Competitive Analysis of Scheduling Algorithms for Aggregated Links

Wojciech Jawor* Marek Chrobak* Christoph Dürr†

Abstract

We study an online job scheduling problem arising in networks with aggregated links. The goal is to schedule n jobs, divided into k disjoint chains, on m identical machines, without preemption, so that the jobs within each chain complete in the order of release times and the maximum flow time is minimized.

We present a deterministic online algorithm Block with competitive ratio $O(\sqrt{n/m})$, and show a matching lower bound, even for randomized algorithms. The performance bound for Block we derive in the paper is, in fact, more subtle than a standard competitive ratio bound, and it shows that in overload conditions (when many jobs are released in a short amount of time), Block's performance is close to the optimum.

We also show how to compute an offline solution efficiently for $k = 1$, and that minimizing the maximum flow time for $k, m \geq 2$ is $\mathcal{NP}$-hard. As by-products of our method, we obtain two offline polynomial-time algorithms for minimizing makespan: an optimal algorithm for $k = 1$, and a 2-approximation algorithm for any k.

1 Introduction

Link Aggregation, or trunking, is a method of grouping physical link segments (channels) between two network devices into a single logical link. The technology can be used to scale the bandwidth between the two devices, provide load balancing and improve system's fault-tolerance. Since Link Aggregation is also cost effective (it is often cheaper to add an additional channel to an existing link, then to replace the link with one of higher capacity) it is becoming very popular: In a survey [6] of 38 major Internet Service Providers (ISPs) conducted in mid 1997, only two (smaller) ISPs did not have parallel links between nodes.

The traffic at the source of a multi-channel system (a network in which at least two nodes are interconnected with parallel links) is often divided into disjoint *conversations*, where a conversation is a distinguishable source-destination pair. In the system every output of a node is equipped with a scheduler. The scheduler receives packets from conversations that traverse the node, and determines the order and times of their transmissions over the output channels.

Since packets of an individual conversation transmitted between two immediate nodes may be serviced concurrently by more than one channel, the order in which the data packets arrive at the receiver (i.e., the order in which the last bits of packets arrive) may be different from the order in which

*Department of Computer Science, University of California, Riverside, CA 92521. Supported by NSF grants OISE-0340752 and CCR-0208856. {marek,wojtek}@cs.ucr.edu

†CNRS, Laboratoire d'Informatique de l'Ecole Polytechnique, 91128 Palaiseau, France. Work conducted while being affiliated with the Laboratoire de Recherche en Informatique, Université Paris-Sud, 91405 Orsay. Supported by the CNRS/NSF grant 17171 and ANR Alpage.

they originally arrived at the sender, even if they are transmitted by the sender in the order of their arrivals. Bennett *et al.* [2] argue that the parallelism in Internet components and links is one of the main reasons of packet reordering, contrary to the common belief that reordering is caused by "pathological" behavior (i.e., by incorrect or malfunctioning network components). The inter-conversation reordering is an important issue for multi-channel systems, as it may have a direct bearing on the performance of the transmission control protocol [2], and thus on the performance of the whole system.

The IEEE 802.3 standard [9] defines the protocol used to communicate link aggregation information between network devices in Local Area Networks. The standard does not specify the algorithm for choosing a channel used to transmit any given packet but assumes that the algorithm neither reorders nor duplicates the packets. In practice, the requirement of maintaining packet ordering is met by ensuring that all packets that compose a given conversation are transmitted on a single channel. The distribution is achieved by using a hash function. This approach has several drawbacks: First, it does not fully utilize the capacity of a link if the number of conversations is smaller than the number of channels. Second, such an algorithm does not provide load balancing, i.e., if traffic increases beyond a single channel's bandwidth, it is not distributed among additional channels. Third, it is hard or even not possible to design a hash function that would distribute the traffic well in all situations. And finally, in some (common) configurations the link aggregation algorithm (which is a part of the Link Layer) must violate the layered architecture of network protocols by accessing higher layer information in order to compute useful hash functions.

Problem formulation. The above discussion raises the problem of designing appropriate scheduling mechanisms which could be used to guide the packet transmission. The goal is to optimize link utilization, under the constraint that packets complete their arrivals at the receiver in the order of their arrivals at the sender. Formally, using scheduling terminology, we state this problem as follows: We are given n jobs (packets) organized in k disjoint chains (conversations), with each job j specified by a triple (r_j, p_j, z_j) where r_j is a positive release (arrival) time, p_j is the processing time, or length of the job, and $1 \leq z_j \leq k$ is an index of the chain to which job j belongs. We assume that $\min_j r_j = 0$. In addition, the jobs are ordered so that if a job j precedes j' (we simply write $j < j'$) then $r_j \leq r_{j'}$. The ordering of job indices within a chain represents the ordering of packets in a conversation. We assume that at the node where scheduling takes places the packets arrived in a correct order, which justifies the ordering of the release times.

The jobs need to be scheduled on m identical machines (channels) and must satisfy the FRFC (**f**irst **r**eleased **f**irst **c**ompleted) order, i.e., if for two jobs $j < j'$ and $z_j = z_{j'}$ then $C_j^{\mathcal{A}} \leq C_{j'}^{\mathcal{A}}$, where $C_j^{\mathcal{A}}$ is the completion time (the time at which the whole packet arrives at the receiver) of job j in schedule $\mathcal{A}$. Preemption is not allowed (since packet transmissions cannot be interrupted).

We would like to note that the above ordering is fixed even for jobs with a common release time. So even these jobs must complete in the final schedule in this predefined order.

In addition to the above constraints we also want to optimize the machine (link) utilization. In this paper we aim at constructing schedules which minimize the *maximum flow time* $F_{\max}^{\mathcal{A}} = \max_j F_j^{\mathcal{A}}$ of jobs in schedule $\mathcal{A}$, where $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$ denotes the flow time of job j. The use of this function is motivated by *Quality of Service* applications in which, in order to provide the delay guarantees, an upper bound on the time *each* job spends in the system must be given. This helps avoid undesirable starvation issues. In the process, we will also construct schedules $\mathcal{A}$ to minimize maximum completion time, or *makespan*, $C_{\max}^{\mathcal{A}} = \max_j C_j^{\mathcal{A}}$.

It is natural to require that the scheduling algorithms for this problem be *online*. In the online version of the problem jobs arrive at their release times, and the algorithm makes scheduling decisions

(whether to schedule a job, and if so, which job and on which processor) without the knowledge about the jobs that will arrive in the future. Once a job arrives, the algorithm gets to know its processing time and chain index. If the value of the objective function on a schedule produced by an online algorithm A is at most c times the value of an optimum schedule of the same instance, then we say that A is *c-competitive*. The smallest such value c is called the *competitive ratio* of A. If A is a randomized online algorithm, then the same definitions apply, except that we replace the value of the objective function with its expected value. The competitive ratio is a commonly used performance measure for online algorithms, and we adopt this measure in this paper.

Our results. We give an online deterministic algorithm Block and prove that on any instance I it produces a schedule $\mathcal{B}$ with maximum flow time

$$F^{\mathcal{B}}_{\max} \leq 8\sqrt{(n/m)p_{\text{avg}}F^*_{\max}(I)} + 4F^*_{\max}(I),$$

where n is the number of jobs, $p_{\text{avg}} = \sum_j p_j/n$ is the average processing time, and $F^*_{\max}(I)$ is the maximum flow time of an optimum schedule of I. Since $F^*_{\max}(I) \geq (n/m)p_{\text{avg}}$, this implies that Block's competitive ratio is at most $12\sqrt{n/m}$.

Observe that the quantity $(n/m)p_{\text{avg}}$ in the upper bound represents an average processor load. This implies that in overload conditions, namely when many jobs are released in a short amount of time – and thus the value of the optimum flow time is close to $p_{\text{avg}}(n/m)$ – the cost incurred by Block is no more than a constant times the optimum cost. Therefore our performance bound for Block is stronger than what can be captured by classical competitive analysis.

We then show that for $m \geq 2$ there is no online randomized algorithm with expected competitive ratio better than $O(\sqrt{n/m})$, even for instances with maximum processing time $p_{\max} = 3$. This means that Block is optimally competitive up to a constant factor, and that the asymptotic guarantee given by Block cannot be improved with randomization.

In addition, we include a series of simpler results about offline algorithms: We show that for $k \geq 2$ and $m \geq 2$ minimizing maximum flow time or makespan in our model is $\mathcal{NP}$-hard, and that for $k = 1$ it is possible to compute optimum solutions in polynomial time. (Recall that k is the number of conversations.) As a by-product of our construction we also obtain an efficient 2-approximation algorithm for minimizing makespan.

Our main goal in this research was to study the theoretical properties of the new FRFC scheduling constraint in the context of link aggregation. The worst-case competitive analysis of the algorithm Block, presented in this paper, is likely to be overly pessimistic, as it does not take into account properties of the real network traffic. Indeed, as described in a subsequent experimental work [11], a slightly refined version of Block actually outperforms the standard hashing method on real network traces.

Past work. Scheduling to minimize various functions of jobs' flow times has recently received a lot of attention. Most works, however, focus on minimizing the total flow time (i.e., $\sum_j F^{\mathcal{A}}_j$), and usually in the preemptive setting. (For recent surveys on online scheduling see [12, 10].)

To the best of our knowledge we are the first to introduce the FRFC model. Therefore, in this section we briefly review only the results for related models in which there are no additional restrictions on the order of completion times.

In the context of online job scheduling with release times, the maximum flow time objective function $F_{\max}$ was first considered by Bender *et al.* [1] who showed a deterministic lower bound of $\frac{4}{3}$

for $m = 2$. They also show that the FIFO algorithm is $(3 - 2/m)$-competitive. (The FIFO algorithm always schedules the job with the earliest release time on the next available machine, and it does not necessarily create schedules which obey the FRFC order.)

Feuerstein *et al.* [5] study online scheduling problems with jobs organized in a number of sequences called *threads*. Each job becomes available as soon as a scheduling decision has been made on all preceding jobs in the same thread. They show that the algorithm List [8] (a.k.a. Graham's algorithm), which always schedules the next job on the least loaded machine, is the best possible algorithm for the makespan problem if the number of machines does not exceed the number of threads by more than 1.

When the objective is to minimize makespan, Shmoys *et al.* [13] give a general online algorithm which uses an offline algorithm for the problem as a subroutine. They show that if the offline algorithm is a c-approximation algorithm, then their online algorithm has competitive ratio at most $2c$.

There is also some literature on *fair queuing* in multi-channel systems, which has some connection to the problems discussed here. See, for example, [4] and the references therein.

2 Preliminaries

FRFC scheduling. We are given m identical machines and n jobs, with each job j specified by a triple (r_j, p_j, z_j) of integers, where $r_j \geq 0$ is the release time, $p_j \geq 1$ is the processing time, or length, of job j, and z_j, $1 \leq z_j \leq k$, is an index of the chain to which job j belongs. The jobs are assumed to be ordered so that for any two jobs $j < j'$ we have $r_j \leq r_{j'}$. Throughout the paper, $p_{\text{avg}} = (\sum_j p_j)/n$ and $p_{\max} = \max_j p_j$ denote, respectively, the average and maximum processing times in the instance.

A schedule $\mathcal{A}$ specifies when and where jobs are executed, i.e., for each job j it specifies a machine and an interval $[S_j^{\mathcal{A}}, C_j^{\mathcal{A}})$, such that (i) $S_j^{\mathcal{A}} \geq r_j$, (ii) $C_j^{\mathcal{A}} - S_j^{\mathcal{A}} = p_j$, (iii) no two jobs overlap, that is if two jobs $i \neq j$ are assigned to the same machine then either $C_i^{\mathcal{A}} \leq S_j^{\mathcal{A}}$ or $C_j^{\mathcal{A}} \leq S_i^{\mathcal{A}}$, and (iv) for any two jobs $j < j'$ from the same chain (with $z_j = z_{j'}$), we have $C_j^{\mathcal{A}} \leq C_{j'}^{\mathcal{A}}$. Of course, all jobs must be scheduled. As usual, $S_j^{\mathcal{A}}$ is called the start time of job j, and $C_j^{\mathcal{A}}$ is its completion time. Whenever the condition (iv) is satisfied we say that jobs are scheduled in FRFC order. If $t \in [S_j^{\mathcal{A}}, C_j^{\mathcal{A}})$, then we say that job j is *running* at time t in $\mathcal{A}$. Let $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$ denote the *flow time* of a job j in schedule $\mathcal{A}$.

For any schedule $\mathcal{A}$, its maximum flow time is $F_{\max}^{\mathcal{A}} = \max_j F_j^{\mathcal{A}}$, and by $C_{\max}^{\mathcal{A}} = \max_j C_j^{\mathcal{A}}$ we denote the makespan of $\mathcal{A}$. Similarly, by $F_{\max}^*(I)$ we denote the maximum flow time of an offline schedule of I which minimizes maximum flow time, and by $C_{\max}^*(I)$ the makespan of an offline schedule of I which minimizes makespan.

Online Algorithms. An algorithm A is called *online* if at each time t it decides which job to execute (if any) based only on the jobs released before or at time t.

Let $|\mathcal{A}|$ denote the value of the objective function on the schedule $\mathcal{A}$. (Recall that in this paper we consider two different objective functions, i.e., maximum flow time and makespan.) Let $\mathsf{A}(I)$ denote the schedule computed by A on I. We say that an algorithm A is *c-competitive* if $|\mathsf{A}(I)| \leq c|\mathcal{B}|$ for any schedule $\mathcal{B}$ of I. The smallest such value c is called the *competitive ratio* of A. If A is a randomized algorithm, then the same definition applies, except that $|\mathsf{A}(I)|$ is replaced with the expected value of the objective function on the schedule produced by A on I, where the expectation is taken over all random choices of the algorithm A.

```
/* initialize */
for i ← 1 to m do
    start_i ← e_n

/* create auxiliary schedule X */
for j ← n downto 1 do
    l ← argmax{start_i : 1 ≤ i ≤ m}
    schedule j on machine l at time S_j^X ← min{e_j, start_l} − p_j
    start_l ← S_j^X

/* construct the final schedule A */
δ ← max_j {r_j − S_j^X}
for j ← 1 to n do
    S_j^A ← S_j^X + δ
```

Figure 1: Algorithm RGreedy

3 Offline Algorithms

In this section we show two offline algorithms RList and RFlow. The algorithm RList minimizes makespan for $k = 1$, and the algorithm RFlow minimizes maximum flow time for $k = 1$. (Recall that k denotes the number of chains.) We also show how to use the algorithm RList to obtain a 2-approximation algorithm for minimizing makespan for $k \geq 2$.

3.1 Computing Optimal Solutions for $k = 1$

In order to derive the algorithms that compute optimal schedules for $k = 1$, we consider a more general objective function, which includes both makespan and maximum flow as special cases. Consider an instance I and suppose that for each job $j \in I$ we define a *reference point*, denoted e_j, such that for any two jobs $j < j'$ their reference points satisfy $e_j \leq e_{j'}$. We show an algorithm RGreedy which, when given an input instance I and a reference point for each job, computes a schedule $\mathcal{A}$ that minimizes $\max_j \{C_j^{\mathcal{A}} - e_j\}$. Observe that if we define $e_j = 0$ for all j, then the schedule $\mathcal{A}$ produced by RGreedy satisfies $C_{\max}^{\mathcal{A}} = C_{\max}^*(I)$; likewise, if we set $e_j = r_j$, then $F_{\max}^{\mathcal{A}} = F_{\max}^*(I)$.

The algorithm RGreedy is shown in Figure 1. It first computes an auxiliary schedule $\mathcal{X}$, ignoring the jobs' release times, and then shifts this schedule to the right to meet all release times. RGreedy can be easily implemented in time $O(n \log n)$.

Theorem 3.1 *For $k = 1$, Algorithm* RGreedy *computes a schedule that minimizes* $\max_j \{C_j^{\mathcal{A}} - e_j\}$.

Proof: We first prove that the schedule computed by RGreedy, denoted $\mathcal{A}$, is feasible. Let $\mathcal{X}$ denote the auxiliary schedule constructed in the second loop. Consider a fixed iteration j of this loop. We make two observations. First, the job j scheduled in this iteration completes at time $\min\{e_j, \max_{i=1\ldots m} start_i\}$. Second, $start_i$ is equal to the minimum starting time of jobs scheduled on machine i in the previous iterations, or to e_n if no jobs are yet scheduled on i. Thus, as j decreases from n to 1, both the reference points e_j and the values of $\max_{i=1,\ldots,m} start_i$ decrease. This implies that $\mathcal{X}$ is an FRFC schedule. Shifting the jobs in $\mathcal{X}$ by δ in the last loop guarantees that all release times are met. As the shifting does not change the order of completion times, the schedule $\mathcal{A}$ is feasible.

Let $\mathcal{O}$ be any feasible schedule of jobs $1, 2, \ldots, n$ and let $\mathcal{Y}$ be a copy of $\mathcal{O}$ in which all jobs are shifted to the left by $\max_i \{C_i^{\mathcal{O}} - e_i\}$, i.e., $S_j^{\mathcal{Y}} = S_j^{\mathcal{O}} - \max_i \{C_i^{\mathcal{O}} - e_i\}$ for all $j = 1, 2, \ldots, n$. Observe that $C_j^{\mathcal{Y}} \leq e_j$ for all jobs j.

Lemma 3.2 *For any $j = 1, 2, \ldots, n$, we have $C_j^{\mathcal{Y}} \leq C_j^{\mathcal{X}}$.*

Proof: The proof of the lemma is by induction on $j = n, n-1, \ldots, 1$. For $j = n$ we have $C_n^{\mathcal{X}} = e_n \geq C_n^{\mathcal{Y}}$, so the lemma holds. Now suppose that the lemma holds for $j = n, n-1, \ldots, i+1$, where $i < n$. We show that it also holds for $j = i$.

If $C_i^{\mathcal{X}} = e_i$ then the lemma holds since $C_i^{\mathcal{Y}} \leq e_i$, so let us assume that $C_i^{\mathcal{X}} < e_i$. Let $D \subseteq \{i+1, i+2, \ldots, n\}$ be the set of jobs which are running at $C_i^{\mathcal{X}}$ in $\mathcal{X}$. Since $C_i^{\mathcal{X}} < e_i$ we have $|D| = m$ by the definition of RGreedy. We consider two cases.

If in $\mathcal{Y}$ all jobs from D are scheduled on different machines then $C_i^{\mathcal{Y}} \leq \max_{h \in D} \{C_h^{\mathcal{Y}} - p_h\} \leq \max_{h \in D} \{C_h^{\mathcal{X}} - p_h\} = C_i^{\mathcal{X}}$, where the second inequality follows from the inductive assumption, and the equality from the definition of RGreedy and the condition $C_i^{\mathcal{X}} < e_i$.

If there are two jobs $g < f$ such that $g, f \in D$ and both are scheduled on the same machine in $\mathcal{Y}$, then $C_i^{\mathcal{Y}} \leq C_g^{\mathcal{Y}} \leq C_f^{\mathcal{Y}} - p_f \leq C_f^{\mathcal{X}} - p_f \leq \max_{h \in D} \{C_h^{\mathcal{X}} - p_h\} = C_i^{\mathcal{X}}$, where the first inequality follows from the FRFC assumption, the third inequality from the inductive assumption, and the equality from the definition of RGreedy and the condition $C_i^{\mathcal{X}} < e_i$. This completes the proof of Lemma 3.2. □

Continuing with the proof of the theorem, observe that $\max_j \{C_j^{\mathcal{A}} - e_j\} = \delta + \max_j \{C_j^{\mathcal{X}} - e_j\} = \delta$, because $C_j^{\mathcal{X}} - e_j \leq 0$ for all $j = 1, 2, \ldots, n$, and $C_n^{\mathcal{X}} = e_n$. Let h be a job in $\mathcal{X}$ such that $r_h - S_h^{\mathcal{X}} = \delta$. We have $\max_j \{C_j^{\mathcal{A}} - e_j\} = \delta = r_h - S_h^{\mathcal{X}} = r_h + p_h - C_h^{\mathcal{X}} \leq r_h + p_h - C_h^{\mathcal{Y}}$, where the last inequality follows from Lemma 3.2. By the definition of $\mathcal{Y}$, we also have $\max_j \{C_j^{\mathcal{O}} - e_j\} = S_h^{\mathcal{O}} - S_h^{\mathcal{Y}} = C_h^{\mathcal{O}} - C_h^{\mathcal{Y}} \geq r_h + p_h - C_h^{\mathcal{Y}}$. Putting these two bounds together, we get $\max_j \{C_j^{\mathcal{A}} - e_j\} \leq r_h + p_h - C_h^{\mathcal{Y}} \leq \max_j \{C_j^{\mathcal{O}} - e_j\}$, which completes the proof of Theorem 3.1. □

Throughout the rest of the paper, let RList denote Algorithm RGreedy when $e_j = 0$ for all $j \in I$, and let RFlow denote the same algorithm with $e_j = r_j$ for all $j \in I$.

Example. Let us now illustrate the difference between RFlow and RList on a simple example. Consider an instance containing the following jobs: job 1 = (0, 1), job 2 = (0, 2), job 3 = (0, 4), job 4 = (0, 1), and job 5 = (2, 2) (since all jobs belong to the same chain we only give job numbers, release times, and processing times in the form $j = (r_j, p_j)$.) We assume that $m = 2$.

Algorithm RFlow schedules jobs $3, 5$ on one machine at times $0, 4$, and jobs $1, 2, 4$ on the other machine at times $0, 1, 3$ obtaining a schedule with maximum flow time equal 4 and makespan 6. RList schedules $1, 2, 5$ on one machine at times $0, 1, 3$, and $3, 4$ on the other machine at times $0, 4$ obtaining a schedule with maximum flow time and makespan equal 5. (See Figure 2.) It is easy to verify that any algorithm that minimizes the maximum flow time on the above instance, cannot minimize the makespan, and vice versa.

3.2 A Note on Approximating Makespan

For an instance I, let $\widetilde{I}$ denote a copy of I in which all jobs are assigned to the same chain. Consider an algorithm RList-M, which on any instance I returns the schedule produced by RList on $\widetilde{I}$. Recall that $p_{\max}$ denotes the maximum processing time.

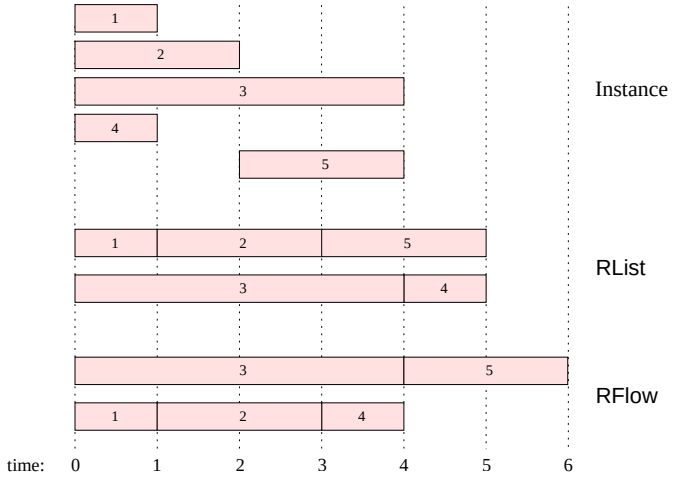

Figure 2: A comparison of RList and RFlow. The input instance is shown on the top, with each job j represented by a rectangle of length p_j starting at r_j.

Theorem 3.3 *Let $\mathcal{A}$ be the schedule produced by* RList-M *on some instance I. Then $\mathcal{A}$ is a feasible FRFC schedule of I, and $C^{\mathcal{A}}_{\max} \leq C^*_{\max}(I) + p_{\max}$, where $p_{\max}$ is the maximum processing time of jobs in I.*

The following proof resembles the analysis of the algorithm List from [8]. We present it for completeness.

Proof: We first observe that the schedule produced by RList-M is feasible. This follows from the definition of the instance $\widetilde{I}$ and the algorithm RList.

We now prove that $C^{\mathcal{A}}_{\max} \leq C^*_{\max}(I) + p_{\max}$. Let j be the last job, according to the $<$-order, such that $S^{\mathcal{A}}_j = r_j$. Consider the set of jobs $J = \{j' \in I : j' > j\}$. Clearly, for each $j' \in J$ we have $r_{j'} \geq r_j$, therefore these jobs must be started not earlier than r_j in any schedule. Let L_v denote the load of a machine v in $\mathcal{A}$ restricted to the set J, that is, the sum of the processing times of jobs from J executed on v in $\mathcal{A}$. Let $x = \operatorname{argmin}_v L_v$. By the definition of RList-M, job j is executed on machine x.

By the choice of j we have

$$\begin{aligned} C^{\mathcal{A}}_{\max} &= r_j + L_x + p_j \\ &\leq r_j + L_x + p_{\max}. \end{aligned} \tag{1}$$

As for each machine v we have $L_v \geq L_x$, the total load of the jobs in J is at least mL_x. Therefore

$$C^*_{\max}(I) \geq r_j + L_x. \tag{2}$$

The theorem follows now from (1) and (2). □

Since $C^*_{\max} \geq p_{\max}$, the above theorem yields the following corollary:

Corollary 3.4 *Algorithm* RList-M *is a 2-approximation algorithm for minimizing makespan.*

3.3 Computing Optimal Solutions for $k \geq 2$ is Hard

We conclude this section by proving that it is not possible to compute optimal solutions in polynomial time for $k \geq 2$ unless $\mathcal{P} = \mathcal{NP}$.

Theorem 3.5 *For $k \geq 2$ and $m \geq 2$, minimizing maximum flow time or makespan is $\mathcal{NP}$-hard, even if all jobs are released at time 0.*

Proof: In the proof we focus on the case when the objective is to minimize the makespan. However, since all release times are equal 0, the value of the makespan equals the maximum flow time, so the very same construction works in the case when the objective is to minimize the maximum flow time.

We consider the decision version of the problem, where we ask if there exists a schedule with makespan at most C. The optimization problem can be solved using the decision problem and binary search.

The proof is by reduction from an $\mathcal{NP}$-complete problem called Partition [7], defined as follows:

Instance: A set of positive integers $\{a_1, a_2, \ldots, a_l\}$.
Query: Does there exist a set $X \subseteq \{1, 2, \ldots, l\}$ such that $\sum_{i \in X} a_i = \frac{1}{2} \sum_{i=1}^{l} a_i$?

Consider an instance I of Partition as above. Let $B = \sum_{i=1}^{l} a_i$. We create the following instance J of the scheduling problem. We set the number of chains $k = 2$. The first chain contains l jobs with processing times $p_i = B$ for $i = 1, 2, \ldots, l$. The second chain contains $2l + m - 2$ jobs indexed $l + 1, l + 2, \ldots, 3l + m - 2$. The processing times of these jobs are defined in the following way: for $i = l + 1, l + 2, \ldots, 3l$, $p_i = B$ if $i - l$ is odd, $p_i = a_{(i-l)/2}$ if $i - l$ is even, and $p_i = (l + \frac{1}{2})B$ for $i = 3l + 1, 3l + 2, \ldots, 3l + m - 2$. The jobs with processing time B will be called the *blocking* jobs, the jobs with processing time less than B *non-blocking*, and the jobs with processing time $(l + \frac{1}{2})B$ *extra* jobs. All release times are equal 0.

Clearly, the reduction takes polynomial time. We claim that there exists an m-machine schedule of J with makespan at most $(l + \frac{1}{2})B$, if and only if there exists a set X such that $\sum_{i \in X} a_i = \frac{1}{2}B$.

($\Rightarrow$) We show that if there exists a schedule $\mathcal{S}$ with makespan at most $(l + \frac{1}{2})B$, then there exists a solution to Partition. Note that in the schedule $\mathcal{S}$ at most l blocking jobs must be executed on each machine, since otherwise it is not possible to obtain a schedule with makespan strictly lower than $(l + 1)B$. Furthermore, each extra job must be scheduled at time 0, and no other jobs may be scheduled on machines occupied by extra jobs. Therefore, the non-blocking jobs must be executed on at most 2 machines, and the total processing time of the non-blocking jobs on each of these machines must be exactly $\frac{1}{2}B$. So, by using the partitioning of non-blocking jobs between the machines it is possible to construct a solution to I.

($\Leftarrow$) Now we show that if there exists a solution to I then there exists a schedule $\mathcal{S}$ with makespan at most $(l + \frac{1}{2})B$. To construct the schedule $\mathcal{S}$, we first construct partial schedules $\mathcal{S}_i$ for $i = 1, 2, \ldots, l$. Each schedule $\mathcal{S}_i$ will be obtained by appending three additional jobs to $\mathcal{S}_{i-1}$ on the first two machines. The final schedule $\mathcal{S}$ will be obtained from $\mathcal{S}_l$ by adding extra jobs to machines $3, \ldots, m$. We assume that $\mathcal{S}_0$ is an empty schedule.

The schedule $\mathcal{S}_i$ is obtained from $\mathcal{S}_{i-1}$, by scheduling new jobs at the completion time of the first or second machine, in the following way: If $i \in X$ then we schedule the job i on the first machine, and jobs $l + 2i - 1, l + 2i$ (in this order) on the second machine. If $i \notin X$ then we schedule the job i on the

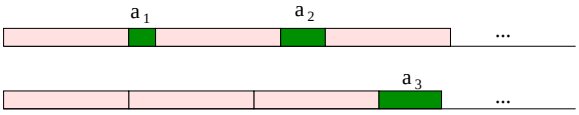

Figure 3: An illustration of the reduction. The figure shows schedule $\mathcal{S}_3$, where $1, 2 \notin X$ and $3 \in X$. Light rectangles represent blocking jobs and dark rectangles represent non-blocking jobs. The top machine is machine 1, the bottom one is machine 2. Extra jobs, scheduled at machines $3, 4, ..., m$, are not shown.

second machine, and jobs $l + 2i - 1, l + 2i$ (in this order) on the first machine. Observe that the first and second machine of $\mathcal{S}_i$ execute the same number of blocking jobs, for each $i = 1, 2, \ldots, l$.

In $\mathcal{S}$ exactly l blocking jobs are scheduled on the first and on the second machine, and each extra job is scheduled on a machine of its own. Since $\sum_{i \in X} a_i = \frac{1}{2}B$, by assumption, each machine processes jobs with total processing time at most $(l + \frac{1}{2})B$. Therefore, to complete the proof it is enough to show that the schedule $\mathcal{S}$ is feasible.

To prove that $\mathcal{S}$ is feasible, we show that each schedule $\mathcal{S}_i$ is a feasible partial schedule. This is done by induction on i.

First observe that in $\mathcal{S}_1$ each chain is entirely executed on one machine, and for each chain, the jobs are started in the order of their indices. Therefore, $\mathcal{S}_1$ is feasible.

Now suppose that the schedule $\mathcal{S}_{i-1}$ is feasible. We claim that schedule $\mathcal{S}_i$ is also feasible. To see this observe that the difference in the total processing time of jobs scheduled on the first and second machine in $\mathcal{S}_{i-1}$ is smaller than B. This follows from the fact that both of these machines execute the same number of blocking jobs. Therefore, the difference in the total processing time is equal to the difference of total processing time of non-blocking jobs scheduled on these machines, and this value must be smaller than B. Three new jobs are added to $\mathcal{S}_{i-1}$: one blocking job from chain 1, and one blocking and one non-blocking from chain 2. Note that after adding the blocking jobs each of them completes in $\mathcal{S}_i$ at time at least $C_{\max}^{\mathcal{S}_{i-1}}$, therefore, the schedule remains feasible. The last job, which is non-blocking, is scheduled on the same machine as the previous job from the same chain, so $\mathcal{S}_i$ is feasible.

This completes the proof of feasibility of $\mathcal{S}_l$. As the makespan of this schedule is at most $(l + \frac{1}{2})B$, which is equal to the length of the extra jobs, adding the extra jobs, each on a separate (initially idle) machine, does not violate feasibility. Therefore $\mathcal{S}$ is feasible. This completes the proof. □

4 Online Algorithms

In this section we give and analyze our online algorithm Block. We also prove that there is no randomized algorithm with expected asymptotic competitive ratio better than $O(\sqrt{n/m})$, even for instances with $p_{\max} = 3$.

4.1 An Upper Bound on the Competitive Ratio

Let A be any offline algorithm for minimizing makespan. We first show how to use algorithm A to create an online algorithm Block[A].

Algorithm Block[A]**:** The algorithm proceeds in phases numbered $1, 2, 3, \ldots$, where phase i starts at time β_i. First, let $\beta_1 = 0$. Consider phase i, and let Q_i be the set of jobs pending at time β_i. We

apply algorithm A to schedule all jobs in Q_i. Suppose that the last job from Q_i completes at time $\beta_i + \delta_i$. Then let β_{i+1} be the first time not earlier than $\beta_i + \delta_i$ such that there is at least one pending job. (If no more jobs arrive, the computation completes.)

The above technique of applying an algorithm A to create an online algorithm Block[A] was first proposed by Shmoys *et al.* [13], who prove that if algorithm A produces a schedule with makespan at most $\rho C^*_{\max}(I)$ on any instance I, then algorithm Block[A] produces a schedule with makespan at most $2\rho C^*_{\max}(I')$ on any instance I'. Note that if A is an FRFC algorithm then so is Block[A].

In the previous section we introduced the offline algorithm RList for minimizing makespan for $k = 1$, and a 2-approximation algorithm RList-M for any $k \geq 2$. (RList-M simply runs RList treating all jobs as belonging to the same chain.) From the previous paragraph, we have the following corollary:

Corollary 4.1 Block[RList] *is a 2-competitive FRFC algorithm for minimizing makespan for $k = 1$, and* Block[RList-M] *is a 4-competitive FRFC algorithm to minimize makespan for any $k \geq 1$.*

From now on we will denote Block[RList-M] simply by Block. We stress here that Block schedules each set Q_i using the offline algorithm for *makespan*. Nevertheless, in this section we show that Block approximates well the maximum flow time.

Theorem 4.2 *Let $\mathcal{B}$ be the schedule produced by* Block *on an instance I. Then*

$$F^{\mathcal{B}}_{\max} \quad \leq \quad 8\sqrt{(n/m)p_{\text{avg}}F^*_{\max}(I)} + 4F^*_{\max}(I),$$

where $p_{\text{avg}} = \sum_j p_j/n$.

Example. Before we analyze Block, let us first illustrate the algorithm on a simple example. Assume that $m = 3$ and let $\mathcal{B}$ denote the final schedule produced by the algorithm. Consider the following set of jobs specified in the form $j = (r_j, p_j, z_j)$: $1 = (0, 1, 1), 2 = (0, 1/2, 1), 3 = (1/5, 2, 1), 4 = (1/2, 1, 1), 5 = (1, 1/2, 1), 6 = (1, 1, 1), 7 = (4, 1, 1)$. At time $\beta_1 = 0$, the jobs $Q_1 = \{1, 2\}$ are pending. Applying RList-M to the set Q_1 yields $S^{\mathcal{B}}_1 = 0, S^{\mathcal{B}}_2 = 1/2$. This schedule finishes at time 1, therefore $\delta_1 = 1$. At time $\beta_2 = \beta_1 + \delta_1 = 1$ the jobs $Q_2 = \{3, 4, 5, 6\}$ are pending. Applying RList-M to this set yields $S^{\mathcal{B}}_3 = 1, S^{\mathcal{B}}_4 = 5/2, S^{\mathcal{B}}_5 = 3, S^{\mathcal{B}}_6 = 5/2$, therefore $\delta_2 = 5/2$. At time $\beta_3 = 4 > \beta_2 + \delta_2 = 7/2$ we have $Q_3 = \{7\}$, and job 7 is scheduled by RList-M at time 4 yielding $\delta_3 = 1$. See Figure 4.

As we can see, the algorithm schedules the jobs in *blocks* (creating one new block in a phase). Each block $i = 1, 2, \ldots$ contains jobs in Q_i, begins at β_i and has length δ_i. It follows from the definition of the algorithm RList-M that in each block at least one machine is continuously busy processing jobs.

In order to prove Theorem 4.2, we will need several lemmas.

Lemma 4.3 *Let $\sigma_0 = 0$ and let $\sigma_1, \sigma_2, \ldots, \sigma_g \geq 0$ be numbers such that $\sigma_i - \sigma_{i-1} \leq \Delta$ for $i = 1, 2, \ldots, g$ and some $\Delta \geq 0$. Then $\sigma_g^2 \leq 2\Delta \sum_{i=1}^{g} \sigma_i$.*

Proof: For any $i = 1, 2, \ldots, g$ we have $\sigma_i^2 - \sigma_{i-1}^2 \leq \Delta(\sigma_i + \sigma_{i-1})$. The lemma follows by adding these inequalities over all values of i. □

Consider any instance I and let $\mathcal{B} = \text{Block}(I)$ be Block's schedule of I. Let B denote the number of blocks in schedule $\mathcal{B}$. (Clearly, $B \leq n$.) We show that in order to prove Theorem 4.2, we may assume that at all times $0 \leq t < C^{\mathcal{B}}_{\max}$ at least one machine is busy in $\mathcal{B}$.

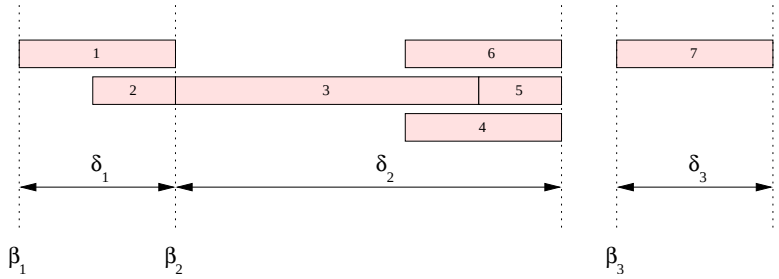

Figure 4: Illustration of algorithm Block

Lemma 4.4 *To prove Theorem 4.2, without loss of generality we may assume that at all times* $t \in [0, C^{\mathcal{B}}_{\max})$ *at least one machine is busy in* $\mathcal{B}$.

Proof: Assume that Theorem 4.2 holds for any instance I' such that at all times $t' \in [0, C^{\mathcal{B}'}_{\max})$ at least one machine is busy in $\mathcal{B}' = \mathsf{Block}(I')$. We claim that then Theorem 4.2 also holds for I.

We divide $\mathcal{B}$ into N segments that do not have idle times. Let $t_1 = \beta_1 = 0$. If t_i is defined, and if there is j such that $t_i < \beta_j + \delta_j < \beta_{j+1}$ then pick such smallest j and set $t'_i = \beta_j + \delta_j$ and $t_{i+1} = \beta_{j+1}$. Otherwise let $t'_i = \beta_B + \delta_B$ and $N = i$. Let I_i be the set of jobs released in the interval $[t_i, t'_i)$, for $i = 1, 2, \ldots, N$. Since Block is never idle when there are available jobs, all jobs from I_i are completed in $\mathcal{B}$ no later than at time t'_i. Let $\mathcal{B}_i$ be the schedule computed by Block on I_i. Note that in the interval $[t_i, t'_i)$ the schedule $\mathcal{B}_i$ is identical to the schedule $\mathcal{B}$. Therefore we have

$$F^{\mathcal{B}}_{\max} = \max_{i=1,\ldots,N} F^{\mathcal{B}_i}_{\max}. \tag{3}$$

Fix some i, $1 \le i \le N$, and denote by $n(I_i)$ and $p_{\text{avg}}(I_i)$, respectively, the number of jobs and the average job length in I_i. Since $n(I_i)p_{\text{avg}}(I_i) \le np_{\text{avg}}$ and $F^*_{\max}(I_i) \le F^*_{\max}(I)$, and using the assumption of the lemma, we have

$$\begin{aligned} F^{\mathcal{B}_i}_{\max} &\le 8\sqrt{(n(I_i)/m)p_{\text{avg}}(I_i)F^*_{\max}(I_i)} + 4F^*_{\max}(I_i) \\ &\le 8\sqrt{(n/m)p_{\text{avg}}F^*_{\max}(I)} + 4F^*_{\max}(I). \end{aligned}$$

This, together with (3), implies that Theorem 4.2 holds for $\mathcal{B}$. □

From now on, in the rest of this sub-section, we assume that $\mathcal{B}$ has no idle times. We now observe that the maximum flow time of any job in schedule $\mathcal{B}$ may be bounded by twice the length of a largest block in $\mathcal{B}$. This is captured by the next lemma.

Lemma 4.5 $F^{\mathcal{B}}_{\max} \le 2\max_i \delta_i$.

Proof: Let j be a job with the maximum flow time in $\mathcal{B}$ and suppose $j \in Q_{i'}$ for some i'. If $i' = 1$ then $F^{\mathcal{B}}_{\max} \le \delta_1$ and the lemma holds. So assume now that $i' \ge 2$. Then $r_j > \beta_{i'-1}$ and $C^{\mathcal{B}}_j \le \beta_{i'} + \delta_{i'}$, therefore $F^{\mathcal{B}}_{\max} = F^{\mathcal{B}}_j \le \beta_{i'} + \delta_{i'} - \beta_{i'-1} = \delta_{i'-1} + \delta_{i'} \le 2\max_i \delta_i$, as $\beta_{i'} = \beta_{i'-1} + \delta_{i'-1}$ by Lemma 4.4. □

The idea behind the analysis of the algorithm is to estimate the size of a largest block in $\mathcal{B}$ in terms of $F^*_{\max}(I)$. In order to do this we derive an inequality that bounds the rate of growth of the blocks in $\mathcal{B}$. This is stated in the next lemma.

Lemma 4.6 *For any $i = 2, \ldots, B$ we have $\delta_i \leq \delta_{i-1} + 2F^*_{\max}(I)$.*

Proof: Let $\mathcal{O}$ denote an optimum schedule of I. Let $\alpha_i = \min_{j \in Q_i} S^{\mathcal{O}}_j$. We first observe that for all $i = 2, \ldots, B$ we have

$$\alpha_i \quad > \quad \beta_{i-1}. \tag{4}$$

To justify this inequality, let $h = \operatorname{argmin}_{j \in Q_i} S^{\mathcal{O}}_j$. We have $S^{\mathcal{O}}_h = \alpha_i$, by the definition of α_i, and $r_h \leq \alpha_i$. On the other hand, since $h \in Q_i$, we have $r_h > \beta_{i-1}$. So we obtain $\beta_{i-1} < r_h \leq \alpha_i$, proving (4).

Let $\lambda_i = \max_{j \in Q_i} C^{\mathcal{O}}_j - \alpha_i$. We now claim that for all $i = 2, \ldots, B$ we have

$$\lambda_i \quad \leq \quad \delta_{i-1} + F^*_{\max}(I). \tag{5}$$

Note that for all $j \in Q_i$ we have $r_j \leq \beta_i = \beta_{i-1} + \delta_{i-1}$ by Lemma 4.4. Let $g \in Q_i$ be a job completed at $\alpha_i + \lambda_i$ in $\mathcal{O}$. We have $F^*_{\max}(I) \geq F^{\mathcal{O}}_g = \alpha_i + \lambda_i - r_g \geq \alpha_i + \lambda_i - (\beta_{i-1} + \delta_{i-1}) \geq \lambda_i - \delta_{i-1}$ by inequality (4), completing the proof of (5).

Theorem 3.3 implies that $\delta_i \leq \lambda_i + p_{\max}$. Thus, using (5) we obtain

$$\begin{aligned} \delta_i &\leq \delta_{i-1} + F^*_{\max}(I) + p_{\max} \\ &\leq \delta_{i-1} + 2F^*_{\max}(I). \end{aligned}$$

This ends the proof. □

We now prove Theorem 4.2.

Proof: As before, let B denote the number of blocks in schedule $\mathcal{B}$. If $B = 1$ then $F^{\mathcal{B}}_{\max} \leq 2F^*_{\max}(I)$ by Corollary 3.4 and the fact that in case $B = 1$ the maximum flow time is equal to the makespan.

So assume from now on that $B \geq 2$. If for all $i = 1, 2, \ldots, B$ we have $\delta_i < 2p_{\max}$, then from Lemma 4.5 we get $F^{\mathcal{B}}_{\max} \leq 2 \max_i \delta_i \leq 4p_{\max} \leq 4F^*_{\max}(I)$, and the theorem follows.

Otherwise, there is i with $\delta_i > 2p_{\max}$. Let $l = \operatorname{argmax}_i \delta_i$. Choose $1 \leq f < l$ to be the maximum index i such that $\delta_i < 2p_{\max}$ and $\delta_{i+1} \geq 2p_{\max}$. If such i does not exist, we set $f = 1$. Define $g = l - f + 1$, $\sigma_0 = 0$, and $\sigma_i = \delta_{f+i-1}$ for $i = 1, \ldots, g$.

For $i = 2, 3, \ldots, g$ we have $\sigma_i - \sigma_{i-1} \leq 2F^*_{\max}(I)$ by Lemma 4.6. If $f = 1$ then $\sigma_1 = \delta_1 \leq 2F^*_{\max}(I)$ (as explained for the case for $B = 1$), and if $f > 1$ then $\sigma_1 = \delta_f \leq 2p_{\max} \leq 2F^*_{\max}(I)$.

We conclude that for each $i = 1, 2, \ldots, g$ we have $\sigma_i - \sigma_{i-1} \leq 2F^*_{\max}(I)$, so the sequence σ_i satisfies the conditions of Lemma 4.3 with $\Delta = 2F^*_{\max}(I)$. We now use this lemma to bound σ_g, the size of a largest block.

Observe that for $i = 2, 3, \ldots, g$ we have $\sigma_i \geq 2p_{\max}$, so the number of jobs in each block Q_{f+i-1} is more than m, by the definition of Block. By an argument similar to that in the proof of Theorem 3.3, and using the fact that all jobs in Q_{f+i-1} are available at the beginning of the $(f+i-1)$th block, we get $\sigma_i \leq p_{\max} + (1/m) \sum_{j \in Q_{f+i-1}} p_j$. Since also $\sigma_i \geq 2p_{\max}$, we conclude that $\sigma_i \leq (2/m) \sum_{j \in Q_{f+i-1}} p_j$, for all $i = 2, 3, \ldots, g$.

We now consider two cases. If $\sigma_1 \geq 2p_{\max}$ then the observation from the previous paragraph also applies to $i = 1$, and

$$\begin{aligned} \sum_{i=1}^{g} \sigma_i &\leq \sum_{i=1}^{g} \left(\frac{2}{m} \sum_{j \in Q_{f+i-1}} p_j \right) \\ &\leq \frac{2}{m} \sum_{j=1}^{n} p_j. \end{aligned}$$

If $\sigma_1 < 2p_{\max}$ then since $\sigma_g \geq 2p_{\max}$, we have $g > 1$ and $\sigma_1 < \sigma_2$, so

$$\begin{aligned}\sum_{i=1}^{g} \sigma_i &\leq \sigma_1 + \sum_{i=2}^{g} \left(\frac{2}{m} \sum_{j \in Q_{f+i-1}} p_j \right) \\ &\leq 2 \sum_{i=2}^{g} \left(\frac{2}{m} \sum_{j \in Q_{f+i-1}} p_j \right) \\ &\leq \frac{4}{m} \sum_{j=1}^{n} p_j.\end{aligned}$$

Overall we have $\sum_{i=1}^{g} \sigma_i \leq (4/m) \sum_{j=1}^{n} p_j$. By Lemma 4.3 we obtain

$$\begin{aligned}(\sigma_g)^2 &\leq 2 \cdot 2F^*_{\max}(I) \cdot \sum_{i=1}^{g} \sigma_i \\ &\leq 4F^*_{\max}(I) \cdot \frac{4}{m} \sum_{j=1}^{n} p_j \\ &= 16(n/m) p_{\text{avg}} F^*_{\max}(I),\end{aligned}$$

and thus, by Lemma 4.5,

$$\begin{aligned}F^{\mathcal{B}}_{\max} &\leq 2\sigma_g \\ &\leq 8\sqrt{(n/m) p_{\text{avg}} F^*_{\max}(I)}.\end{aligned}$$

This ends the proof of Theorem 4.2. □

4.2 A Lower Bound on the Competitive Ratio

We conclude the paper by proving that the algorithm Block has asymptotically the best possible competitive ratio.

Theorem 4.7 *No online randomized algorithm can be better than $O(\sqrt{n/m})$-competitive for $m \geq 2$.*

Proof: We use Yao's minimax principle [14, 3], and show a distribution on instances with $k = 1$ (one chain) that forces each deterministic online algorithm to have expected ratio $\Omega(\sqrt{n/m})$.

Choose a large integer $a > 0$. The general idea is to create a random instance I, with $F^*_{\max}(I) = O(1)$, that will consist of a sequence of a sub-instances $I_1, I_2, ..., I_a$, where the makespan of each I_i will be $\Theta(i)$. The number n of generated jobs will be $\Theta(ma^2)$. Any online algorithm will be forced to increase its maximum flow time between any two consecutive sub-instances, and thus its overall maximum flow time will be at least $\Theta(a) = \Theta(\sqrt{n/m})$.

For the sake of simplicity we assume that m is even. At the end of the proof we explain how to modify the construction in the case when m is odd.

We generate each instance from a random binary string $\mu = \mu_1 \mu_2 \dots \mu_a$, where $\mu_i = 0$ or $\mu_i = 1$, each with probability $\frac{1}{2}$, independently. Let $t_1 = 0$. For any $i = 1, 2, \dots, a$, suppose that t_i and $I_1 \cup I_2 \cup \dots \cup I_{i-1}$ have already been defined. Then take $t_{i+1} = t_i + 3(i+1) + 2\mu_i + 4$, and define a sub-instance I_i of $m(i + 5 + \mu_i)$ jobs as follows:

- m jobs of length 1 released at time t_i;
- $i+1$ batches, each consisting of m jobs of length 3, where batch j is released at time $t_i + 3j$ for $j = 0, 1, \ldots, i$;
- If $\mu_i = 1$, $m/2$ jobs of length 3 released at time $t_i + 3(i+1)$, and $m/2$ jobs of length 1 released at time $t_i + 3(i+1) + 2$; (if $\mu_i = 0$ these jobs are not included);
- $3m$ jobs of length 1 released at time $t_i + 3(i+1) + 3\mu_i$.

The complete instance I is obtained by concatenating sub-instances I_i for $i = 1, 2, \ldots, a$. Figure 5(a) shows a sub-instance I_i for $\mu_i = 1$.

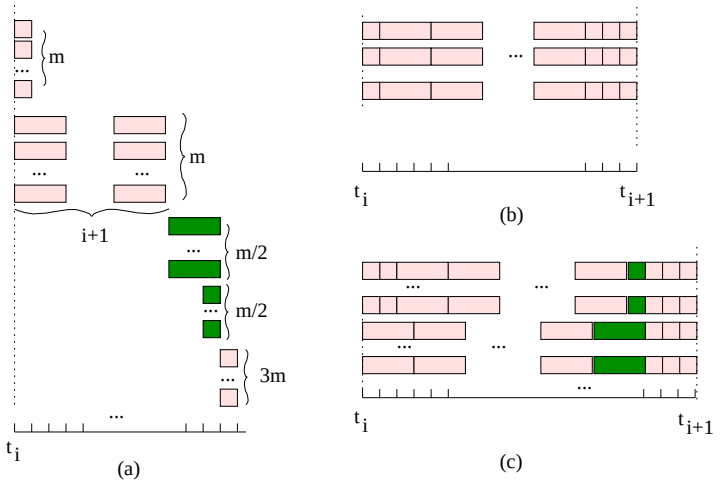

Figure 5: (a) A sub-instance I_i for $\mu_i = 1$. When $\mu_i = 0$ the last $3m$ jobs are released three units earlier and the dark gray jobs are not released at all. (b) Optimum schedule of sub-instance I_i with $\mu_i = 0$. (c) Optimum schedule of I_i for $\mu_i = 1$.

Since every sub-instance I_i contains at most $m(i+6)$ jobs, the total number of jobs in I is $n \leq \sum_{i=1}^{a} m(i+6) = O(ma^2)$.

Lemma 4.8 *There exists a schedule of the instance I with maximum flow time 5 and makespan at most t_{a+1}.*

Proof of the lemma: We show that for every i it is possible to schedule all jobs from I_i in the interval $[t_i, t_{i+1})$, such that the flow time of any job does not exceed 5.

If $\mu_i = 0$, then we schedule the first m jobs from I_i at time t_i each on a separate machine. If $\mu_i = 1$, then we schedule the first $m/2$ jobs from I_i at time t_i on the first $m/2$ machines and the next $m/2$ jobs at time $t_i + 1$ on these same machines. The remaining jobs are always placed on the next available machine (ties broken arbitrarily). By routine inspection, this is a valid schedule, with the last job completed at time t_{i+1}. See Figures 5(b) and 5(c). It is easy to verify that in both cases the flow time of any job is at most 5. □

Let A be an online algorithm. Since all release times and processing times of jobs in I are integral, we may restrict our attention to algorithms which start jobs at integral times only. This justifies the following lemma:

Lemma 4.9 *Without loss of generality we may assume that in any schedule of I algorithm* A *starts and completes jobs at integral times.*

A *time slot* is a unit interval $[t, t+1)$, where t is an integer time. By the lemma above, a job of length p occupies a processor for p consecutive time slots. Let $\mathcal{X}'$ be a schedule of the first n' jobs from I. A time slot $[t, t+1)$ is *idle on machine* h if h does not execute a job in $[t, t+1)$. A *stall slot at* t *on machine* h is an idle slot on h such that there either is a job which completes after time t on h in $\mathcal{X}'$, or $t < C^{\mathcal{X}'}_{\max} - 3$. Since a largest job in I has size 3, in any extension of $\mathcal{X}'$ to a full schedule of I no job will be running in any stall slot.

The main idea of the proof is to show that A creates a schedule with $\Omega(ma)$ idle slots for I, on average. Since the total work is mt_{a+1}, the last completion time in the schedule created by A is $t_{a+1} + \Omega(a)$. It follows that the flow time of the job that completes last in this schedule is $\Omega(a)$, as it is released before t_{a+1}. Finally, as $n = O(ma^2)$ and the flow time of any job in an optimum schedule is bounded by a constant, we can conclude that the competitive ratio of A is $\Omega(a) = \Omega(\sqrt{n/m})$.

To complete the proof it suffices to show that A creates a schedule with $\Omega(ma)$ stall slots for I, on average. This follows from the following lemma:

Lemma 4.10 *For all $i = 1, 2, \ldots, a$ the schedule of $I_1 \cup I_2 \cup \cdots \cup I_i$ computed by* A *contains $\Omega(mi)$ stall time units on average, w.r.t. the random choice of $\mu_1 \ldots \mu_i$.*

Proof of the lemma: Let $\mathcal{A} = \mathsf{A}(I)$ be the schedule created by A. Fix i and let $\mathcal{B}$ denote the sub-schedule of $\mathcal{A}$ consisting of all jobs in $I_1 \cup I_2 \cup \cdots \cup I_{i-1}$ and the first m (unit) jobs from I_i.

If $C^{\mathcal{B}}_{\max} \geq t_i + i + 5$ then, since the total processing time of jobs in $\mathcal{B}$ is $m(t_i + 1)$, the algorithm must have already created mi stall slots and the lemma follows. We can, therefore, assume that $C^{\mathcal{B}}_{\max} < t_i + i + 5$, which implies that $C^{\mathcal{B}}_{\max} < t_i + 3(i+1)$. So at $C^{\mathcal{B}}_{\max}$ the jobs which distinguish instance I_i for $\mu_i = 0$ from I_i for $\mu_i = 1$ are not released yet. We will prove that since at $C^{\mathcal{B}}_{\max}$ the algorithm cannot know the value of μ_i, it cannot avoid creating $\Omega(m)$ stall slots when scheduling jobs from I_i, on average.

Let $J = \{j \in I_i : p_j = 3\}$ and let $\mathcal{B}'$ denote the sub-schedule of $\mathcal{A}$ (and the extension of $\mathcal{B}$) consisting of all jobs in $I_1 \cup I_2 \cup \cdots \cup I_{i-1}$, the first m jobs in I_i and all jobs in J. We wish to count the number of stall slots in $[C^{\mathcal{B}}_{\max} - 2, C^{\mathcal{B}'}_{\max} - 1)$.

Observe that jobs in J are followed in I_i by at least $3m$ jobs of length 1. None of these jobs of length 1 may be started in $\mathcal{A}$ before $C^{\mathcal{B}'}_{\max} - 1$ (otherwise $\mathcal{A}$ would not be an FRFC schedule), and some of these jobs must complete at $C^{\mathcal{B}'}_{\max} + 3$ or later (since the total processing time of these jobs is at least $3m$). Since $p_{\max} = 3$, these jobs force each idle slot in $[C^{\mathcal{B}'}_{\max} - 2, C^{\mathcal{B}'}_{\max} - 1)$ in $\mathcal{B}'$ to be a stall slot in $\mathcal{A}$. This observation will play a crucial role in the argument that follows.

By M we denote the set of machines which complete all jobs in $\mathcal{B}$ at or before time $C^{\mathcal{B}}_{\max} - 2$. Let $\overline{M}$ be the set of the remaining machines. For each h, let l_h be the number of jobs from J scheduled on machine h and $\lambda_h = i + 1 - l_h$. For an integer x, define $[x]^+ = \max\{x, 0\}$.

We now show that the number of new stall slots (i.e., stall slots which are in $\mathcal{B}'$ and not in $\mathcal{B}$) is at least $m/2$ for some value of μ_i. We distinguish two cases.

Case 1: $|M| \geq m/2$ and $\mu_i = 0$. Note that in this case $|J| = \sum_{i=1}^{m} l_h = m(i+1)$, and thus $\sum_{h=1}^{m} \lambda_h = 0$.

First we claim that $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} + 3i + 3$. Indeed, if $l_h = i + 1$ for $h = 1, 2, \ldots, m$, then $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} + 3l_h = C^{\mathcal{B}}_{\max} + 3i + 3$. If for some h we have $l_h \geq i + 2$ then, since the first job from J on h may not complete before $C^{\mathcal{B}}_{\max}$, we also have $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} + 3(l_h - 1) \geq C^{\mathcal{B}}_{\max} + 3i + 3$.

For any machine h, we now estimate the number of stall slots on h in the interval $Q = [C^{\mathcal{B}}_{\max} - 2, C^{\mathcal{B}'}_{\max} - 1)$. The number of stall slots for $h \in M$ is at least $[3i + 4 - 3l_h]^+ = [3\lambda_h + 1]^+$, because there are $|Q| \geq 3i + 4$ slots in Q that will become stall slots if no job is run in them. Similarly, for $h \in \overline{M}$ the number of stall slots in Q is at least $[3i + 2 - 3l_h]^+ = [3\lambda_h - 1]^+$. Therefore the total number of stall slots is at least

$$\sum_{h \in M} [3\lambda_h + 1]^+ + \sum_{h \in \overline{M}} [3\lambda_h - 1]^+. \tag{6}$$

We want to show that for $\sum_{h=1}^{m} \lambda_h = 0$ the sum (6) is minimized when $\lambda_h = 0$ for all h. Indeed, suppose there are f and g for which $\lambda_f < 0 < \lambda_g$. No matter whether $f \in M$ or not, if we increase λ_f by 1 the value of (6) will increase by at most 1. Similarly, if we decrease λ_g by 1, the value of (6) will decrease by at least 2. Thus if we simultaneously increase λ_f and decrease λ_g by 1, the value of $\sum_{h=1}^{m} \lambda_h$ will remain 0 and the sum (6) will decrease.

We thus conclude that the sum (6) is minimized when $\lambda_h = 0$ for all $h = 1, 2, \ldots, m$, yielding a lower bound of $|M| \geq m/2$ on the total number of new stall slots.

Case 2: $|\overline{M}| > m/2$ and $\mu_i = 1$. Note that in this case we have $|J| = \sum_{i=1}^{m} l_h = m(i+1) + m/2$, so $\sum_{i=1}^{m} \lambda_h = -m/2$.

We now claim that $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} + 3i + 5$. If $l_h \geq i+2$ for some $h \in \overline{M}$, then $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} - 1 + 3l_h \geq C^{\mathcal{B}}_{\max} + 3i + 5$. Otherwise, we have $l_h \leq i+1$ for all $h \in \overline{M}$, and therefore, since $|J| = m(i+1) + m/2$ and $|M| < m/2$, there is $h \in M$ for which $l_h \geq i+3$. In this case, $C^{\mathcal{B}'}_{\max} \geq C^{\mathcal{B}}_{\max} + 3(l_h - 1) > C^{\mathcal{B}}_{\max} + 3i + 5$.

For any machine h we now estimate the number of stall slots on h in the interval $Q = [C^{\mathcal{B}}_{\max} - 2, C^{\mathcal{B}'}_{\max} - 1)$. The number of stall slots for $h \in M$ is at least $[3i + 6 - 3l_h]^+ = [3\lambda_h + 3]^+$, because there are $|Q| \geq 3i + 6$ slots in Q that will become stall slots if no job is run in them. Similarly, for $h \in \overline{M}$, the number of stall slots in Q is at least $[3i + 4 - 3l_h]^+ = [3\lambda_h + 1]^+$. Therefore the total number of stall slots is at least

$$\sum_{h \in M} [3\lambda_h + 3]^+ + \sum_{h \in \overline{M}} [3\lambda_h + 1]^+. \tag{7}$$

We claim that for $\sum_{i=1}^{m} \lambda_h = -m/2$ the sum (7) is minimized when $\lambda_h = -1$ for $h \in M$ and $\lambda_h \in \{-1, 0\}$ for $h \in \overline{M}$. The proof of the claim is by an exchange argument similar to that in the previous case. If $\lambda_f \geq 1$ for some f, we can decrease λ_f and increase any λ_g such that $\lambda_g < 0$. If $\lambda_f \leq -2$ for some f, we can increase this λ_f and decrease any other λ_g. Both operations preserve $\sum_{h=1}^{m} \lambda_h$ and do not increase (7). When we are done, we have $\lambda_h \in \{-1, 0\}$ for all h. If $\lambda_f = 0$ for some $f \in M$, choose any $g \in \overline{M}$ such that $\lambda_g = -1$ (there must exist one, since $\sum_{h=1}^{m} \lambda_h = -m/2$ and $|M| < m/2$). If we decrease λ_f and increase λ_g, the value of (7) will decrease.

By the above paragraph, the sum (7) is at least $m/2$, yielding an $m/2$ lower bound on the number of new stall slots.

Since the choice of i was arbitrary, it follows that each schedule of $I_1 \cup I_2 \cup \cdots \cup I_i$ computed by A contains $\Omega(mi)$ stall slots for each $i = 1, 2, \ldots, a$ on average. This completes the proof of Claim 4.10. □

At the beginning of this proof we assumed that m is even. When m is odd we may define each sub-instance I_i in the following way:

- $m-1$ jobs of length 1 released at time t_i;
- $i+1$ batches of m jobs of length 3, where batch j is released at time $t_i + 3j$ for $j = 0, 1, \ldots, i$;
- If $\mu_i = 1$, $(m-1)/2$ jobs of length 3 released at time $t_i + 3(i+1)$, and $(m-1)/2$ jobs of length 1 released at time $t_i + 3(i+1) + 2$;
- $3m+1$ jobs of length 1 released at time $t_i + 3(i+1) + 3\mu_i - 1$.

One may verify that Lemma 4.8 and Lemma 4.10 still hold for the new instance. This completes the proof of Theorem 4.7. □

We would like to point out that in the above construction we only use jobs with two different processing times: 1 and 3.

Acknowledgments. We wish to thank the anonymous referees for insightful comments and for pointing out several mistakes in the earlier version of this paper.

References

[1] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proc. 9th Symp. on Discrete Algorithms (SODA)*, pages 270–279. ACM/SIAM, 1998.

[2] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on networking*, 7(6):789–798, 1999.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[4] J. A. Cobb and M. Lin. A theory of multi-channel schedulers for quality of service. *J. High Speed Netw.*, 12(1,2):61–86, 2003.

[5] E. Feuerstein, M. Mydlarz, and L. Stougie. On-line multi-threaded scheduling. *J. Sched.*, 6(2):167–181, 2003.

[6] R. Gareiss. Is the Internet in trouble? *Data Communications Magazine*, Sept. 1997.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the theory of of NP-Completeness*. Freeman and Company, San Francisco, 1979.

[8] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

[9] C. S. IEEE. Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. In *IEEE Std 802.3. Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements.* The Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, New York, NY 10016-5997, USA, 2002.

[10] W. Jawor. Three dozen papers on online algorithms. *SIGACT News*, 36(1):71–85, 2005.

[11] W. Jawor, M. Chrobak, and M. Molle. Experimental analysis of scheduling algorithms for aggregated links. In preparation, 2006.

[12] K. Pruhs, E. Torng, and J. Sgall. Online scheduling. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15. CRC Press, 2004.

[13] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. In L. A. McGeoch and D. D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–166. AMS/ACM, 1992.

[14] A. C. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Symp. Foundations of Computer Science (FOCS)*, pages 222–227. IEEE, 1977.