Minimizing Total Weighted Flow Time with Calibrations

Vincent Chau
Shenzhen Institutes of Advanced Technology
China
vincentchau@siat.ac.cn

Minming Li
Department of Computer Science
City University of Hong Kong, Hong Kong
minming.li@cityu.edu.hk

Samuel McCauley
IT University of Copenhagen
Denmark
samc@itu.dk

Kai Wang
Department of Computer Science
City University of Hong Kong, Hong Kong
kai.wang@my.cityu.edu.hk

ABSTRACT

In sensitive applications, machines need to be periodically calibrated to ensure that they run to high standards. Creating an efficient schedule on these machines requires attention to two metrics: ensuring good throughput of the jobs, and ensuring that not too much cost is spent on machine calibration.

In this paper we examine flow time as a metric for scheduling with calibrations. While previous papers guaranteed that jobs would meet a certain deadline, we relax that constraint to a tradeoff: we want to balance how long the average job waits with how many costly calibrations we need to perform.

One advantage of this metric is that it allows for online schedules (where an algorithm is unaware of a job until it arrives). Thus we give two types of results. We give an efficient offline algorithm which gives the optimal schedule on a single machine for a set of jobs which are known ahead of time. We also give online algorithms which adapt to jobs as they come. Our online algorithms are constant competitive for unweighted jobs on single or multiple machines, and constant-competitive for weighted jobs on a single machine.

1 INTRODUCTION

Modern industrial products like processors and digital cameras must be manufactured, consistently, to exacting standards. The machines that make these products are high-performance and high-maintenance, and perform very precise tasks. As such, they need to be calibrated before they can be trusted to perform a job. Performing these calibrations can be very expensive, possibly much more than the cost of running the machines. The calibrations are only effective for a set period of time, after which the machine may no longer be accurate, and must be (expensively) recalibrated before running more jobs.

With these costs in mind, recent work has considered scheduling on machines that need to be calibrated [1, 8, 14]. The goal of algorithms in this framework is to minimize the number of calibrations, while ensuring that all jobs complete by their deadlines.

This objective function has an interesting property: in most objective functions (say, minimizing total waiting time, or maximizing number of jobs completed by a deadline), it is generally profitable to schedule a job as early as possible. But to minimize calibrations, the algorithm wants to group jobs together as much as possible, possibly delaying some jobs considerably. The known algorithms in this framework are designed to effectively handle this tradeoff: grouping jobs for more efficient calibrations, while maintaining a reasonable schedule where jobs are not delayed too long.

The original motivation for scheduling with calibrations came directly from the Integrated Stockpile Evaluation (ISE) program to test nuclear weapons periodically. The tests for these weapons require calibrations that are expensive in a monetary (thought not necessarily temporal) sense. This motivation is specified further in [8, 12, 20].

However, this motivation can extend to any context where machines performing jobs must be calibrated periodically. For example, high-precision machines require periodic calibration to ensure precision. Methods for calibrating these machines is itself an area of research; some examples include [23, 25, 27, 28]. Oftentimes, machines are no longer considered to be accurately calibrated after a set period of time. There are many guidelines to determine this *calibration interval*, or period of time between calibrations, both from industry and academia [5, 7, 17, 18, 22, 26].

Calibrations have applications in many areas, including robotics [9, 13, 21], pharmaceuticals [3, 7, 15], and digital cameras [2, 6, 29].

We formally model calibrations as follows. We must ***calibrate*** a machine before it is able to perform tasks. The machine stays ***calibrated*** for $T \geq 2$ time steps, after which it must remain idle until it is recalibrated. We refer to these T time steps following a calibration as an ***interval***. Calibrating a machine is instantaneous, meaning that a machine can be recalibrated between two job executions that run in successive time steps.

We consider the following calibration costs/constraints. In the online setting (Section 3), each calibration has cost G; our objective is to minimize the sum of the calibration cost and the total flow time. In the offline setting (Section 4) we have a budget of K calibrations;

The work described in this paper was supported by a grant from Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11268616), by NSFC (No. 61433012, U1435215), by Shenzhen basic research grant JCYJ20160229195940462, by NSF grants CCF 1617618, IIS 1247726, IIS 1251137, CNS 1408695, and CCF 1439084, and by Sandia National Laboratories. Work done in part while Samuel McCauley was visiting the City University of Hong Kong.

SPAA'17, , July 24–26, 2017, Washington, DC, USA.

ACM ISBN 978-1-4503-4593-4/17/07…$15.00
http://dx.doi.org/10.1145/3087556.3087573

the objective is to find the schedule that minimizes total flow time while only calibrating K times.

While minimizing calibrations, we still want to make sure the algorithm provides good throughput for the jobs. In this paper, we want to minimize the average weighted flow time of jobs (the weighted average of the time each job waits, from when it is released to its completion). Because the number of jobs remains constant, minimizing the total flow time is equivalent to minimizing the average; the remainder of the paper only discusses minimizing total flow time for simplicity.

1.1 Results

Our online algorithms include:

- a 3-competitive algorithm for a single machine with unweighted jobs,
- a 12-competitive algorithm for a single machine with weighted jobs, and
- a 12-competitive algorithm for multiple machines with unweighted jobs.

We give a lower bound of 2 for the online case.

For the offline case with one machine and weighted jobs, we give an $O(Kn^3)$ dynamic programming algorithm, where n is the number of jobs and K is the budget for the number of calibrations.

While proving the performance of our algorithms we give several structural lemmas for scheduling with calibrations, which may be useful for future research.

1.2 Related Work

The initial study of scheduling with calibrations was by Bender et al. [8], who considered minimizing the number of calibrations for jobs with release times and deadlines. They gave an optimal solution for a single machine, but were only able to achieve a 2-approximation for multiple machines (except for some special cases).

Later, Fineman and Sheridan extended these results for jobs with non-unit processing times [14]. Their work begins with the observation that minimizing calibrations for jobs with deadlines generalizes the well-known machine minimization problem—as T becomes arbitrarily large, the problem is simply to minimize the number of machines. Somewhat surprisingly, the calibrations for arbitrary T can be minimized nearly as efficiently: an algorithm with a given performance bound for machine minimization leads to an algorithm with similar performance for minimizing calibrations.

More recently, Angel et al. developed dynamic programming algorithms for further generalizations—where for example there are multiple kinds of calibrations, or jobs have nonunit processing times, but are preemptible [1].

Scheduling with calibrations has similarities with some other well-known scheduling problems, such as minimizing idle periods [4], and scheduling on cloud-based machines which must be rented to perform work [19].

2 PRELIMINARIES

Our algorithm must schedule a set of n jobs $\mathcal{J}$ on P machines. Each job j has a release time r_j, a weight w_j, and a processing time $p_j = 1$ (all jobs have identical, unit length). If a job is scheduled to begin at t_j, it ends at $t_j + 1$, and incurs flow $w_j(t_j + 1 - r_j)$.

We assume jobs are indexed in ascending release time order, i.e. $r_1 \leq \ldots \leq r_n$. Furthermore, we assume that there are at most P jobs with any given release time without loss of generality (so if $P = 1$, we assume that all release times are distinct).[1]

We refer to the time period $[t, t + 1)$ as ***time step*** t. We assume all such t, as well as all release times, are integers. We call a time step ***idle*** if no job is being processed during that time step.

A machine can be ***calibrated*** at a time step t for cost G. We refer this time step as the ***calibration time***. This calibration happens instantaneously (the calibration does not prevent the machine from running jobs during both t and $t - 1$, if it is otherwise able). We refer to the T calibrated time steps from t to $t + T - 1$ as an ***interval***; we say that the interval ***begins*** at t and ***ends*** at $t + T$. We call an interval ***full*** if the machine is never idle during the interval, and ***non-full*** otherwise.

A schedule $\mathcal{S}$ consists of two parts:

- an assignment of each job j to a time step t and a machine m, and
- a set of calibration times for each machine.

A correct schedule only assigns one job to each time step on any machine, and only assigns jobs to calibrated time steps (i.e. no more than T time steps after the machine was calibrated).

We begin with an observation that if the set of calibration times is given, we can optimally assign the jobs to machines and time steps using an efficient, online algorithm.

Observation 2.1. *Consider a set of calibration times C, a set of jobs $\mathcal{J}$, and a number of machines P, we can optimally assign $\mathcal{J}$ to time steps using the following online algorithm:*
For each time step t:

1. *Add all jobs arriving at t to the set of waiting jobs W.*
2. *For every calibration in C at t, calibrate the next machine in round-robin order.*
3. *For each machine m that is calibrated at t, schedule the highest-weight job $j \in W$ on m at time t. If multiple jobs are tied for heaviest weight, choose the job with earliest release time. Remove j from W.*

Proof. Let $\mathcal{S}$ be the schedule given by this algorithm.

Since the machines are indistinguishable, they can be calibrated in round-robin order without loss of generality. In the case that there are more than P calibrations in less than T time steps, scheduling in round-robin order maximizes the number of calibrated time steps. See [8, Lemma 7] for a thorough discussion.

To show that this is an optimal assignment of jobs to times, we proceed by contradiction. Let t be the first time that $\mathcal{S}$ deviates from every optimal schedule—that is, $\mathcal{S}$ schedules j at t, but there exists an optimal $\mathcal{S}'$ that either schedules j' at t with $w_j \neq w_{j'}$, or chooses not to schedule at all. (Meanwhile, no schedule that matches $\mathcal{S}$ up to $t - 1$ schedules j at t.)

Consider a schedule $\mathcal{S}''$ which is the same as $\mathcal{S}'$, except we swap the times when j and j' are scheduled (if S' does not schedule a job, we let $w_{j'} = 0$ and swap j' with the empty time step at t). Since $w_j \neq w_{j'}$, $w_{j'} < w_j$ by definition of $\mathcal{S}$. Then $\mathcal{S}''$ has smaller weighted flow time than $\mathcal{S}'$, contradicting its optimality.

[1] If more than P jobs have the same release time r, we take the lightest job and increase its release time by 1. Since at most P jobs can be scheduled at r (and it's always cheapest to delay the lightest job), this does not change the optimal cost of the instance.

A similar exchange argument shows that ties can be broken by release time without loss of generality. □

3 ONLINE ALGORITHMS

In this section we consider an online model for scheduling with calibrations. In the online setting, an algorithm first learns about job j at time r_j: before the algorithm makes any decisions at time t, it receives a list of jobs with $r_j = t$, as well as w_j for each such job.

In this section, we limit our calibrations by giving them a (potentially very large) cost that must be balanced with flow. Each calibration costs G; our goal is to minimize

$$G \cdot (\#\text{ calibrations}) + \sum_{j \in \mathcal{J}} w_j(t_j + 1 - r_j)$$

where job j begins at time t_j and ends at time $t_j + 1$.

Our online algorithms do not make any assumptions on G or T. In particular, if $T < G/T$, while our algorithms work as-is, they can be further simplified while achieving equal or better bounds. For example, the "immediate calibrations" in Algorithm 1 can be removed entirely, and the corresponding charging arguments can be considerably simplified. Similarly, if $G/T < 1$, our online algorithms all schedule every incoming job immediately, again greatly simplifying their operation and analysis. We focus on bounds that apply for all G and T, leaving the analysis of online scheduling with calibrations under special cases to future work.

We analyze our algorithms using competitive analysis. An algorithm is said to be ***α-competitive*** if, for every sufficiently large input I, the algorithm performs at most an α factor worse than the optimal offline algorithm on I [10, 24].

We refer to an optimal schedule for an input $(\mathcal{J}, P)$ as $\mathrm{OPT}(\mathcal{J}, P)$. Our analysis usually considers a fixed input, in which case we refer to an optimal schedule as OPT.

We begin with a lower bound, which shows that no deterministic algorithm is better than 2-competitive.

Lemma 3.1. *For a single machine and unweighted jobs, any deterministic online algorithm can not be better than $(2-o(1))$-competitive. (The $o(1)$ term depends on T and G.)*

Proof. The adversary begins by releasing a job at time 0. Our lower bound considers two cases.

(1) If the online algorithm calibrates at time 0, the adversary releases another job at time T. Then the online algorithm incurs calibration cost $2G$ and flow cost 2, with a total cost of $2G + 2$. Meanwhile, the optimal solution can calibrate at $t = 1$, with cost $3 + G$. This gives a competitive ratio of $\frac{2G+2}{G+3} = 2 - \frac{4}{G+3}$.
(2) If the online algorithm waits (does not calibrate immediately) then the adversary releases one more job for each time step from 1 to $T - 1$. Then the online algorithm has cost at least $2T + G$ (since each job has flow at least 2), whereas an optimal algorithm calibrates at time 0 and has cost $T + G$. Thus the competitive ratio is $\frac{2T+G}{T+G} = 2 - \frac{G}{T+G}$.

Thus, the competitive ratio can be arbitrarily close to 2 for large G, and $T \gg G$. □

3.1 Unweighted Single-Machine Algorithm

In this section, we examine a special case where all jobs are unweighted ($w_j = 1$) and must be scheduled on a single machine ($P = 1$). We give a 3-competitive algorithm.

The idea of the algorithm is to delay arriving jobs until their flow is equal to the calibration cost G. However, the algorithm has a maximum number of jobs it can delay, after which it must schedule.

Algorithm 1 Online Unweighted Calibration on One Machine

```
Q ← empty priority queue of jobs, ordered by release time
for each time step t do
    if a job j arrives at time t then
        Insert j in Q
    end if
    if t is not calibrated then
        f ← flow cost of scheduling all j ∈ Q starting at t + 1
        if Q contains ≥ G/T jobs or f ≥ G then
            Calibrate at t
        else
            p ← total flow of jobs in the most recent calibration
            if p < G/2 and a job is released at t then
                Calibrate at t
            end if
        end if
    end if
    if Q is not empty and t is calibrated then
        Remove the earliest-released job j′ from Q
        Schedule j′ at t
    end if
end for
```

At a high level, Algorithm 1 is similar to known algorithms, like the optimal solution to the ski rental problem. Both delay until a large penalty cost is reached (G total flow in this case).

On the other hand, Algorithm 1 is more aggressive at some points: if there are G/T waiting jobs, or if a job is released after a calibration in which jobs had total flow less than $G/2$, it calibrates regardless of the flow of the current waiting jobs. Interestingly, although the algorithm appears to schedule somewhat early in these cases (especially if G/T is small), it is still 3-competitive.

We bound the algorithm's performance using a charging argument. To begin, we observe that OPT and Algorithm 1 schedule jobs in the same order: release time order (this follows from Observation 2.1). Our proof relies heavily on this structure.

Before proving the competitive ratio, we need a structural lemma to show that we never double-charge to a calibration. Let J_i be the set of jobs scheduled in interval i by Algorithm 1. We partition J_i into two subsets: J_i^E is the set of jobs scheduled earlier in OPT than in Algorithm 1 or at the same time in both, and J_i^L is the set of jobs scheduled strictly later.

Lemma 3.2. *Consider an interval i scheduled by Algorithm 1 starting at time b_i with nonempty J_i^E. Let i^{OPT} be the earliest interval in OPT containing a job in J_i. Then for all $i' > i$, i^{OPT} contains no jobs in $J_{i'}$.*

Proof. Since J_i^E is nonempty, and both OPT and Algorithm 1 schedule in release time order, the first job in J_i must be in J_i^E. Since the first job in J_i is scheduled in the first time step of i, i^{OPT} must begin no later than i.

Let j' be the first job in J_{i+1}. If j' is released after $b_i + T$, it (and thus all jobs in i') must be scheduled after $b_i + T$ as well, so it cannot be scheduled in i^{OPT}. On the other hand, if j' is released before $b_i + T$, since Algorithm 1 schedules according to Observation 2.1, there must be T jobs in J_i. Then there are $T + 1$ jobs in $J_i \cup \{j'\}$; thus j' cannot be scheduled in i^{OPT}, and neither can any job in $J_{i'}$. □

Theorem 3.3. *Algorithm 1 is* 3*-competitive.*

Proof. We use a charging argument for each interval i. We charge to both costs in OPT: the cost of calibrating intervals, and the flow of jobs. We argue that each calibration is only charged to once using Lemma 3.2. The flow of each job is only charged to once because we only charge to the flow of jobs in J_i.

We break into two cases: in the first, Algorithm 1 calibrated i due to total flow of G; in the second, the calibration was due to G/T waiting jobs. If Algorithm 1 calibrated $i + 1$ because i had flow at most $G/2$ (but there were less than G/T waiting jobs with total flow less than G), we call $i + 1$ an ***immediate calibration***. If $i + 1$ is an immediate calibration, we consider the cost of i and $i + 1$ simultaneously.

Let b_i be the time when Algorithm 1 schedules interval i. Let f_i be the total flow of all jobs in i released before b_i, and let e_i be the total flow of all jobs in i released on or after b_i. We have $f_i \le G$ by Algorithm 1, and $e_i \le G$ since there be at most T incoming jobs, and each is delayed by $|Q| \le G/T$. As in Lemma 3.2, let i^{OPT} be the earliest interval in OPT containing a job in J_i.

Case 1 (Calibrated due to flow G): We split into two subcases. First, we assume that all jobs in i are scheduled later in OPT than in Algorithm 1 (J_i^E is empty). Then we charge to the flow of the jobs in OPT. Algorithm 1 incurs a cost of $G + f_i + e_i < 2G + e_i$, while OPT incurs a cost of at least $f_i^{OPT} + e_i \ge G + e_i$, leading to a competitive ratio of 2.

Otherwise, (in the second subcase) J_i^E is nonempty. We charge to i^{OPT}; this charging is unique by Lemma 3.2. Algorithm 1 incurs a cost of $G + f_i + e_i < 3G$ while OPT incurs a calibration cost of G; we obtain a competitive ratio of 3.

Case 2 (Calibrated due to G/T waiting jobs): We split into three subcases.

Case 2a (Nonempty J_i^E): First, we assume that J_i^E is nonempty. If there is no immediate calibration following i, we can charge to i^{OPT}. This calibration is only charged to once by Lemma 3.2. Algorithm 1 has cost $G + f_i + e_i < 3G$; we charge to an interval with calibration cost G.

If $f_i + e_i \le G/2$, and $i + 1$ is an immediate calibration, consider whether interval $i+1$ has a job scheduled earlier in OPT (i.e. whether J_{i+1}^E is empty). If interval $i+1$ has a job scheduled earlier in OPT, we charge to two intervals in OPT: i^{OPT} and $(i + 1)^{OPT}$ (the earliest interval in OPT containing a job in J_{i+1}). Lemma 3.2 shows that these are two different intervals, and that they are not charged to by any other interval. Intervals i and $i + 1$ have total cost $2G + e_i + f_i + e_{i+1} + f_{i+1} < 5G/2$; we charge to a calibration cost of $2G$ in OPT. Otherwise, if all jobs in J_{i+1} are scheduled later in OPT, Algorithm 1 has cost $5G/2 + e_{i+1} + f_{i+1}$, and OPT has cost at least $G + e_{i+1} + f_{i+1}$ (again charging to i^{OPT}). In all cases, we achieve a competitive ratio of 3.

Case 2b (All jobs in J_i delayed at least T): In the second subcase, we assume that J_i^E is empty (all jobs in J_i are scheduled later in OPT), and all jobs in J_i are scheduled at or after $b_i + T$ in OPT. We charge the cost of J_i ($G + e_i + f_i$) to the jobs' flow in OPT, which must be at least $G + e_i + f_i$.

If J_{i+1} is an immediate calibration, all jobs in J_{i+1} have flow cost 1, and must have larger flow cost in OPT. Thus we charge the calibration and flow costs of J_i and J_{i+1} ($2G + e_i + f_i + e_{i+1} + f_{i+1}$) to their flow costs in OPT, which must be at least $G + e_i + f_i + e_{i+1} + f_{i+1}$, giving a competitive ratio of 2.

Case 2c (All jobs delayed; some by a small amount): Finally, we assume that all jobs in J_i are scheduled later in OPT, but at least one is scheduled before $b_i + T$. This means that the interval i^{OPT} must be scheduled before $b_i + T$. This interval contains no jobs in $J_{i'}$ for $i' > i + 1$: if i^{OPT} contains a job from J_{i+1}, this follows from Lemma 3.2; otherwise, all jobs in J_{i+1} (and thus $J_{i'}$) are scheduled after i^{OPT} ends. We charge both i and $i + 1$ to i^{OPT}—even if $i + 1$ is not an immediate calibration.

First, assume that $i+1$ is an immediate calibration. Then i and $i+1$ have total cost at most $2G + e_i + f_i + e_{i+1} + f_{i+1} \le 5G/2 + |J_{i+1}|$, as each job in J_{i+1} is scheduled immediately. We charge to cost $G + |J_{i+1}|$ in OPT: G from the calibration cost of i^{OPT}, and $|J_{i+1}|$ since each job in J_{i+1} must incur flow cost at least 1 in OPT. This gives a competitive ratio of 5/2.

If $i+1$ is not an immediate calibration (and i has total flow at least $G/2$), these intervals have total cost at most $2G + f_i + e_i + f_{i+1} + e_{i+1}$. Since all jobs in J_i are scheduled later in OPT, we charge to their flow as well as the calibration of i^{OPT}, so OPT has cost at least $G + f_i + e_i$. The ratio between these is minimized when $f_i + e_i$ is minimized, at $f_i + e_i = G/2$. Then Algorithm 1 has cost $5G/2 + f_{i+1} + e_{i+1} < 9G/2$, and OPT has cost at least $3G/2$, giving a competitive ratio of 3. □

3.2 Weighted Single-Machine Algorithm

When jobs have weights (but must be scheduled on a single machine), we follow a similar algorithm to the unweighted case. The main differences are that the algorithm now calibrates if the waiting jobs have total weight G/T, and the algorithm no longer performs immediate calibrations (after an interval with flow $\le G/2$).

We assume that all algorithms schedule the heaviest possible job first, breaking ties by release time (Observation 2.1). Thus, our algorithm and OPT may schedule jobs in a different order, since which job is scheduled next depends on which time steps are calibrated.

This lack of a common ordering makes charging more difficult. Our solution to this is twofold. First, in Lemma 3.4, we show that we can restrict ourselves to algorithms that schedule in order of release time, losing only a factor 2 in cost. Second, we charge sequences of full intervals (rather than each interval individually). While it is difficult to reason about Algorithm 2's job order within each interval, we show in Lemmas 3.6 and 3.7 that we can make useful statements about its behavior between sequences.

Lemma 3.4. *If an optimal solution of a given instance has cost C_{OPT}, there exists a solution with all jobs scheduled in order of release time with cost at most $2C_{OPT}$.*

Algorithm 2 Online Weighted Calibration on One Machine

```
Q ← empty priority queue of jobs, ordered by release time
for each time step t do
    if a job j arrives at time t then
        Insert j in Q
    end if
    if t is not calibrated then
        f ← flow cost of scheduling all j ∈ Q starting at t + 1
        if ∑_{j∈Q} w_j ≥ G/T or |Q| = T or f ≥ G then
            Calibrate at t
        end if
    end if
    if Q is not empty and t is calibrated then
        Extract the job j′ with smallest weight from Q
        Schedule j′ at t
    end if
end for
```

Proof. We show how to transform an optimal schedule (with C calibrations) into a new schedule where jobs are in release time order. This new schedule will schedule all jobs no later than they were in the original schedule, so the flow cost will be smaller. Furthermore, the new schedule will have no more than $2C$ calibrations. Thus the total cost of the new schedule is at most $2C_{\text{OPT}}$.

Consider each job in the schedule, in order from latest to earliest release time. We maintain the invariant that after considering a job j, it is scheduled at a time t_j such that $t_j \geq r_j$, and all jobs with $r_{j'} > r_j$ have $t_{j'} > t_j$. At intermediate points while constructing this schedule, several jobs may be scheduled at the same time. However, we also maintain that after considering job j, no two jobs with release times at least r_j are scheduled at the same time.

For each job j, move j before all jobs j' with $r'_j > r_j$. In other words, move j to the time step immediately before the job with the next-largest release time is scheduled (even if there is a job already scheduled at this time step).

We now show that these invariants are maintained. Each job j is scheduled before all jobs with later release times by definition. To show $t_j \geq r_j$, we must have all jobs with $r_{j'} > r_j$ scheduled at $t_{j'} \geq r_{j'} \geq r_j + 1$; thus, r_j is a time step before all jobs with later release times are scheduled, so it satisfies the requirement for t_j. Since t_j is the maximum time step satisfying the requirement, $t_j \geq r_j$. Finally, no job with release time larger than r_j can be scheduled at t_j; thus j is not scheduled at the same time as a job with larger release time. Since the invariant was maintained for all jobs with larger release time, the invariant is maintained.

Now we show that there are no more than $2C$ calibrations. Each time we add a job, we add one new time step—this time step is already calibrated, or is adjacent to a sequence of newly-filled time steps, the last of which is adjacent to a calibrated time step.

Consider a sequence of calibrated time steps in the new schedule. All jobs from this sequence must have come from intervals contained in this sequence, as a job j being pushed back never "skips over" an uncalibrated time step. As j is pushed back, it may empty some slots, but we assume that they remain calibrated in the new schedule. Thus, all jobs from this calibrated sequence must come from an interval in this sequence. Let p be the number of previously-uncalibrated slots in this sequence. Then the number of extra calibrations necessary is $\lceil p/T \rceil$. Furthermore, the number of calibrations previously in this interval was at least $\lceil p/T \rceil$. Thus the number of calibrations for each sequence is at most doubled. Thus the total number of calibrations is no more than $2C$. □

In some circumstances we want to charge the online algorithm's flow cost to OPT's calibration cost (like we did for the cases with nonempty J_E in the proof of Theorem 3.3). But if jobs have extremely large weights, our intervals can have correspondingly large flow, even if they are scheduled at release time. Similarly, Algorithm 2 calibrates immediately if the waiting jobs have total weight more than G/T; however, heavy incoming jobs can cause a much larger total queue weight. Lemma 3.5 helps deal with these issues by showing that if we ignore the unavoidable flow cost of w_j for every job, we can bound the total flow of jobs in an interval.

Lemma 3.5. *Let J_i be the set of jobs scheduled in interval i by Algorithm 2, where each $j \in J_i$ is processed at time t_j.*[2] *Then $\sum_{j \in J_i} w_j(t_j - r_j) < 2G$.*

Proof. Let i begin at b_i. The total flow cost of jobs in interval i is the sum of three terms:

(1) the flow incurred by jobs if they had been scheduled at b_i (the value f in Algorithm 2 is this flow at $b_i + 1$),
(2) the flow of incoming jobs scheduled immediately, and
(3) for each time step t, w_j flow for each unfinished $j \in J_i$.

Term 1 is at most G by Algorithm 2, and term 2 does not count towards our cost (since these jobs are scheduled at their release time). Thus, our goal is to show that the total cost of term 3 is at most $G + \sum_{j \in J_i} w_j$.

Let Q_t denote the queue of waiting jobs at time $b_i + t$, not including the job scheduled at $b_i + t$. Then we want to show $\sum_{t=0}^{T-1} \sum_{j \in Q_t} w_j \leq G$.

First we bound the cost of Q_0. At the time step before b_i, Algorithm 2 did not schedule an interval, so the total weight of all waiting jobs was less than G/T. At b_i, any job released at time b_i is added to the queue, and the largest job in Q_0 is removed; thus $\sum_{j \in Q_0} w_j < G/T$.

Now we show that $\sum_{j \in Q_{i-1}} w_j \leq \sum_{j \in Q_i} w_j$. At each time step, a job enters the queue, and a job is popped off the queue to be scheduled. If no job arrives, or if the arriving job does not have the largest weight in the queue, this is a net decrease in weight and we have $Q_i < Q_{i-1}$. If the arriving job has the largest weight in the queue, it is scheduled immediately, and $Q_i = Q_{i-1}$. Thus, $\sum_{t=0}^{T-1} \sum_{j \in Q_t} w_j \leq \sum_{t=0}^{T-1} \sum_{j \in Q_0} w_j < G$. □

As mentioned before, our charging argument considers groups of consecutive intervals. In particular, we partition the schedule of Algorithm 2 into maximal groups of consecutive intervals, called ***sequences***, such that all but the last interval in each sequence is full.[3] This partitioning is unique, since the boundaries between sequences are exactly the non-full intervals. We say a sequence ***ends*** at e_I, the final time step of its last interval, and ***begins*** at b_I, the time step immediately after the previous sequence ends

[2] Thus j finishes at $t_j + 1$ and incurs flow $w_j(t_j + 1 - r_j)$.
[3] The last interval may be full if it is the last interval in the entire schedule.

($b_I = 0$ for the first sequence). Observation 2.1 implies that all jobs scheduled within each sequence I are released on or after b_I.

Let OPT_r be the optimal algorithm that schedules all jobs in order of release time.

The following lemma is the basis of our charging argument. Essentially, we show that for any sequence I, since most of the intervals are full, OPT_r must calibrate all but the last interval earlier than in Algorithm 2.

Lemma 3.6. *For every sequence I, for every $k < |I|$, there are at least k intervals scheduled by OPT_r that:*

- *end after b_I, but*
- *begin no later than the kth interval in I.*

Proof. Assume the contrary: let k^{OPT} be the kth interval scheduled by OPT_r after b_I, and k^I be the kth interval in I, with k^{OPT} scheduled after k^I. Since k^I is full, there were at least kT jobs released between b_I and the time k^I ends. In particular, since OPT_r schedules in release time order, all jobs in k^{OPT} were released by the time k^I ends. This means that k^{OPT} can be scheduled one time step earlier, improving its flow time and contradicting the definition of OPT_r. □

Our final structural lemma shows that we can, in some cases, bound the flow of jobs scheduled later in OPT_r than in Algorithm 2 for a given sequence. The assumptions in the lemma are specific because they closely match our charging scheme.

Lemma 3.7. *Let I be a sequence of intervals scheduled by Algorithm 2 with jobs J_I, and let ℓ be the last interval in I. Let ℓ^{OPT} be the $|I|$th interval in OPT_r containing jobs in J_I. Assume ℓ^{OPT} begins after ℓ ends. Let f_ℓ^q be the total flow of the jobs in ℓ if no new jobs were incoming (corresponding to the value f in Algorithm 2 one time step before ℓ is scheduled), and let f_ℓ be the actual flow incurred by all jobs in ℓ. Then the total flow incurred by OPT_r of all jobs in J_I scheduled in ℓ^{OPT} and later intervals is at least $f_\ell - f_\ell^q$.*

Proof. Deferred to the full version for space. □

Now we are ready to prove the performance of our algorithm. We show that Algorithm 2 is 6-competitive with OPT_r, which implies by Lemma 3.4 that it is 12-competitive with OPT.

Theorem 3.8. *Algorithm 2 is 12-competitive.*

Proof. Consider a sequence of intervals I. We charge the last unit of flow of all jobs scheduled by Algorithm 2 to their last unit of flow in OPT_r.

We charge the remainder of the cost of the kth interval of I to the kth interval of OPT_r ending after b_I, for all $k < I$. In particular, we charge to half its calibration cost, $G/2$. We only charge to half the cost because we will charge to some intervals twice.

For the last interval in I, let ℓ^{OPT} be the $|I|$th interval containing jobs in J_I in OPT_r.

(1) If ℓ^{OPT} begins after e_I and is charged to by two later sequences, we charge to the flow of all jobs in J_I scheduled by OPT_r in ℓ^{OPT} or later.
(2) Otherwise, if $|I| = 1$ and ℓ^{OPT} begins after b_I, we charge to half the calibration cost of ℓ^{OPT}, *and* the flow cost of all jobs in J_I scheduled by OPT_r in ℓ^{OPT} or later.
(3) Otherwise, we charge to one time step of flow for each $j \in J_i$, plus half the calibration cost of ℓ^{OPT}.

First, we show that we charge to any interval i in OPT_r at most twice. By Lemma 3.6, i cannot be charged to by two intervals from the same sequence I. Since all jobs in a sequence I are released after I begins, and all but the last interval charge to earlier intervals by Lemma 3.6, if an interval is charged to by two different sequences, it must be charged to by the $|I|$th interval for some I. Then by the definition of Case 1, it can only be charged to twice.

We now examine the total costs of an interval i with jobs J_i, in a sequence I with jobs J_I. We split into two cases.

Case 1 (i charges to a calibration cost in OPT_r): We charge the flow incurred during the time step in which each job is scheduled to its flow in OPT_r, and the remainder of the cost of i to the kth interval in OPT_r containing jobs from J_I. By Lemma 3.5 we charge a cost of $3G + \sum_{j \in J_i} w_j$ to a cost in OPT of at least $G/2 + \sum_{j \in J_i} w_j$; this leads to a competitive ratio of 6.

Case 2 (i charges to flow): If i charges to flow, by Case 1, there are two intervals i_1, i_2 that charge to the calibration cost of an interval ℓ^{OPT} containing jobs from J_I. If $|I| > 1$, we examine the total cost of i, i_1, and i_2, and lower bound the cost they charge to. Otherwise, if $|I| = 1$, we charge the cost of i directly.

First, for this argument to be correct, we need that for every ℓ^{OPT}, there is only one interval in a sequence of size greater than one that charges to flow in ℓ^{OPT} (since we charge to the calibration cost of ℓ^{OPT} when $|I| > 1$). This follows from Lemma 3.6.

Now, consider the case where $|I| = 1$. The total cost is $G + f_i$, where f_i is the flow in I. Let f_i^{OPT} be the flow of the jobs in J_I incurred in OPT_r. Since all jobs in i (and thus all jobs in J_I) are scheduled after ℓ^{OPT}, and ℓ^{OPT} begins after e_I, they are scheduled beginning at least T time steps later in OPT_r. Immediately, we have $f_i^{OPT} > f_i$. If i was scheduled due to flow G we have $f_i^{OPT} > G$ by definition; if i was scheduled due to G/T weight of waiting jobs, each waits for T additional steps, and again $f_i^{OPT} > G$. Thus $G + f_i$ is at most twice f_i^{OPT}.

Now, consider $|I| > 1$. Let f_i be the flow of all jobs in i, and f_i^q be the flow of all jobs in i incurred if there had been no incoming jobs after i began (see Lemma 3.7). Then by Lemma 3.7, i charges to a flow of at least $f_i - f_i^q$. Either i_1 or i_2 must charge to ℓ^{OPT} using Case 2; without loss of generality we call this i_1. Then i_1 is in a sequence of size 1. In particular, all jobs scheduled in i_1 are scheduled in ℓ^{OPT} or later and must incur at least the same flow.

Then the total cost of i, i_1, and i_2 is $G + f_i + G + f_{i_1} + G + f_{i_2} \leq 5G + f_i + f_{i_1} + \sum_{j \in J_{i_2}} w_j$ by Lemma 3.5. We charge to a cost of $G + (f_i - f_i^q) + f_{i_1} + \sum_{j \in J_{i_2}} w_j$. Since $f_i^q \leq G$, our competitive ratio is

$$\frac{6G + (f_i - f_i^q) + f_{i_1} + \sum_{j \in J_{i_2}} w_j}{G + (f_i - f_i^q) + f_{i_1} + \sum_{j \in J_{i_2}} w_j} \leq 6.$$ □

3.3 Multiple Machines

We give a competitive online algorithm when jobs can be assigned to multiple machines, and jobs do not have weights.

When there are multiple machines, charging becomes infeasible, as small perturbations in the intervals can lead to significant

changes in which jobs belong to which interval (if they are scheduled in order of release time).

Instead, we use another method: the primal-dual approach. This allows us to use linear-programming-based techniques to derive a lower bound on the cost of OPT directly, without charging to intervals containing specific jobs. On the other hand, this seems to come at some loss of efficiency: we only show that our algorithm for multiple machines is 12-competitive.

Our algorithm is an extension of Algorithm 2. We wait until there are G/T waiting jobs or the jobs have total flow G to calibrate. Then, we schedule jobs in order of release time, starting with the machine with smallest index first.

With multiple machines, we need to be careful how we define waiting jobs. In particular, once we calibrate, we want the jobs we will schedule in that interval to no longer count as waiting jobs when deciding if we should calibrate further. Thus, when we calibrate, we assign jobs to the intervals immediately. This means that the guarantee that jobs are scheduled in order of release time does not hold across machines for this algorithm.

In particular, Algorithm 3 schedules jobs explicitly, rather than using the schedule in Observation 2.1. While we are still able to prove a constant approximation ratio, in practice one would almost certainly only use Algorithm 3 to determine calibration times, and use Observation 2.1 for the actual assignments. In particular, Algorithm 3 may schedule jobs late in an interval (in Step 13), incurring extra flow over Observation 2.1's schedule if a largely-empty interval is scheduled concurrently.

Algorithm 3 Online Unweighted Calibration on Multiple Machines

1: $Q \leftarrow$ empty priority queue of jobs, ordered by release time
2: **for** each time step t **do**
3: **for** all jobs j arriving at time t **do**
4: Insert j in Q
5: **end for**
6: **for** all calibrated machines m idle at t **do**
7: Remove the earliest-released job j from Q
8: Schedule j on m at t
9: **end for**
10: $f \leftarrow$ flow cost of scheduling all jobs in Q starting at $t+1$
11: **while** Q contains $\geq G/T$ jobs **or** $f \geq G$ **do**
12: Calibrate at t on the next machine in round robin order
13: Schedule up to G/T jobs from Q in this interval in release time order
14: **end while**
15: **end for**

We begin with several simple observations about the flow costs of each interval scheduled by Algorithm 3.

Observation 3.9. *Let i be an interval scheduled on machine m at time step b_i by Algorithm 3. Then*

- *every job in i incurs flow at most $2G/T$ after b_i,*
- *the total flow of all jobs in i is at most $3G$, and*
- *if Algorithm 3 schedules i due to flow ($f \geq G$ in Step 11), the total flow of jobs in i is at least $G - G/T$.*

Proof. Let $j \in J_i$ be in Q at b_i or a later time. Since jobs are scheduled in release time order, and all but the first G/T time steps of i are free, j is scheduled at most $G/T + |Q| \leq 2G/T$ time steps after b_i.

The total flow of all jobs in i is at most G up to time b_i (since all jobs incurring flow before b_i must be released before b_i and thus must be in Q at b_i). There are at most T jobs in i, each of which incurs flow at most $2G/T$ after b_i, giving a total flow of at most $3G$.

Let f be the total flow if the jobs were scheduled at $b_i + 1$ (as defined in Step 10 of Algorithm 3), and f_i be the actual flow of all jobs in I, scheduled beginning at b_i. Since only jobs in Q are waiting at time b_i (by definition), $f \leq f_i + |Q|$. Since $f \geq G$ and $|Q| < G/T$, $f_i \geq G - G/T$. □

In Figure 1 we give the linear program we use to analyze the algorithm. By observation, any schedule $\mathcal{S}$ with total cost C corresponds to a solution to the linear program with cost C. We use the dual of the linear program to lower bound this cost, thus lower bounding the cost of any schedule $\mathcal{S}$.

In particular, the weak duality theorem guarantees that *any* solution to the primal LP has at least the cost of any solution to the dual LP. Our proof shows that at every time step, the cost incurred by Algorithm 3 can be offset by a cost increase in the dual LP, and thus a cost increase in OPT. See [11, 16] for further discussion of this technique.

We have four constraints in the linear program. The first ensures that flow is high until we fully calibrate. The second ensures that the flow being incurred at a given time step can only decrease by 1 from the previous time step (per machine), and only during calibrated time steps. The third ensures that each job j is assigned to at least one machine m. The fourth makes sure that every job incurs flow cost of at least 1. The first and second constraints seem slightly redundant; however, both are necessary for our proof.

These following variable assignments show that any schedule for an online calibrations instance satisfies these constraints (and help explain the intuition behind them). Let $f_{t,j} = 1$ if job j incurs flow at time step t. Let $c_{t,m} = 1$ if an interval begins on machine m at time t. Finally, let $a_{j,m} = 1$ if job j is scheduled on machine m. All variables are 0 in all other cases.

Primal:

$$\text{minimize} \quad \sum_t \sum f_{t,j} + G \sum c_{t,m}$$

$$f_{t,j} + \sum_{t'=r_j-T} c_{t',m} - a_{j,m} \geq 0 \quad \forall j, t \geq r_j, m$$

$$\sum_{r_j<t} (f_{t,j} - f_{t-1,j}) + \sum_m \sum_{t'=t-T}^{t} c_{t',m} \geq 0 \quad \forall t$$

$$\sum_m a_{j,m} \geq 1 \quad \forall j$$

$$f_{r_j,j} = 1 \quad \forall j$$

Figure 1: A linear program for scheduling with calibrations.

Taking the dual, we obtain the LP given in Figure 2. All variables are required to be nonnegative in the primal. In the dual, we have $x_{t,j} \geq 0$, $y_t \geq 0$, $v_j \geq 0$, and z_j unbounded.

Theorem 3.10. *Algorithm 3 is 12-competitive.*

Dual:

$$\begin{aligned}
\text{maximize } & \sum z_j + \sum v_j \\
\sum_p x_{t,j,m} - y_{t+1} + z_j &\le 1 \quad \forall t, j \text{ with } t = r_j \\
\sum_p x_{t,j,m} + y_t - y_{t+1} &\le 1 \quad \forall t, j \text{ with } t \neq r_j \\
\sum_{j | r_j \le t+T} \sum_{t' > t} x_{t',j,m} + \sum_{t'=t}^{t+T} y_t &\le G \quad \forall t, m \\
\sum_{t > r_j} -x_{t,j,m} + v_j &\le 0 \quad \forall j, m
\end{aligned}$$

Figure 2: The dual of the linear program given in Figure 1.

Proof. As mentioned earlier, we use the primal dual technique. We set the variables every time Algorithm 3 calibrates (or, in some cases, only after Algorithm 3 calibrates twice. This is similar to when two intervals are charged simultaneously in the proof of Theorem 3.3). In each case, we show that the increase in the dual objective function is at least 1/12 the cost incurred by Algorithm 3. This shows that Algorithm 3 is 12-competitive.

We let $v_j = \max_m \sum_t x_{t,j,m}$. Intuitively, we use $x_{t,j,m}$ to keep track of the flow of job j at time t on machine m. Since Algorithm 3 only assigns flow to one machine, v_j can store the total flow of job j. We set $y_t = G/2T$ for all t, and $z_j = G/2T$ for all j. These variables help contribute to cost when there are a large number of jobs that may not have much flow (i.e. Algorithm 3 calibrates due to G/T waiting jobs).

We examine each interval i one by one and find the increase in the dual LP objective after the interval is scheduled. Let b_i be the time when interval i is scheduled, let m_i be the machine i is scheduled on, and let J_i be the jobs scheduled in i. Our argument is split into cases based on why the algorithm decided to schedule the jobs.

Case 1 (Total flow G): In this case, we calibrate because the waiting jobs have total flow at least G. We split into two subcases. First, assume that there is a job which is released before i ends, and scheduled on m_i, but not scheduled in i. Then there are T total jobs scheduled during i. We keep $x_{t,j,m} = 0$ for all t, m for all $j \in J_i$, but recall that $z_j = G/2T$ for all $j \in J_i$. This gives a total dual cost of $G/2$; since i has total cost at most $4G$ by Observation 3.9, this gives a competitive ratio of 8.

Now we can assume that all jobs scheduled on m_i with $r_j < b_i + T$ are scheduled in i or an earlier interval.

First, we deal with the corner case $T = 1$. Then i has only one job j. We charge to $z_j = G/2$, again giving a competitive ratio of 8.

Otherwise, $T \ge 2$. Note that all jobs on each machine are scheduled in order of release time. For all $t \le b_i$ and for all jobs $j \in J_i$, let $x_{t,j,m_i} = 1/2$ if job j is waiting at time t, and $x_{t,j,m} = 0$ otherwise. Since the jobs incur at most G flow before b_i, the sum of these $x_{j,t,m}$ is at most $G/2$. Therefore, the third condition is satisfied for all times t, m.

Let f_j be the total flow of job j. By Observation 3.9, $f_j \le 2G/T + \sum_t 2x_{t,j,m}$. Algorithm 3 incurs cost $G + \sum_{j \in J_i} f_j$.

The dual cost increases by $\sum_{j \in J_i} \left(G/2T + \sum_t x_{t,j,m} \right) \ge \sum_{j \in J_i} f_j/4$. By Observation 3.9, the jobs in J_i have total flow at least $G - G/T \ge G/2$, so $\sum_{j \in J_i} f_j/6 \ge G/12$.

Thus the competitive ratio is

$$\frac{G + \sum_{j \in J_i} f_j}{\sum_{j \in J_i} f_j/4} \le \frac{G + \sum_{j \in J_i} f_j}{G/12 + \sum_{j \in J_i} f_j/12} \le 12.$$

Case 2 (G/T waiting jobs): We consider intervals i and $i+1$. These two intervals have cost at most $6G$. We show that the dual increases in cost by at least $G/2$. We set $x_{j,t,m_i} = c$ for all waiting jobs $j \in J_i$,[4] for a constant c defined momentarily. Otherwise, $x_{j,t,m} = 0$ for all t, m for all $j \in J_{i+1}$ and all other $j \in J_i$.

If there are exactly G/T jobs scheduled in i with release time at most b_i, we let $c = 1/2$. If there are more than G/T jobs remaining (e.g. if G/T is not an integer or many jobs are released at b_i), we reduce c until $cT(\#\text{ waiting jobs}) = G/2$.

The third condition is satisfied since only the waiting jobs have $x_{t,j,m_i} > 0$ for $b_i \le t < b_i + T$; by definition of c the sum of these terms is exactly $G/2$.

By Observation 3.9, Algorithm 3 incurs cost at most $4G + \sum_{j \in J_i} 2G/T + \sum_{j \in J_{i+1}} 2G/T$, while the dual increases by at least $G/2 + \sum_{j \in J_i} G/2T + \sum_{j \in J_{i+1}} G/2T$. This gives a competitive ratio of 8. □

4 OFFLINE SCHEDULING

In this section, rather than a calibration cost, we have a calibration budget K. Our goal is to minimize the flow time while calibrating at most K times on $P = 1$ machine.

This is a generalization of the online model, as we can use a binary search to find the optimal calibration budget (between 1 and n calibrations) for a given calibration cost G.[5]

As before, we consider unit jobs. Our offline solution allows arbitrary weights for each job, and guarantees an optimal solution on one machine.

Lemma 4.1. *Let j be a job scheduled at t_j in interval i starting at b_i. Then in any optimal schedule, either:*

- *j starts at its release time ($t_j = r_j$), or*
- *there is no idle time between b_i and t_j.*

Proof. By contradiction. Let j be the first contradicting job in some optimal schedule; in other words, j is the earliest job scheduled such that $t_j \neq r_j$, and there exists an empty time step t between b_i and t_j.

If r_j does not have a job scheduled, we can schedule j at r_j, strictly improving the flow time of the schedule and achieving a contradiction.

Otherwise, let j' be the job scheduled at r_j. Since all release times are distinct, $r_{j'} < r_j$; since we assumed j is the earliest job violating the lemma, we must have $t > r_j$. Then we can schedule j at t, strictly improving the flow time of the schedule; thus the schedule cannot be optimal. □

[4] By "waiting jobs" we mean the jobs with $r_j \le t < t_j$.

[5] One may hope for an online result using a calibration budget as well. However, a calibration budget leaves an online algorithm completely helpless: whether it should spend a calibration now, or wait until more jobs have accumulated, depends entirely on the future schedule.

Lemma 4.2. *There exists an optimal schedule such that the last time step of each interval i is a job that is scheduled at its release time.*

Proof. Without loss of generality we assume the last time step of i is not idle (otherwise, we move the calibration back in time until a job is scheduled in the last time step).

First, assume that i is non-full. Then by Lemma 4.1, the last job in j (indeed, all jobs scheduled after an empty time step in i) must be scheduled at its release time.

Otherwise, assume that i is full. For a contradiction, assume that the last job in i is not scheduled at its release time. Let t be the latest release time of any job in i. If $t = b_i + T$, the job released at t must be scheduled at t and we are done. Otherwise, push i back so that it ends immediately after t (in other words, $b_i \leftarrow t + 1 - T$). Since all jobs have distinct release times, all jobs previously in i can still be scheduled in i; if we schedule using Observation 2.1 (without loss of generality), each job is scheduled no later than before. This leads to a schedule with better flow, contradicting the optimality of the original schedule. □

By Lemma 4.1, we have the following corollary.

Corollary 4.3. *In any optimal schedule, for any non-full interval i, any job that is released before the end of interval i must be scheduled in an interval which starts no later than i.*

Proof. Let t be the first idle time step in interval i. No job is released at time t (otherwise time step t will not be idle). For jobs released before t, they can not be scheduled in a time step after t since t is idle and calibrated. For jobs released after t and before the end of interval i, they should be scheduled at its release time according to Lemma 4.1, as time step t is idle. □

With these lemmas in mind, our offline schedule is built around critical jobs.

Definition 4.4. Given a feasible schedule, job j is ***critical*** if it is scheduled at its release time and any job that is released before r_j is scheduled before r_j.

The remainder of the proofs in this section are deferred to the full version for space.

4.1 The Offline Algorithm

By Corollary 4.3, in an optimal schedule, the last job of each non-full interval must be a critical job. We consider two adjacent non-full intervals such that all intervals between them (maybe zero) are full, let jobs j and j', $j < j'$ be the last job of the two non-full intervals respectively, then there must be a group of $\left\lceil \frac{j'-j}{T} \right\rceil$ intervals which only schedule jobs $\{j+1, ..., j'\}$. Moreover, there is at most one non-full interval in the group, and it must be the last interval in the group.[6] Then, we propose a dynamic programming approach to find such groups of intervals and obtain an optimal schedule.

We denote for each job $j \in \mathcal{J}$ a distinct rank $\mu_j \in \{1, 2, ..., n\}$ which corresponds to the ascending order of their weights (break ties by ranking the job of latest release time first). Let $J(u, v, \mu) = \{j \mid r_u \le r_j \le r_v, \mu_j > \mu\}$ be the set of jobs that are released during $[r_u, r_v + 1)$ with ranks higher than μ. We define $f(u, v, \mu)$ as the minimum total weighted completion time of jobs in $J(u, v, \mu)$ with a group of exactly $\lceil |J(u, v, \mu)|/T \rceil$ intervals such that the last interval starts at $r_v + 1 - T$ and all intervals in the group are full (except the last one in the case that $|J(u, v, \mu)|$ is not divisible by T). We define $F(k, v)$ as the minimum total weighted completion time of jobs $\{1, ..., v\}$ with at most k calibrations.

Proposition 1.

$$F(k, v) = \min_{u \le v} \left\{ F\left(k - \left\lceil \frac{v-u+1}{T} \right\rceil, u-1\right) + f(u, v, 0) \right\}$$

We set $F(k, v) \leftarrow +\infty$ if $kT < v$, and $F(k, v) \leftarrow 0$ if $v = 0$.

We maintain the invariant that job v is critical and test for each job u before or equal to job v whether job $u - 1$ is a critical job. Then, we use another dynamic program to calculate the minimum total weighted completion time $f(u, v, 0)$ of jobs $\{u, ..., v\}$.

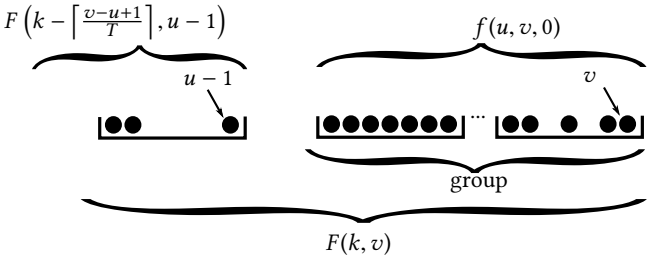

Figure 3: Illustration of Proposition 1

In the following, we show how to compute $f(u, v, \mu)$ for jobs $J(u, v, \mu)$ with a budget of $\lceil |J(u, v, \mu)|/T \rceil$ calibrations.

Let i be the last interval, and $b_i = r_v + 1 - T$ be the time when interval i starts (it ends at $r_v + 1 = b_i + T$). We look for the optimal schedule such that, except for i, all other intervals are full. Without loss of generality, we assume that no two intervals overlap with each other.[7]

Definition 4.5. Given u, v, μ such that $J(u, v, \mu) \neq \emptyset$:

- let $e = \arg\min_{j \in J(u,v,\mu)} \mu_j$ be the job of smallest rank,
- let $\Psi = \{j \mid |J(u, j, \mu)| \mod T \equiv 0,\ j \in J(u, v-1, \mu)\}$ be the set of jobs j such that the number of jobs $J(u, j, \mu)$ is a multiple of T,
- let $j_\ell = \arg\max_{j \in \Psi} r_j$ be the job with latest release time r_ℓ of jobs Ψ if $\Psi \neq \emptyset$, and
- let $s = \min \{h \mid h \equiv |\{j \mid r_j < r_v + 1 - T + h,\ j \in J(u, v, \mu)\}| \mod T\}$.

Lemma 4.6. *As defined above, s is the smallest value such that the machine is completely busy during $[b_i, b_i + s)$, and every job is scheduled at its release time during $[b_i + s, b_i + T)$.*

We focus on where we should schedule the job e of smallest rank. Suppose in the optimal schedule job e is scheduled in interval $i' = [b_{i'}, b_{i'} + T)$ at time step t_e. Firstly, no job except e could be released at t_e, otherwise a schedule with better or equal cost can be

[6] These groups are similar to the sequences defined in Section 3.2, but are used significantly differently.

[7] This can clearly be done by perturbing intervals. We do need to be slightly careful to maintain Lemma 4.2; in particular, if two intervals overlap, we should perturb them by scheduling the first one earlier rather than the second one later.

obtained by swapping the schedule of that job and job e, since job e has the smallest weight. Secondly, jobs that are released before t_e must not be scheduled after t_e. Assume by contradiction that such job i exists. Then a better schedule can be obtained by swapping the schedule of job i and job e. Therefore, any job that is scheduled in interval $[t_e + 1, b_{i'} + T)$ must be scheduled at its release time. Consequently, the last job of i' must be a critical job.

Proposition 2. *We set $f(u, v, \mu) \leftarrow 0$ if $J(u, v, \mu) = \emptyset$; and $f(u, v, \mu) \leftarrow \infty$ if $\Psi \neq \emptyset$ and $b_i \leq r_\ell$.*

Otherwise, $f(u, v, \mu) =$

$$\min \begin{cases} f(u, v, \mu_e) + w_e(r_e + 1) & \textbf{if } r_e \geq I_{begin} + s \\ f(u, v, \mu_e) + w_e(r_v + 1 - T + s) & \\ & \textbf{if } r_e < I_{begin} + s, s > 0 \\ \min\limits_{j \in \Psi,\ r_j \geq r_e} f(u, j, \mu) + f(j + 1, v, \mu) & \textbf{if } \Psi \neq \emptyset \end{cases}$$

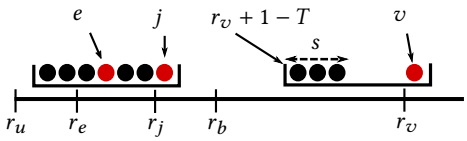

Figure 4: Illustration of Proposition 2

Theorem 4.7. *This dynamic programming algorithm requires $O(Kn^3)$ time.*

5 CONCLUSION

This work is the first to our knowledge to study online scheduling with calibrations, and the first to give algorithms that allow for a tradeoff between throughput and calibrations. However, open problems remain. We would like to tighten the bounds for the single-machine unweighted case: the online algorithm is 3-competitive, while the lower bound is only 2. Can this be tightened?

An interesting open problem is to fully explore the connection with machine minimization. Fineman and Sheridan showed that the problems are essentially equivalent in the offline case with resource augmentation [14]; can a similar statement be made about online scheduling?

REFERENCES

[1] Eric Angel, Evripidis Bampis, Vincent Chau, and Vassilis Zissimopoulos. 2017. On the Complexity of Minimizing the Total Calibration Cost. In *Proceedings of the 11th International Workshop on Frontiers in Algorithmics (FAW '17) (LNCS)*, Vol. 10336. Springer, Heidelberg. To appear.

[2] Richard Baer. 2005. Self-calibrating and/or self-testing camera module. (Sept. 30 2005). US Patent App. 11/239,851.

[3] Surendra K Bansal, Thomas Layloff, Ernest D Bush, Marta Hamilton, Edward A Hankinson, John S Landy, Stephen Lowes, Moheb M Nasr, Paul A St Jean, and Vinod P Shah. 2004. Qualification of analytical instruments for use in the pharmaceutical industry: A scientific approach. *Aaps Pharmscitech* 5, 1 (2004), 151–158.

[4] Philippe Baptiste. 2006. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th. ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*. SIAM, Philadelphia, 364–367.

[5] HP Barringer. 1995. Cost Effective Calibration Intervals Using Weibull Analysis. In *Proceedings of the 11th. Quality Congress*. American Society For Quality Control, Milwaukee, 1026–1038.

[6] Andrew Barton-Sweeney, Dimitrios Lymberopoulos, and Andreas Savvides. 2006. Sensor localization and camera calibration in distributed camera sensor networks. In *Proceedings of the 3rd. International Conference on Broadband Communications, Networks and Systems (BROADNETS '06)*. IEEE, Piscataway, 1–10.

[7] Beamex. 2007. Traceable and Efficient Calibrations in the Process Industry. (October 2007). https://www.beamex.com/wp-content/uploads/2016/12/CalibrationWorld_2007-03-ENG.pdf.

[8] Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. 2013. Efficient Scheduling to Minimize Calibrations. In *Proceedings of the 25th. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '13)*. ACM Press, New York, NY, 280–287.

[9] Rolf Bernhardt and S Albright. 1993. *Robot calibration*. Springer Science & Business Media, Heidelberg.

[10] Allan Borodin and Ran El-Yaniv. 2005. *Online computation and competitive analysis*. Cambridge University Press, Cambridge.

[11] Niv Buchbinder and Joseph Naor. 2009. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science* 3, 2–3 (2009), 93–263.

[12] Chris Burroughs. 2006. New Integrated Stockpile Evaluation program to better ensure weapons stockpile Safety, Security, Reliability. http://www.sandia.gov/LabNews/060331.html. (March 2006). Online; posted March 2006.

[13] Roger C Evans, John E Griffith, David D Grossman, Myron M Kutcher, and Peter M Will. 1982. Method and Apparatus for Calibrating a Robot to Compensate for Inaccuracy of the Robot. (Dec. 7 1982). US Patent 4,362,977.

[14] Jeremy T Fineman and Brendan Sheridan. 2015. Scheduling Non-Unit Jobs to Minimize Calibrations. In *Proceedings of the 27th. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '15)*. ACM Press, New York, NY, 161–170.

[15] M Forina, MC Casolino, and C De la Pezuela Martínez. 1998. Multivariate calibration: applications to pharmaceutical analysis. *Journal of pharmaceutical and biomedical analysis* 18, 1 (1998), 21–33.

[16] Michel X Goemans and David P Williamson. 1997. *The primal-dual method for approximation algorithms and its application to network design problems*. PWS Publishing Co, Boston. 144–191 pages.

[17] James R. Lakin. 2014. Establishing Calibration Intervals, How Often Should One Calibrate? http://www.inspec-inc.com/home/company/blog/inspec-insights/2014/09/30/establishing-calibration-intervals-how-often-should-one-calibrate. (Sep 2014). Online; posted 30 September 2014.

[18] Kuo-Huang Lin and Bin-Da Liu. 2005. A gray system modeling approach to the prediction of calibration intervals. *IEEE Transactions on Instrumentation and Measurement* 54, 1 (2005), 297–304.

[19] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. 2016. Cost-efficient Scheduling on Machines from the Cloud. In *Proceedings of the 10th. International Conference on Combinatorial Optimization and Applications (COCOA '16)*. Springer, Heidelberg, 578–592.

[20] National Nuclear Safety Administration. 2014. Office of Test and Evaluation. http://nnsa.energy.gov/aboutus/ourprograms/defenseprograms/stockpilestewardship/testcapabilitiesand-eval. (September 2014). Online; posted 30 September 2014.

[21] Hoai-Nhan Nguyen, Jian Zhou, and Hee-Jun Kang. 2013. A new full pose measurement method for robot calibration. *Sensors* 13, 7 (2013), 9132–9147.

[22] Emilia Nunzi, Gianna Panfilo, Patrizia Tavella, Paolo Carbone, and Dario Petri. 2005. Stochastic and reactive methods for the determination of optimal calibration intervals. *IEEE Transactions on Instrumentation and Measurement* 54, 4 (2005), 1565–1569.

[23] SR Postlethwaite, DG Ford, and D Morton. 1997. Dynamic calibration of CNC machine tools. *International Journal of Machine Tools and Manufacture* 37, 3 (1997), 287–294.

[24] Daniel D Sleator and Robert E Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.

[25] Tobias Wilken, C Lovis, A Manescau, Tilo Steinmetz, L Pasquini, G Lo Curto, Theodor W Hänsch, Ronald Holzwarth, and Th Udem. 2010. High-precision calibration of spectrographs. *Monthly Notices of the Royal Astronomical Society: Letters* 405, 1 (2010), L16–L20.

[26] Donald W. Wyatt and Howard T. Castrup. 1991. Managing Calibration Intervals. (1991).

[27] G Zhang and R Hocken. 1986. Improving the accuracy of angle measurement in machine calibration. *CIRP Annals-Manufacturing Technology* 35, 1 (1986), 369–372.

[28] Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334.

[29] Zhengyou Zhang. 2002. Method and system for calibrating digital cameras. (Aug. 20 2002). US Patent 6,437,823.