bound: $L + \mathcal{O}(m\sqrt{L} + m\min\{f, L\} + m\alpha)$

Non-reemptive, lower bound: $L + \Omega(\frac{L}{n}m\sqrt{n} + \alpha m \min\{f, n\})$

If L is the factor that dominates the bound in the non-preemptive model, then by simple comparison to other factors we have that L also dominates the bound in the preemptive model. In what follows both formulas are asymptotically equal when the total number of tasks is appropriately large.

When $\frac{L}{n}m\sqrt{n}$ dominates the non-preemptive formula, then by a simple observation we have that $1 \leq \sqrt{\frac{L}{n}}$, i.e., the average length of a job is at least equal 1 (some unit length), and applying a square root does not effect the inequality. Thus, multiplying both sides of the inequality by $m\sqrt{L}$ gives us that $m\sqrt{L} \leq \frac{L}{n}m\sqrt{n}$. It is also easy to see that $\alpha m \min\{f, n\}$ from the non-preemptive formula is asymptotically greater than $m\min\{f, L\} + m\alpha$ from the preemptive one.

Consequently, if $\frac{L}{n}m\sqrt{n}$ or $\alpha m \min\{f, n\}$ dominates the bound in the non-preemptive formula then the non-preemptive formula is asymptotically greater than the preemptive one.

On the other hand when L dominates the bound then they are asymptotically equal as both formulas linearize for such magnitude. Such results are somehow compatible with our suspicions about the two models as intuitively the non-preemptive model for channel without collision detection is more demanding in most cases.

Note that the upper bound obtained for randomized algorithm RANSCATRI against non-adaptive adversary matches the absolute lower bound $L + \Omega(m\sqrt{L} + m\alpha)$ in this model, given in Lemma 2, and therefore beats both the compared formulas for deterministic solutions.

7 Conclusions

In this paper we addressed the problem of performing jobs of arbitrary lengths on a multiple-access channel. In particular, we investigated two scenarios. The first one was the model with preemption, where jobs can be done partially, even by different machines.

Jobs with preemption are likely to be seen as chains of tasks, that need to be done in a certain order. The natural bottleneck for such problem is therefore the longest job α. We showed a lower bound for the considered problem that is $\Omega(L+m\sqrt{L}+m\min\{f,L\}+m\alpha)$ and designed an algorithm that meets the proved bound, what makes the problem solved. Additionally we considered a distinction between oblivious and non-oblivious jobs and showed that such a distinction does not matter for the channel without collision detection and the adaptive adversary.

On the other hand we answered the question of how to deal with jobs without preemption, that need to be done in one piece in order to confirm progress. Here, as well, we showed a lower bound for the problem and developed a solution, basing on the one for preemptive jobs, which also turned out not to be optimal. The formula for work is $L+\mathcal{O}(\frac{L}{n}m\sqrt{n}+\alpha m\min\{f,n\})$, while we proved the lower bound for the problem to be $L+\Omega(\frac{L}{n}m\sqrt{n}+\frac{L}{n}m\min\{f,n\}+m\alpha)$.

What is more, even though we may hide factor L in the asymptotic notation, it still appears with constant 1 in both solutions and represent the amount of work, resulting from the number of tasks needed to perform for the considered problem.

We also showed that randomization helps in case of non-adaptive adversary, lowering the cost of preemptive scheduling to the absolute lower bound $L+\Omega(m\sqrt{L}+m\alpha)$, while it does not help against more severe adaptive adversary.

Considering the open directions for the piece of research considered in this paper, it is somehow natural to address the question of channels with collision detection. It may happen that the distinction between oblivious and non-oblivious jobs will matter when such feature is available. Furthermore it is worth considering whether randomization could help improving the formulas while considering non-preemptive setting or against adversaries of intermediate power, as it took place in similar papers considering the Do-All problem on shared channel. Finally, the primary goal of this work was to translate scheduling from classic models to the model of a shared channel, in which it was not considered in depth; therefore, a natural open direction is to extend the model by other features considered in the scheduling literature.

References

[1] Bogdan S Chlebus. *Randomized communication in radio networks, a chapter.* Kluwer Academic Publisher, Drodrecht, 2001.

[2] Bogdan S. Chlebus, Roberto De Prisco, and Alex A. Shvartsman. Performing tasks on synchronous restartable message-passing processors. *Distrib. Comput.*, 14(1):49–64, January 2001.

[3] Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Alexander A. Shvartsman. Bounding work and communication in robust cooperative computation. In *Proceedings of the 16th International Conference on Distributed Computing*, DISC '02, pages 295–310, London, UK, UK, 2002. Springer-Verlag.

[4] Bogdan S. Chlebus and Dariusz R. Kowalski. Randomization helps to perform independent tasks reliably. *Random Structures and Algorithms*, 24(1):11–41, 2004.

[5] Bogdan S. Chlebus, Dariusz R. Kowalski, and Andrzej Lingas. Performing work in broadcast networks. *Distributed Computing*, 18(6):435–451, 2006.

[6] Roberto De Prisco, Alain Mayer, and Moti Yung. Time-optimal message-efficient work performance in the presence of faults. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 161–172, New York, NY, USA, 1994. ACM.

[7] Cynthia Dwork, Joseph Y. Halpern, and Orli Waarts. Performing work efficiently in the presence of faults. *SIAM J. Comput.*, 27(5):1457–1491, 1998.

[8] Z. Galil, A. Mayer, and Moti Yung. Resolving message complexity of byzantine agreement and beyond. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, page 724, Washington, DC, USA, 1995. IEEE Computer Society.

[9] Robert G. Gallager. A perspective on multiaccess channels. *IEEE Trans. Information Theory*, 31:124–142, 1985.

[10] Chryssis Georgiou, Dariusz R. Kowalski, and Alexander A. Shvartsman. Efficient gossip and robust distributed computation. *Theor. Comput. Sci.*, 347(1-2):130–166, November 2005.

[11] Chryssis Georgiou and Alexander A. Shvartsman. *Cooperative Task-Oriented Computing: Algorithms and Complexity.* Morgan & Claypool Publishers, 2011.

[12] Marek Klonowski, Dariusz R. Kowalski, and Jarosław Mirek. Ordered and delayed adversaries and how to work against them on shared channel. *CoRR*, abs/1706.08366, 2017.

[13] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *Journal American Statistics Association*, vol. 58, pages 13-30, 1963.

[14] Dariusz R. Kowalski and Alex A. Shvartsman. Performing work with asynchronous processors: Message-delay-sensitive bounds. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 265–274, New York, NY, USA, 2003. ACM.

[15] Dariusz R. Kowalski, Prudence W. H. Wong, and Elli Zavou. Fault tolerant scheduling of tasks of two sizes under resource augmentation. *J. Scheduling*, To appear (`http://dx.doi.org/10.1007/s10951-017-0541-1`).

[16] Giorgio Lucarelli, Abhinav Srivastav, and Denis Trystram. From preemptive to non-preemptive scheduling using rejections. In *Preceedings of the 22nd International Conference on Computing and Combinatorics*, COCOON 2016, pages 510–519, Ho Chi Minh City, Vietnam, 2016.

[17] Robert McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6(1):1–12, October 1959.