
Article:
Difallah, D., Checco, A., Demartini, G. et al. (2019) Deadline-aware fair scheduling for multi-tenant crowd-powered systems. ACM Transactions on Social Computing, 2 (1). pp. 1-29. ISSN: 2469-7818

https://doi.org/10.1145/3301003

Deadline-Aware Fair Scheduling for Multi-Tenant Crowd-Powered Systems

DJELLEL DIFALLAH, New York University, United States
ALESSANDRO CHECCO, University of Sheffield, United Kingdom
GIANLUCA DEMARTINI, University of Queensland, Australia
PHILIPPE CUDRÉ-MAUROUX, University of Fribourg, Switzerland

Crowdsourcing has become an integral part of many systems and services that deliver high-quality results for complex tasks such as data linkage, schema matching, and content annotation. A standard function of such *crowd-powered* systems is to publish a batch of tasks on a crowdsourcing platform automatically and to collect the results once the workers complete them. Currently, these systems provide limited guarantees over the execution time, which is problematic for many applications. Timely completion may even be impossible to guarantee due to factors specific to the crowdsourcing platform, such as the availability of workers and concurrent tasks. In our previous work, we presented the architecture of a crowd-powered system that reshapes the interaction mechanism with the crowd. Specifically, we studied a push-crowdsourcing model whereby the workers receive tasks instead of selecting them from a portal. Based on this interaction model, we employed scheduling techniques similar to those found in distributed computing infrastructures to automate the task assignment process. In this work, we first devise a generic scheduling strategy that supports both fairness and deadline-awareness. Second, to complement the proof-of-concept experiments previously performed with the crowd, we present an extensive set of simulations meant to analyze the properties of the proposed scheduling algorithms in an environment with thousands of workers and tasks. Our experimental results show that, by accounting for human factors, micro-task scheduling can achieve fairness for best-effort batches and boosts production batches.

CCS Concepts: • **Information systems → Crowdsourcing**; • **Theory of computation → Scheduling algorithms**;

Additional Key Words and Phrases: Crowdsourcing Systems; Task Scheduling; Priority; Human Factors; Task Scheduling; Deadline

ACM Reference Format:
Djellel Difallah, Alessandro Checco, Gianluca Demartini, and Philippe Cudré-Mauroux. 2019. Deadline-Aware Fair Scheduling for Multi-Tenant Crowd-Powered Systems. *ACM Trans. Soc. Comput.* 2, 1, Article 1 (January 2019), 30 pages.
https://doi.org/10.1145/3301003

1 INTRODUCTION

Thanks to micro-task crowdsourcing platforms, such as Amazon Mechanical Turk (AMT) and Crowdflower (now Figure-Eight), it is today possible to build *crowd-powered* systems combining both the scalability of computers with the yet unmatched cognitive abilities of the human brain.

2469-7818/2019/1-ART1 $15.00
https://doi.org/10.1145/3301003

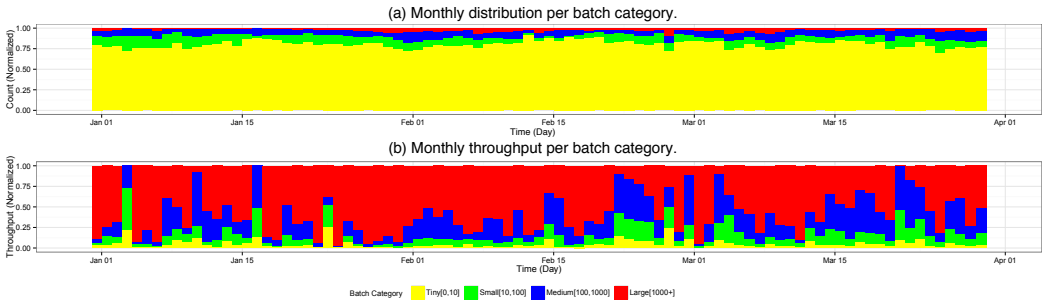


Fig. 1. An analysis of three months of activity logs on Amazon MTurk (January-March 2014) obtained from mturk-tracker.com [29] The crawler frequency is 20 minutes, hence it might miss some batches. All tasks considered in this plot are rewarded $0.01. Throughput measured in task/minute for batches of different sizes.

Micro-task crowdsourcing has already been used in database systems and search engines [18, 24]. In such systems, human and machines behave fundamentally differently: While machines can deal with large volumes of data, with real-time requests, and with flocks of concurrent users interacting with the system, crowdsourcing is mostly used as a batch-oriented, offline data processing paradigm. Moreover, crowdsourcing platforms do not provide guarantees on task completion times, due to the unpredictability of the crowd workers, who are free to come and go at any point in time and to selectively focus on an arbitrary subset of the tasks (also known as Human Intelligence Tasks or HITs) only.

With increased momentum around crowdsourcing for both academic and commercial purposes, obtaining timely responses has become indispensable. Practitioners circumvent this issues by adjusting the price of the tasks or by repeatedly re-publishing them [7, 19, 23]. We argue that some of these issues are due to the *pull*-based model of AMT, where the workers can browse and choose among a list of batches. Before delving into the details of our work, we first present the issues surrounding a pull-based platform by performing an analysis of logs collected from AMT, and discussing the operating model of select services that use crowdsourcing.

1.1 Motivation I: Batch Starvation

To showcase the disparity in the treatment of batches running on AMT we computed the throughput (i.e., the number of tasks completed per minute) of every batch publicly visible during a three months period. We grouped the batches into four categories: tiny (less than 10 HITs), small (10-100 HITs), medium (100-1000 HITs), and large (more than 1000 HITs). The results are depicted in Figure 1. We observe that large batches dominate the throughput of AMT even if the vast majority of the running batches are very small (less than 10 HITs). As a result, large batches are first completed at a faster rate then gradually lose momentum as the final remaining tasks take a much longer time to finish.

Such a *batch starvation* phenomenon has been observed in a number of recent reports, e.g., in [23, 49] where authors observe that the batch completion time depends on its size and HIT pricing. A similar observation was made in [24], where the authors compared the throughput of different batch sizes and concluded that large batches have the highest throughput. Workers on AMT tend to explore new batches with many HITs, since they have a high reward potential, without requiring to search for and select a new type of HIT, these conclusions were reached in qualitative and quantitative studies in [11, 14].

1.2 Motivation II: Production versus Best-effort Batches

By examining the logs of batches that appeared on AMT, we observed that the demand (new batches arriving) exhibits a strong weekly periodicity [14]. From this observation, we conjuncture that crowdsourcing is used routinely for production purposes. Although industry surveys [37] supports this claim, it is difficult to characterize the needs and constraints of different requesters. Broadly, we can distinguish two classes of demand:

- *Production Batches*: These are tasks that are posted routinely by large production pipelines, e.g., real-time trend detection on Twitter [9], slow search queries [16], active learning [39], etc. Routine batches can also be submitted by service providers such as *SpeechPad.com*, *Grammarly.com*, or *Tagasaurus.com*, that use human computation as a backend to fulfill their own customers' requests. In these production setups, a time-bounded response is often critical to the service.
- *Best-effort Batches*: Other crowdsourcing tasks can be published in a best-effort manner by engineers and scientists for a wide variety of purposes, e.g., data gathering, running surveys, testing or debugging. Also, offline systems that publish large batches can operate in that mode, for instance, annotating a large number of pictures to improve an existing machine learning model. While such use-cases do not necessarily have a time constraint, steady progress is often desirable or expected.

Currently, on a pull-crowdsourcing platform, these two classes of batches run side-by-side, and compete for the same pool of workers, whereas both have different execution requirements. For instance, consider the extreme case of a crowdsourced customer service agent *CrowdIO.com* that needs to be assigned within seconds from the chat initiation. Or, a transcription service like SpeechPad that has several pricing schemes for different turnaround levels (as short as 24 hours). At peak demand, such services will still have access to the same crowd and would share the workforce along with thousands of concurrent batches, including annotation tasks and surveys. To solve this issue, other service providers like CloudFactory.com and ScaleAPI.com have built their crowd communities. A similar approach is being adopted by multiple large corporations who have developed internal platforms powered by their employees. This solution guarantees a dedicated crowd but fails to scale when necessary.

1.3 Contributions

In this line of work, we propose to overhaul the pull-crowdsourcing scheme by taking control of the distribution process of tasks submitted by multiple requesters. In [20], we have proposed a push-crowdsourcing architecture that automatically assigns tasks to workers. Such architecture allows us to order the execution of the tasks and to apply scheduling strategies tailored to the unpredictable nature of the humans. The focus was on analyzing human factors in task scheduling environment, and we investigated the following questions: "Can we apply scheduling algorithms to an online human workforce?" and "What are the adaptations required to schedule micro-tasks on crowdsourcing platforms successfully?".

In the present work, we revisit our push-crowdsourcing architecture by providing a theoretical scheduling framework that supports productions batches in systems with multiple requesters (aka tenants). We first review our previous results concerning human factors in scheduling environments. Next, and while we do not support explicit soft or hard deadlines, we make production batches "deadline-aware", that is by giving them priority to *complete* before concurrent jobs. Finally, in addition to previous results reported on a real crowd, we perform an empirical crowd-simulation to test the viability of our proposed solutions on a realistic scale.

In our empirical evaluation, we vary the size of the crowd, the ordering, priority, as well as the size of the batches. We take into account the characteristics of crowd workers such as the effects of context switching and work continuity to devise our *crowd-aware* scheduling algorithms. Our experimental settings include i) a controlled setup with a fixed number of workers involved in the experiments, ii) a real-world setup with varying number of workers and workloads taken from a real crowdsourcing platform, as well as iii) a simulation with realistic parameters demonstrating the ability of our scheduling algorithm to handle simultaneously best-effort batches and production batches that require a boost in execution speed. The results of our evaluations indicate that using scheduling approaches for micro-task crowdsourcing minimizes the overall latency of the batches, gracefully balances the workload, and significantly improves the productivity of the workers measured as their average execution time.

In summary, the main contributions of this paper are:

- A crowdsourcing architecture that enables task scheduling;
- A scheduling system that assigns pending tasks to available workers;
- A theoretical scheduling framework and algorithms that optimize for both fairness and deadline-awareness among tasks;
- A delayed scheduling mechanism that lessens the effects of context switching caused to workers when exposed to series of heterogeneous tasks;
- A series of experiments with real crowd workers to study the factors that impact human task scheduling;
- An extensive empirical evaluation using a crowd simulator to analyze the properties of the proposed algorithms.

The rest of the paper is structured as follows. We introduce our new architecture in Section 2 with the different design requirements that we aim for. In Section 3, we provide a formal problem definition and different scheduling algorithms that tackle fairness, deadline-awareness, and worker-consciousness. Section 4 presents the results of our extensive experimental evaluation of the proposed techniques. Section 5 gives a brief overview of current approaches in crowd-powered systems and micro-task crowdsourcing, and why they motivated our investigation. Finally, we highlight some real-world aspects that should be taken into account when using task scheduling in Section 6, before we conclude the paper in Section 7.

2 THE ARCHITECTURE OF MULTI-TENANT CROWD-POWERED SYSTEMS

We start by describing a task scheduling architecture that works on top of an existing crowdsourcing platform. This system engineering step is necessary when we do not have access to a dedicated crowd.

2.1 Setup

We utilize AMT as a medium to create a virtual pool of workers. To do so, we introduce the concept of `HIT-BUNDLE`, that is, a batch container where heterogeneous tasks of comparable complexity and reward get published. The `HIT-BUNDLE` acts as a container for a stream of tasks, which allows us to apply custom scheduling strategies. Furthermore, we assume a single crowd-powered system with an administrator and multiple users (tenants) whose tasks get pushed through the system and onto the backlog of the `HIT-BUNDLE`.

2.2 Architecture

Our general framework is depicted in Figure 2. The input to our system comes from the different crowdsourced queries submitted through a crowdsourcing interface. A `Crowdsourcing Decision`

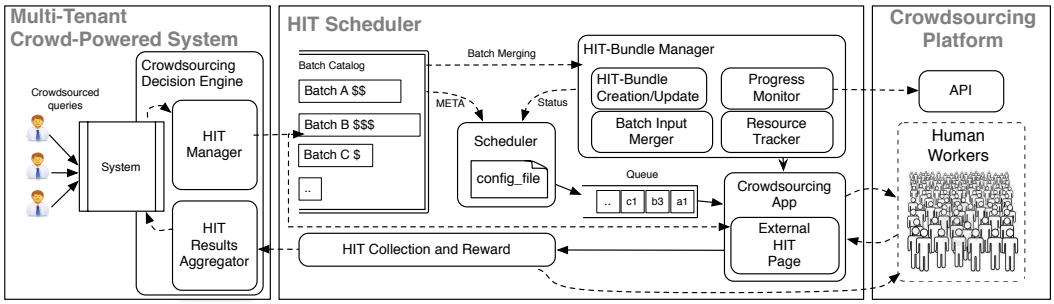


Fig. 2. The role of the HIT Scheduler in a Multi-Tenant Crowd-Powered System Architecture.

`Engine` takes the role of extracting the parts of the queries (and their input) to crowdsource. Subsequently, the `HIT-Manager` generates HIT batches together with a given monetary budget, and passes its requests to the `HIT-Scheduler`. This last step differentiate us from traditional crowd-powered systems, where batches are directly sent to the crowdsourcing platform.

The `HIT-Scheduler` aims at improving the execution time of selected HITs. Once new HIT batches are generated, they are put in a container of tasks to be crowdsourced. The scheduler is constantly monitoring the progress of the work through AMT's Application Programming Interface (API) and assigning dynamically the next HIT to the next available worker based on a scheduling algorithm. More specifically, the HIT Scheduler collects in its `Batch Catalog` the set of HIT batches generated by the `HIT-Manager` together with their reward and priorities.

Finally, the `HIT-BUNDLE Manager` creates crowdsourcing batches on AMT. Based on the scheduling algorithm adopted, a HIT queue (specifying which HIT must be served next in the `HIT-BUNDLE`) is generated and periodically updated. As soon as a worker is available, the HIT Scheduler serves the first element in the queue. When HITs are completed, the results are collected and sent back to the system for aggregation, merging and forwards the final results to the end-users.

Workers are may accept, skip, or return HITs they decide not to complete. The workers can also leave the system at any point in time. In these cases, the scheduler reacts accordingly by updating its queue.

2.3 HIT Scheduling Requirement Analysis

Next, we formulate a set of design requirements that we aim to achieve with our crowd scheduler.

(R1) Runtime Scalability: Unlike parallel schedulers, where the compiled query plan dictates where and when the operators should be executed, crowd-powered systems are bound to adopt a runtime scheduler that i) dynamically adapts to the current availability of the crowd, and ii) scales to make realtime scheduling decisions as the work demand grows higher.

(R2) Fairness: An important feature that any shared system should provide is fairness across the users of the system. By taking control of task scheduling, the scheduler acts as the load balancer of tasks across the available workforce. With this capability, the scheduler should ensure that large and small batches are treated alike.

(R3) Deadline-Awareness: Production batches may have a deadline associated with them. It is, however, often unclear whether missing a deadline of a crowdsourcing batch leads to batch failures, e.g., completing 90% of a crowdsourced workflow might still be useful. Therefore, we introduce a deadline-awareness requirement, where the goal is to give production batches priority towards the finish line.

(R4) Priority: In a multi-tenant system, some queries have a higher priority than others. For this reason, the tasks generated from the queries should be scheduled according to their priority. In a crowdsourcing scheduling setting, as workers are not committed to the platform and can leave at any point in time, the scheduler should be best-effort, that is, the system should do its best to meet the priority requirements without any hard guarantee.

(R5) Worker Conscious: In contrast to CPUs, human performance is impacted by many factors including: bias, priming, boredom, fatigue, etc. Scheduling approaches over the crowd should take these factors into account. In this paper, we experimentally test worker-conscious scheduling approaches that aim at balancing the trade-off between serving similar tasks to workers and providing fair execution to different batches.

Next, we will briefly revisit common scheduling approaches found in computing environments like Apache Hadoop or High-Performance Computing (HPC) clusters. We also provide an early discussion of the advantages and drawbacks of scheduling algorithms when applied to the crowd.

2.4 Basic Space-Sharing Schedulers

Crowdsourcing platforms usually operate in a non-preemptive mode, that is, they do not interrupt a worker performing a task of low priority to have her perform a task of higher priority. We consider common algorithms where a crowd worker is assigned a task until it is finished, returned or abandoned.

2.4.1 First-in, First-out (FIFO). On crowdsourcing platforms, this scheduling has the effect of serving lists of tasks of the same batch to the workers until they are finished. By concentrating the entire workforce on a single batch until it is done, FIFO provides the best throughput *per batch*[1] one can expect from the platform at a given moment in time, *after the batch has started.*

The potential shortcomings of this scheme are as follows: i) short batches and high priority batches can get stuck behind long-running tasks, minimizing the overall efficiency of the crowdsourcing system, and ii) when a batch has a large number of tasks, assigned workers can potentially get bored [44].

2.4.2 Shortest Job First (SJF). Other simple scheduling schemes offer different trade-offs depending on the requirements of the multi-tenant system. Shortest Job First (SJF) offers fast turn-around for short batches, and can lead to a minimum of a context switch for part of the crowd since the shortest batches are either quickly finished or scheduled to the first available workers. In detail, with this approach, the next served task is the one with the least expected time to complete (as estimated by the system and indicated by the requester).

However, SJF is not *strategy-proof* on current crowdsourcing platforms as the requesters can lie about the expected HIT execution times. Hence, these schemes should be used in trusted settings mostly (e.g., in enterprise crowd-Database Management Systems (DBMSs)). Moreover, these schemes do not systematically interweave tasks from different batches, and thus, present also the same shortcomings as FIFO.

2.4.3 Earliest Deadline First (EDF). Used mainly in real-time operating systems application, an Earliest Deadline First scheduler assigns dynamically a worker to the batch with earlier deadline[2] [10] . EDF guarantees the best performances if the resources are enough, but suffers from fairness problems and from pathological cases when a new batch with a short deadline arrives, potentially disrupting the previous batches in the queue. We will consider this algorithm in our simulation of batches with deadlines.

[1]The number of HITs completed per unit of time for a specific batch.
[2]A deadline is the the latest time by which the batch should be completed.

2.4.4 Round Robin (RR). The previous schemes do not meet our fairness requirement, in the sense that they give an advantage to one batch over the others. Round Robin resolves this issue by assigning HITs from batches in a cyclic fashion. In this way, all the batches are guaranteed to make regular progress. While Round Robin ensures an even distribution of the workforce and avoids starvation, it does not meet one of our requirement (R4) since it is not priority-aware: All the batches are treated equally with the side effect that batches with short HITs would (proportionally) get more workforce than longer HITs. Another risk is that a worker might find herself bouncing across tasks and being forced to continuously switch context, hence loosing time to understand the specific instructions of the tasks. The negative effect of a context switch is evident from our experimental results (see Section 4) and should be avoided.

In the next section, we introduce more sophisticated scheduling algorithms that meet some or all of our design requirements.

3 HIT SCHEDULING MODELS

We start by formalizing the problem of task scheduling on a crowdsourcing platform. Then we propose new scheduling algorithms adapted to the case of a shared crowd workforce.

3.1 Problem Definition

When a new request is submitted to the system it generates a batch b of HITs. We define a batch as a set containing a total of T_b HITs to be crowdsourced. Each batch batch has additional metadata attached to it: a monetary budget M_b and a priority value P_b. We assume that the priority value is proportional to the price and is controlled by and administrator. Batches with higher priority should be executed before batches with lower priority. If a high-priority batch is submitted to the platform while a low-priority batch is still uncompleted, the HITs from the high-priority batch are to be scheduled to run first.

A scheduling algorithm takes as input a set of N batches currently queued $\{b_1, .., b_N\}$, and a set $\mathcal{W}$ of crowd of workers $\{w_1, .., w_m\}$ currently active on the platform, the algorithm mainly computes an ordered list of batches indexed. And, when a worker w_i is available, the system assigns a task from the list of ordered batches.

At any moment in time and a given batch b, let L_b be the number of remaining HITs (currently running or not yet started); R_b the number of currently running HITs but not yet completed; C_b the number of completed HITs. Additionally, let $\mathcal{B}$ the set of non-empty batches (batches with $C_b < T_b$); and $\mathcal{A}_b$ the set of workers currently assigned tasks to a batch b. Note that: $|\mathcal{A}_b| = R_b$.

We introduce $\mathbf{x} \in \mathbb{R}^{\mathcal{B}}$ to be the *weighted workforce vector* at any given time. For each batch $b \in \mathcal{B}$, the component x_b of the vector represents a function of the number of active workers R_b and, when applicable, the number of unassigned tasks L_b. A scheduling algorithm will then minimize the cost of a joint combination of the workforce vector components. For example, when $x_b = \frac{R_b}{T_b}$, the workforce vector is simply the vector where each component represents the proportion of active workers currently assigned to batch b.

Finally, let $\boldsymbol{X}$ be the set of all feasible vectors resulting from an assignment decision that scheduler can make at run time i.e., assigning a task from a batch b_i to worker w_j. In other words, it is the set of possible ways to allocate the available workers to the uncompleted batches, keeping in account the constraint of number of tasks left per batch and available workforce. We will show in the following how choosing different workforce functions will affect the behavior of the scheduling algorithms.

3.2 Fair Schedulers

In order to deal with batches of HITs having different priorities while avoiding starvation, we build on techniques used in cluster computing. Sharing heterogeneous resources across tasks having different priorities is a well-known problem, one popular approach is Fair Scheduling (FS) [25]. In this work, we expand on our approach introduced in [20] and propose a universal model of fairness, one that can directly embed several aspects defined in our requirements described in Section 2.3.

Following the work of Lan et al. [33] on the axiomatic theory of fairness, all different fairness indices can be represented by a universal index $J_\beta(\mathbf{x})$ of the weighted workforce vector $\mathbf{x}$, parametrized with the real β, as:

$$J_\beta(\mathbf{x}) = \text{sign}(1-\beta)\left[\sum_{b\in\mathcal{B}}\left(\frac{x_b}{\sum\limits_{c\in\mathcal{B}} x_c}\right)^{1-\beta}\right]^{\frac{1}{\beta}} \tag{1}$$

This fairness index corresponds to Jain's fairness index [30] when $\beta = -1$, and it favours more fair (and potentially less efficient) allocations as β grows. The limit for $\beta \to \infty$ will seek for fairness only, disregarding efficiency completely, obtaining as fairness measure:

$$J_\infty(\mathbf{x}) = -\max_{b\in\mathcal{B}} \frac{\sum\limits_{c\in\mathcal{B}} x_c}{x_b} \tag{2}$$

We can represent various scheduling algorithms as maximization of the fairness index using different functions to define the vector $\mathbf{x}$ and varying β. The scheduling algorithm will have to assign the available worker to maximize J_β , i.e., $\mathbf{x} = \arg\max_{\mathbf{x}\in\boldsymbol{X}} J_\beta(\mathbf{x})$. Subsequently, we will focus on the fairness index J_∞, changing only vector $\mathbf{x}$ to achieve different fairness goals. While the cost of computing such index is negligible, it is important to notice that to be able to optimize the index over time, it is necessary to evaluate the whole set $\boldsymbol{X}$ every time a group of workers become ready. This might not be feasible because the cardinality of such a set grows linearly with the number of batches, and exponentially with the number of workers available at the same time[3]. Luckily, in practice we can achieve a good approximation for J_∞ by choosing the batch with the lowest workforce value and performing the scheduling one worker at a time. We will compare the two approaches in Section 4.6.

3.2.1 Fair Scheduling (FS). In our previous work [20], we proposed a Fair Scheduler that operates with the following strategy: Whenever a worker is available, assign a HIT from the batch with the lowest number of currently assigned HITs R_b. Unlike Round Robin, this ensures that all the batches continuously get a non-zero amount of resources. Algorithm 1 gives a pseudo code of this strategy.

In our universal model, when it is not feasible to evaluate the index over the whole $\boldsymbol{X}$, by setting $\beta \to \infty$ and $x_b = \frac{R_b}{T_b}$, assigning the worker to the batch with less active workers ratio is the best approximation of $\arg\max_{\mathbf{x}\in\boldsymbol{X}} J_\infty(\mathbf{x})$.

3.2.2 Weighted Fair Scheduling (WFS). In order to schedule batches with higher priority first (see R2 in Section 2.3), we use weighted fair scheduling to assign a task from the batches with the least $\frac{R_b}{P_b}$ value. Algorithm 1 (Line 2) gets in that case updated to: `Sort` $\mathcal{B}$ `by increasing` $\frac{R_b}{P_b}$. Intuitively, this gives higher priority to batches with fewer running tasks and a high priority value. The normalized fair share of resources (i.e., number of active crowd workers) allocated to each batch in $\mathcal{B}$ is given by $\frac{P_b}{\sum_1^N P_i}$.

[3]This exponential growth will not happen if the selection is done continuously as a new worker becomes free: it is thus problematic only at the start of a scheduling process.

Algorithm 1 Basic Fair Sharing

Input: $\mathcal{B} = \{b_1, \ldots, b_N\}$ set of batches currently queued, each with:
- priority P_b,
- initial number of HITs per batch T_b,
- number of assigned tasks R_b,
- number of unassigned tasks per batch L_b.

Output: HIT h.

When a worker is available for a task
$\mathcal{B}_{Sorted}$ = Sort $\mathcal{B}$ by increasing R_i
$h = \mathcal{B}_{Sorted}$.top.getHit()
return h

Algorithm 2 Generalized Deadline-Aware β-Fair Scheduling

Input: $\mathcal{B} = \{b_1, \ldots, b_N\}$ set of batches currently queued, each with:
- priority P_b,
- initial number of HITs per batch T_b,
- number of assigned tasks R_b,
- number of unassigned tasks per batch L_b,
- fairness parameter $\beta \in [0, \infty]$,
- weighted workforce function vector x_b e.g., DAFS $x_b = \frac{1}{P_b T_b} \sum_{i=0}^{R_b - 1} f\left(\frac{L_b - i}{T_b}\right)$

Output: HIT h.

When a set of workers is available for a task
compute the set $\boldsymbol{X}$ of feasible actions
$\mathbf{x} = \arg\max_{\mathbf{x} \in \boldsymbol{X}} J_\beta(\mathbf{x})$
$b = \arg\min_{b \in B} x_b$
return b.getHit()

In our universal model, when it is not feasible to evaluate the index over the whole set $\boldsymbol{X}$, by setting $\beta \to \infty$ and $x_b = \frac{R_b}{P_b T_b}$, then maximizing $J_\infty(\mathbf{x})$ is equivalent to fair weighted scheduling.

3.3 Deadline-Aware Fair Scheduling (DAFS)

While weighted fair scheduling weighs each task in a batch equally, we might want to give a higher priority to the tasks as we progress in a batch, in order to ensure that a production batch will be more likely to be served than a best-effort one. Moreover, we want such per-task priority to be low at the beginning of a batch when few workers are involved, and to grow as more workers get involved in the batch. In this way, a new batch that just entered the system will not overtake batches that are almost finished, solving a typical problem of EDF schedulers.

A natural way to extend WFS is to consider each component of the workforce vector x_b as a sum over the number of running tasks. Moreover, we consider a calibration function f, which will be decreasing as the batch gets closer to completion. Choosing:

$$x_b = \frac{1}{P_b T_b} \sum_{i=0}^{R_b - 1} f\left(\frac{L_b - i}{T_b}\right) \tag{3}$$

allows us to introduce a Deadline-Aware Fair Scheduler (DAFS) by maximizing $J_\infty(\mathbf{x})$. This scheduler can work seamlessly in combination with best-effort batches, for example by simply using x_b as defined in Section 3.2.2 for best-effort batches. A possible choice for f is the identity function, or even a logarithmic function to obtain a more dramatic effect when the number of running tasks approaches T_b. Unless otherwise specified, in the rest of the work we will use the latter. DAFS is thus a particular case of our Generalized Deadline-Aware β-Fair Scheduling algorithms family, described in Algorithm 2, choosing $\beta \to \infty$.

We also define a Simplified Deadline-Aware Fair Scheduler (SDAFS) that is particularly useful when computing the whole $\boldsymbol{X}$ is not feasible. By choosing the batch assignment with lower x_b at each scheduling decision, as shown in Algorithm 3, and as before choosing $\beta \to \infty$. We compare the performance of the various schedulers in detail in Section 4.6, where we also evaluate the effect of this simplification.

Algorithm 3 Simplified Deadline-Aware β-Fair Scheduling

Input: $\mathcal{B} = \{b_1, \ldots, b_N\}$ set of batches currently queued, each with:

- priority P_b,
- initial number of HITs per batch T_b,
- number of assigned tasks R_b,
- number of unassigned tasks per batch L_b,
- fairness parameter $\beta \in [0, \boldsymbol{\infty}]$,
- weighted workforce function vector x_b, e.g.:

  FS $x_b = \frac{R_b}{T_b}$

  WFS $x_b = \frac{R_b}{P_b T_b}$

  SDAFS $x_b = \frac{1}{P_b T_b} \sum_{i=0}^{R_b - 1} f\left(\frac{L_b - i}{T_b}\right)$

Output: HIT h.

1: When one worker is available for a task
2: $b = \arg\min_{b \in B} x_b$
3: **return** b.getHit()

3.4 Worker Conscious Fair Scheduler (WCFS)

From a worker perspective, scheduling can lead to randomly alternating task types that a single worker might receive. In such a situation, the worker has to adapt to the new task instructions, interface, question, and the like, and this could be penalizing (see the related work section below 5.4). This overhead is called *context switch*. Our goal is to avoid as much as possible that a worker jumps back and forth between different tasks. We propose a Worker Conscious Fair Scheduler (WCFS) that increases the chances that a worker receives a task from a batch that he completed recently. In order to preserve the properties of previous algorithms, we employ a light weight modification that achieves this goal. In detail, when selected, a batch may concede its turn if the available worker has been doing other types of tasks recently. Each batch can concede his turn up to K times, a predefined concession threshold, which is reset to zero every time a task from this batch is assigned. This approach is the equivalent of Delay Scheduling [52].

4 EXPERIMENTAL EVALUATION

We describe in the following our experimental setting and results. As a general experimental setup, we implemented the architecture described in Section 2 on top of AMT. To test the scalability

Algorithm 4 Worker Conscious Fair Scheduler

Input: $\mathcal{B} = \{b_1, \ldots, b_N\}$ set of batches currently queued, each with:
- priority P_b,
- initial number of HITs per batch T_b,
- number of assigned tasks R_b,
- number of unassigned tasks per batch L_b,
- number concessions K_b initialized to 0.

Input: $\mathcal{W} = \{w_1, .., w_m\}$ set of active workers, each with:
- l_j is the last batch that w_j worked on initialized to null.

Input: k = maximum number of allowed concessions.
Output: HIT h.

When a worker w_j is available for a HIT do:
$\mathcal{B}_{Sorted}$ = Sort B by increasing $\frac{R_b}{P_b}$
for b in $\mathcal{B}_{Sorted}$ **do**
  if $(l_j == b)$ or $(l_j == Null)$ **then**
    $K_b = 0$
    $l_j = b$
    return b.getHit()
  else if $K_b < k$ **then**
    $K_b = K_b + 1$
    continue
  else
    $K_b = 0$
    $l_j = b$
    return b.getHit()
  end if
end for

of our methods, we also resorted to simulated experiments. Our implementation and datasets are available as an open-source project for reproducibility purposes and as a basis for potential extensions: https://github.com/XI-lab/HIT-Scheduler.

4.1 Datasets

For our experiments, we used a dataset composed of 7 batches of varying complexity, size, and reference price. The data was partly created by us and partly collected from related works; and overall includes a representative mix of crowdsourcing tasks. Table 1 gives a summary of our dataset and provides a short description and references when applicable. We note that for the purpose of our experiments, we vary the batch sizes and prices according to the setup.

4.2 Micro Benchmarks

The goal of the following micro-benchmark experiments is to validate some of the hypotheses that motivate the use of a `HIT-BUNDLE` and the design of a worker-aware scheduling algorithm that minimizes tasks switching for the crowd workers.

4.2.1 Batch Split-up. The first question we try to answer is whether smaller or larger batches of homogeneous HITs are more attractive to the workers on AMT. We experimentally check if a single

ID	Dataset	Description	Price per HIT	#HITs	Avg. Time per HIT
B1	Customer Care Phone Number Search	Find the customer-care phone number of a given US-based company using the Web.	$0.07	50	75sec
B2	Image Tagging	Type all the relevant keywords related to a picture from the ESP game dataset. [48]	$0.02	50	40sec
B3	Sentiment Analysis	Classify the expressed sentiment of a product review (positive, negative, neutral).	$0.05	200	22sec
B4	Type a Short Text	This a is study on short memory, where a worker is presented with text for a few seconds, then he is asked to type it from memory. [47]	$0.03	100	11sec
B5	Proof Reading	A collection of short paragraphs to proof-read from StackExchange.	$0.03	100	36sec
B6	Butterfly Image Classification	Classify a butterfly image to one of 6 species (Admiral, Black Swallowtail, Machaon, Monarch, Peacock, and Zebra). [36]	$0.01	600	15sec
B7	Item Matching	Uniquely identify products that can be referred to by different names (e.g., 'iPad Two' and 'iPad2nd Generation'). [50]	$0.01	96	22sec

Table 1. Description of the batches constituting the dataset used in our experiments.

large batch executes faster than when breaking the same batch into smaller ones. To this end, we use the batch B6 which we split into 1, 10 and 60 individual batches, containing respectively 600, 60 and 10 HITs each. Next, we run all these batches on AMT concurrently with non-indicative titles and similar unit prices of $0.01. Note that the batch combinations were published at the same time on the crowdsourcing platform to have all the variables (like crowd population and size, concurrent requesters, and rewards) the same across the different settings.

Figure 3 shows how the three different batch splitting strategies executed over time on B6. We observe that running B6 as one large batch of 600 HITs completed first. We also observe that the strategy with 10 batches only really kicks-off when the large batch finishes (and similarly for the strategy with 60 batches). From this experiment, we conclude that larger batches provide better throughput and constitute a better organizational strategy. This finding is especially interesting for requesters who would periodically run queries that use a common crowdsourcing operator (albeit, with a different input), by pushing new HITs into an existing `HIT-BUNDLE`.

4.2.2 Merging Heterogenous Batches. We extend the above experiment to compare the execution of two heterogenous batches run separately or within a single `HIT-BUNDLE`. Unlike the previous experiment, where the fine-grained batches were one to two orders of magnitude smaller than the larger one, this scenario involves two batches of type B6 and B7 containing 96 HITs each, versus one `HIT-BUNDLE` regrouping all 192 HITs. We run the three batches concurrently on AMT, with non-indicative titles and similar unit prices of $0.01 and without altering the default serving order

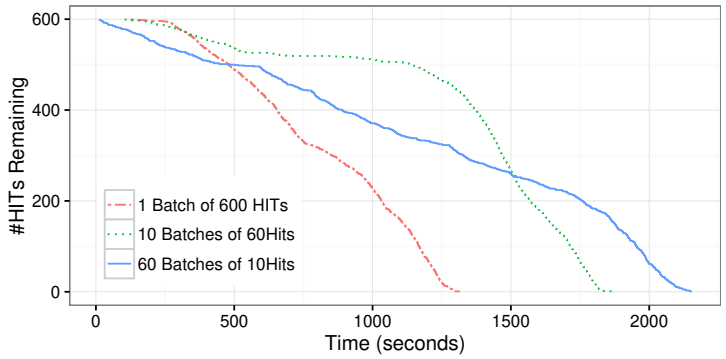


Fig. 3. A performance comparison of batch execution time using different grouping strategies publishing a large batch of 600 HITs vs smaller batches (From B6).

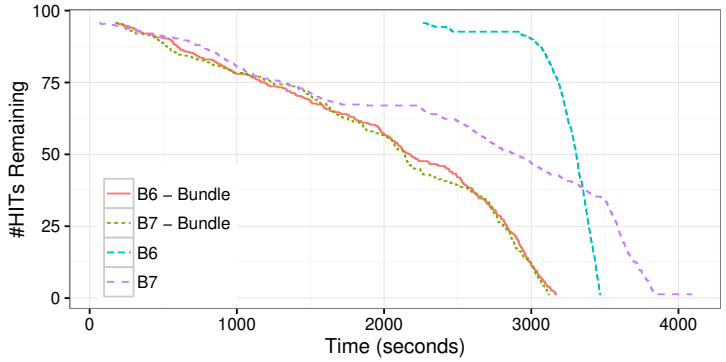


Fig. 4. A performance comparison of batch execution time using different grouping strategies publishing two distinct batches of 192 HITs separately vs combined inside a HIT-BUNDLE.

within the HIT-BUNDLE (We observe that AMT randomly selects the input to serve). The results are depicted in Figure 4.

Again, the HIT-BUNDLE exhibits a faster throughput as compared to individual batches. Moreover, the embedded batches both finish before their counterparts that are running separately.

At this point, we have shown that requesters who would run queries invoking different crowdsourcing operators can also benefit from pushing their HITs into the same HIT-BUNDLE. Since a system might support multiple crowdsourcing operators, the next question we explore is whether context switches (i.e., alternating HIT types) affects workers efficiency.

4.2.3 Workers' Sensitivity to Context Switch. The following experimental setup involves three groups of 24 distinct workers each. Each group was exposed to three types of HIT serving strategies, namely:

— *RR*: a worker in this group would receive tasks in alternating order from batches B6 and B7.
— *SEQ10*: here the workers will receive 10 tasks from B6 then 10 tasks from B7 then again 10 from B6 and so on.
— *SEQ25*: similar to *SEQ10* but with sequences of 25 tasks. To cause the context switch for each participant, the workers were asked to do at least 10, and up to 100, tasks.

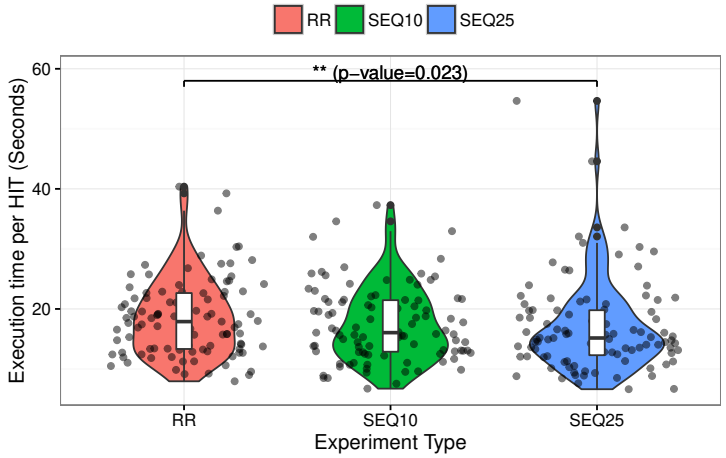


Fig. 5. Average Execution time for each HIT submitted from the experimental groups RR, SEQ10 and SEQ25.

Figure 5 shows the distribution (with violin plot and boxplot) of the per-user average execution time of all the 100 HITs under each execution group. We observe that the HITs average execution time is worst when using RR as compared to workers performing longer alternating sequences in SEQ10 and SEQ25. To test the statistical significance of these improvements, and since the distribution of HIT execution time cannot be assumed to be normally distributed, we perform a Willcoxon signed-rank test. SEQ10 has a p=0.09 which is not enough to achieve statistical significance. However, the SEQ25 improvement over RR is statistically significant with p<0.05.

In conclusion, context switch generates a significant slowdown for the workers, thus reducing their overall efficiency. Hence, this result motivates the design of a scheduling algorithm that takes into account workers efficiency by scheduling more extended sequences of HITs of the same type.

4.3 Scheduling HITs for the Crowd

We now move our attention to experimentally comparing the scheduling algorithms that are used to manage the distribution of HITs within a `HIT-BUNDLE`.

4.3.1 Controlled Experimental Setup. To develop a clear understanding of the properties of classical scheduling algorithms when applied to crowdsourcing, we put in place an experimental setup that mitigates the effects of workforce variability over time.

In our controlled setting, each experiment that we run involves a number of crowd workers ranging between: $Min_w \leq |\text{workforce}| \leq Max_w$, at any point in time. To be within this range target, the workers who arrive first are presented with a reCaptcha to solve (paid $0.01 each), until Min_w workers join the system, at that point, the experiment begins serving tasks. From that point on, new workers are still accepted up to a maximum Max_w. If the number of active sessions drops below Min_w, then the system starts accepting new workers again.

Unless otherwise stated, we use the following configuration:

- Number of workers: $10 \leq |\text{workforce}| \leq 15$.
- Weighted Fair Sharing, with price as weighting factor.
- a `HIT-BUNDLE` composed of $\{B1, B2, B3, B4, B5\}$.
- FIFO order is $[B1, B2, B3, B4, B5]$.

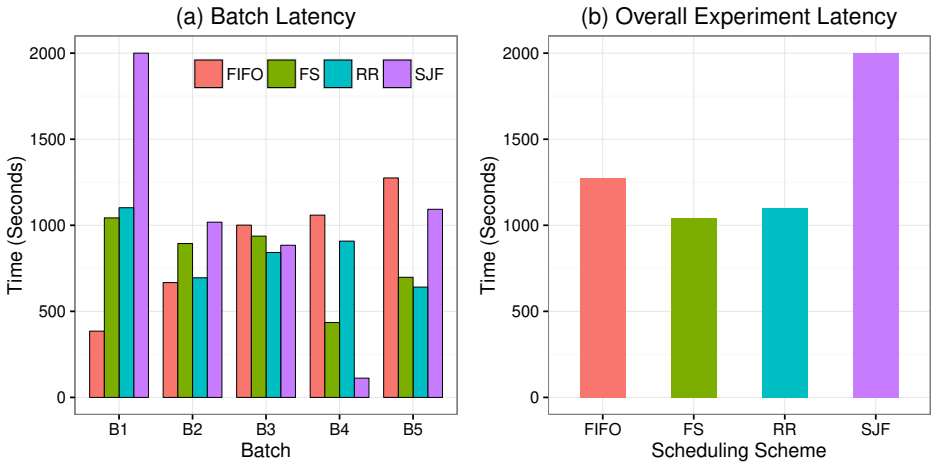


Fig. 6. Scheduling approaches applied to the crowd.

— SJF order is $[B4, B3, B5, B2, B1]$.

Also, we note that each experiment involves a distinct crowd of workers to avoid any further training effects on the tasks.

4.3.2 Comparing Scheduling Algorithms. First, we compare how different scheduling algorithms perform from a latency point of view, taking into account the results of individual batches as well as the overall performance. We create a `HIT-BUNDLE` out of {B1,B2,B3,B4,B5}, which is then published to AMT. In each run, we use a different scheduling algorithm from: FIFO, FS, RR, and SJF, with $10 \leq |\text{workforce}| \leq 15$.

Figure 6 shows the completion time of each batch in our experimental setting and the cumulative execution time of the whole `HIT-BUNDLE`.

FS achieved the best overall performance, thus maximizing the system utility, though, at the batch level, FS did not always win (e.g., for B2). We see how FIFO assigns tasks from a batch until it is completed. In our setup, we used the predefined order of the batches, which explains why B1 is getting preferential treatment as compared to B5, which finishes last. Similarly, SJF performs unfairly over all the batches but manages to get B4 completed extremely fast. In fact, SJF uses statistics collected from the system on the execution speed of each operator (see Table 1); this explains the fast execution of B4. On the positive side, we observe that both RR and FS perform best in terms of fairness with respect to the different batches, i.e., there was no preferential treatment.

4.3.3 Varying the Control Factors. To test our priority control mechanism across different batches of a `HIT-BUNDLE` (tuned using the price), we run an experiment with the same setup as in Section 4.3.2, but varying the price attached to B2 and using the FS algorithm only. Figure 7a shows that batches with a higher priority (reward) lead to faster completion times using the FS scheduling approach (green bar of B2 lower than the red one). This comes at the expense of other batches being completed later.

Another dimension that we vary is the crowd size. Figure 7b shows the batch completion time of two different crowdsourcing experiments when we vary the crowd size from $10 \leq |\text{workforce}| \leq 15$ to $20 \leq |\text{workforce}| \leq 25$ (keeping all other settings constant). We can see batches being completed faster when more workers are involved. However, different batches obtain different levels of improvement.

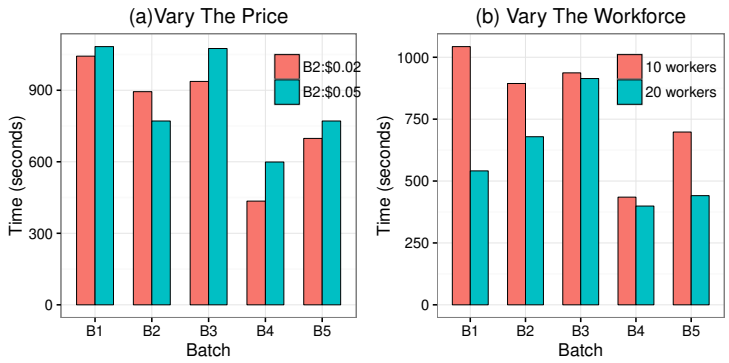


Fig. 7. (a) Effect of increasing B2 priority on batch execution time. (b) Effect of varying the number of crowd workers involved in the completion of the HIT batches for the FS algorithm.

4.4 Live Deployment Evaluation

After the initial evaluation of the different dimensions involved in scheduling HITs over the crowd, we now evaluate our proposed fair scheduling techniques FS and WCFS in an uncontrolled crowdsourcing setting using HIT-BUNDLE, and compare it against a standard AMT execution.

More specifically, we create a workload that mimics a 1-hour activity on AMT from a requester who had 28 batches running concurrently. Since we do not have access to the input of the batches, we randomly select batches from all our experimental datasets and adapt the price and the size to the actual trace. The trace used in that sense is composed of 28 batches with similar rewards of $0.01; the largest batch has 45 HITs and the smallest 1 HIT only. For analysis purposes, we group batches by size: 16 *small* batches (1-9 HITs), 8 *medium* batches (9-15 HITs), and 4 *large* batches (16-45 HITs). The total size of this trace is 286 HITs.

4.4.1 Live Deployment Experimental Setup. We publish the 28 batches concurrently from the previously described trace as individual batches (standard approach) as well as into two HIT-BUNDLEs, one using FS and the other using WCFS. The individual batches use meaningful titles and descriptions of their associated HIT types; on the other hand, the HIT-BUNDLE informs the crowd workers that they might receive HITs from different categories. Other parameters like requester name and reward are similar.

4.4.2 Average Execution Time. Figure 8 shows the average HIT execution time obtained by the different setups. We observe that workers perform better when working on individual batches because of the missing context switch effect (though the performance difference is minimal and not statistically significant), in accordance with the results from Section 4.2.3. Instead, when HITs are scheduled, execution time increases with the benefit of prioritizing certain batches. We also see that WCFS provides a trade-off between letting workers work on the same type of HITs longer and having the ability to schedule batches fairly as we shall see next.

4.4.3 Results of the Live Deployment Run. We plot the CDFs of HIT completion per category in Figure 9. For example, 25% of small batches completed in 500 seconds when running individually. For all batch sizes, we observe that individual batches started faster. However, in all cases they also ended last, especially for smaller batches suffering from some starvation (i.e., a long period without progress); here, we see the benefits of both FS and WCFS at load balancing.

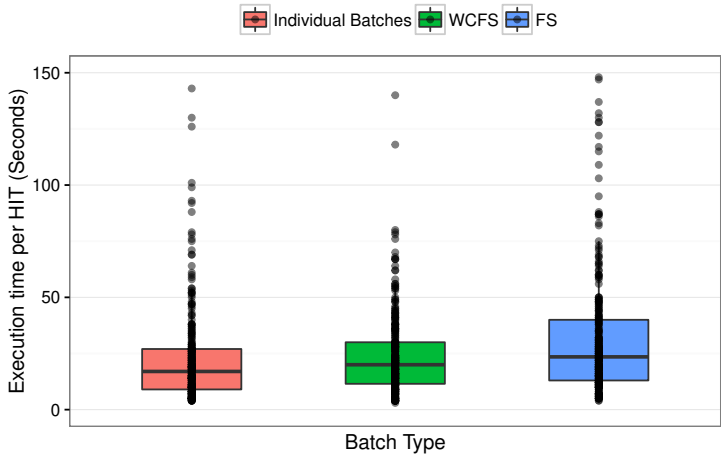


Fig. 8. Average execution time per HIT under different scheduling schemes.

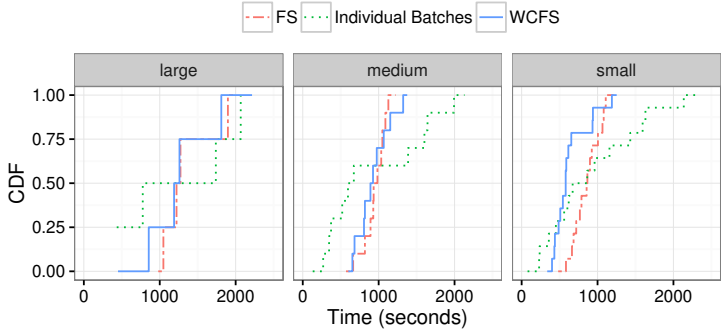


Fig. 9. CDF of different batch sizes and scheduling schemes.

The final plot (Figure 10) shows how a large workload executes over time on the crowdsourcing platform. We can see how many workers are involved in each setting and which HIT batch they are working on (each color represents a different batch). Finally, as expected, the number of active workers varied wildly over time in each setup. Corroborating the results of the previous paragraph, Individual Batches received more workforce in the beginning (they start faster) then workers either left or took some time to join the remaining batches in the [11:25 - 11:35] time period. Our main observation is that FS and WCFS i) achieve their desired property of load balancing the batches when there are a sufficient number of workers, ii) they finish all the batches well before the individual execution (10-15 minutes considering the 95th percentile).

4.5 Large Scale Crowd Scheduling Simulations

In the previous sections, we have presented the results of an experimental system that we have implemented on top of Amazon Mechanical Turk to allow push-crowdsourcing with real workers. The workers on the platform were presented tasks that the schedulers picked given a specific

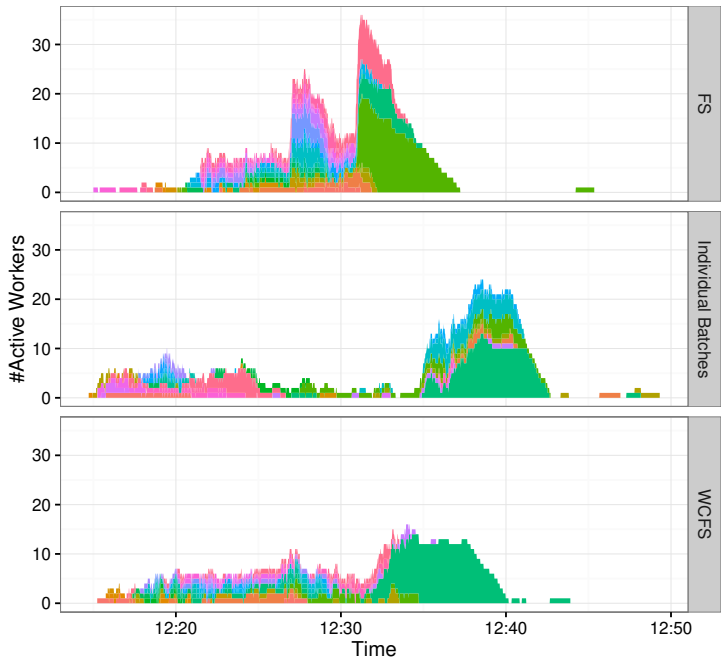


Fig. 10. Worker allocation with FS, WCFS and classical individual batches in a live deployment of a large workload derived from crowdsourcing platform logs. Each color represents a different batch.

strategy. To reach general conclusions about the properties, running a large number of experiments is necessary. For this purpose we have a implemented a simulator for crowd workers and tested our methods at scale in a simulated environment. Our simulator mimics the main property of crowd workers that is the uncertainty (quality and response time) associated with the responses. In particular, and since our primary focus is the efficiency of the overall system, we modeled the crowd worker as stochastic processes having a response time following a lognormal distribution. In the following, we keep the same task properties introduced in table 1, and vary the crowd size, schedulers and priorities. All the experiments were repeated 100 times.

4.5.1 Scheduling Algorithms in a Crowd Simulation. Like in section 4.3.2, we compare how different scheduling algorithms perform from a latency point of view, taking into account the results of individual batches, the overall performance as well as the effect of having a large size of available workers on the platform. We again use an experiment composed of batches {B1,B2,B3,B4,B5}. In each run, we use a different scheduling algorithm from FIFO, FS, RR, and SJF, with 10, 20, 50, 100 and 200 workers.

Figure 11 shows the completion time of each batch in our experimental setting (left panel) and the cumulative execution time (right panel). We recall that an ideal strategy is one that maximizes the overall utility of the system while avoiding batch starvation. First, we note that having a larger pool of simulated crowd workers yields a shorter execution time across the different experimental setups. This is due to higher parallelism, in fact, our largest configuration (i.e., 200 simulated workers) represents half of the available tasks.

Next, we observe that all the strategies finish the work approximately at the same time except for SJF which seems to perform significantly worse when we reduce the size of the workforce. This

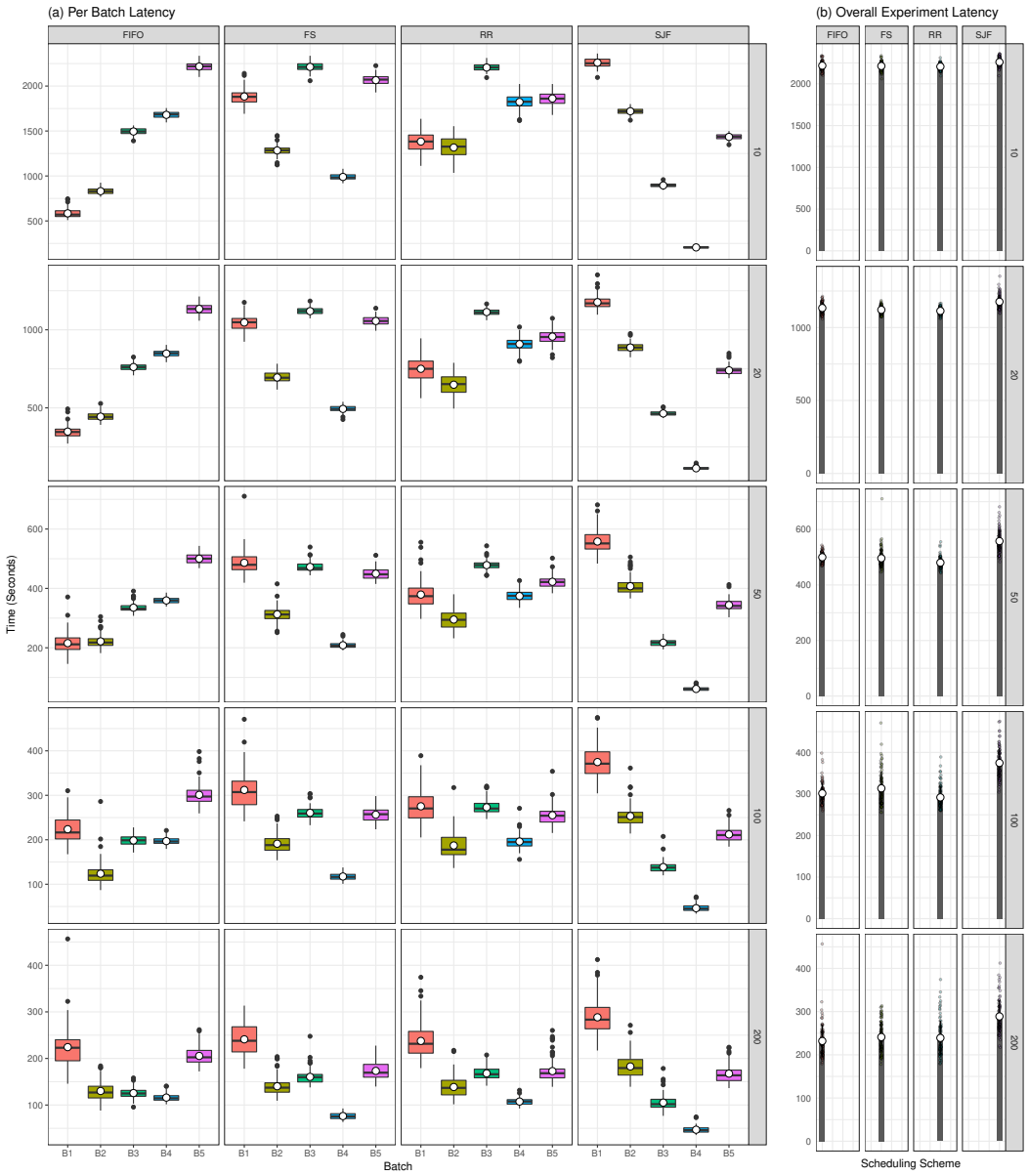


Fig. 11. Simulated scheduling approaches: completion time distribution of each batch in our experimental setting (left panel), and cumulative execution time (right panel).

corroborates the results seen in section 4.3.2. Finally, we observe how discriminatory strategies (such as FIFO and SJF) consistently give an advantage to their preferred batches. Again, we observe that both RR and FS perform best in terms of fairness to the different batches, i.e., there is no preferential treatment. It is worth noting that FS and RR have a much closer average execution time for the batches, with FS exhibiting a slightly higher variance than RR that is the result of auto balancing large and fast batches versus small and slow batches.

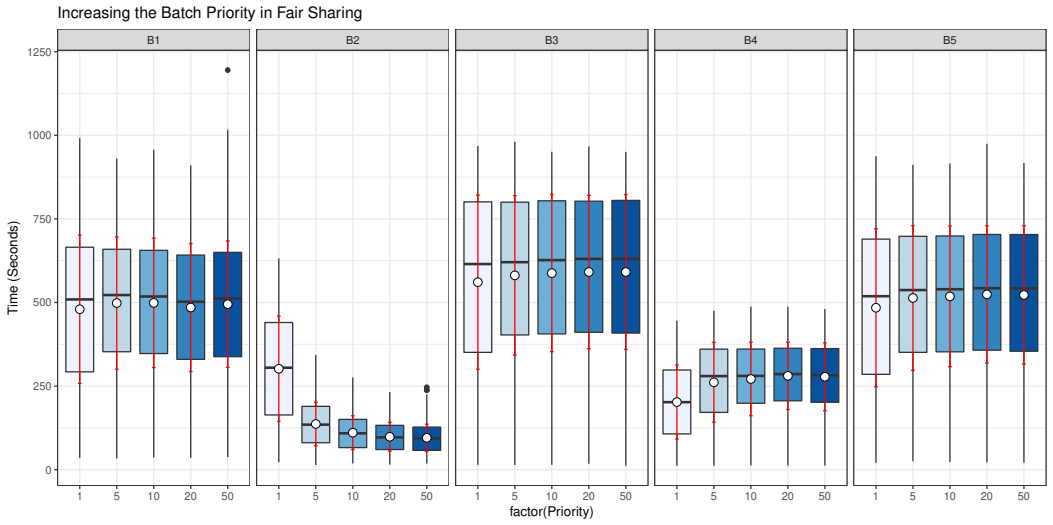


Fig. 12. Effects of increasing B2 priority on batch execution time. We can see how B2 gets better performance, while other batches collectively counterbalance the need for resources.

4.5.2 Tuning the Priority in Fair Sharing. The proof of concept results shown in section 4.3.3 demonstrated how a priority weight could play a role in pushing more tasks from a specific batch in a fair sharing environment. To confirm this observation in our simulation framework, we increased the priority of batch B2 from 1 to 50, while simulating 25 workers. Figure 12 shows the results of the experiment. Overall, we could confirm our previous observation. Fair sharing can be utilized as an effective mechanism to boost a specific batch gradually, while slightly degrading the performance of the other executions. In practice, a company administrator could be in charge of such performance tuning parameters. In an open market, the priority can be bound to a price for the additional service.

4.6 Large Scale Best-effort and Production Simulations

We now turn to larger scale experiments. As summarized in Table 2, we generated a set of 100 synthetic scenarios to evaluate DAFS, described in Section 3.3, and compared it with the other schedulers. We analyzed the behavior of the scheduler over a time window of 2400 seconds (unless otherwise specified), during which 200 workers arrive according to a homogeneous Poisson distribution [23] (the time window can be considered small enough to allows us to ignore non-homogeneity) and have an exponential distributed service time with an average $\tilde{s}$ of 11 seconds.

We set the simulator so that at the beginning of the time window, 1000 batches are already active. During the time window, an average of 80 more batches arrives each minute following a Poisson distribution, with some of them (the largest 10%) are production batches.

In order to choose a meaningful deadline horizon, we span from the smallest deadline (achievable when enough workers are available to allow full parallelization) to the largest one corresponding to the case when only one worker is available. More rigorously: $f(\alpha) = \alpha T_b(\tilde{s} - \frac{\tilde{s}}{W}) + (1-\alpha)\frac{\tilde{s}}{\min(W, T_b)}$, where W is an upper bound on the number of workers expected to be available in the time window. When $\alpha = 0$, we simulate full parallelization, linearly sweeping towards full serialization as α approaches 1 (and $\alpha > 1$ will mean an even longer deadline). In some scenarios, a large batch may arrive towards the end of the time window; this will cause that particular scenario to be unsatisfiable during the given time window, i.e., in some situations it is not possible to have all

Parameter	Values	Notes
Time window	2400s	–
Deadline parameter α	$[0-1]$	0 is full parallel 1 is full serial
Workers	200	Poisson arrivals
Service time	11s	Exponential
Pre-existing batches	1000	–
New batches	3200	Poisson arrivals deadline to longest 10%

Table 2. Synthetic scenarios parameters.

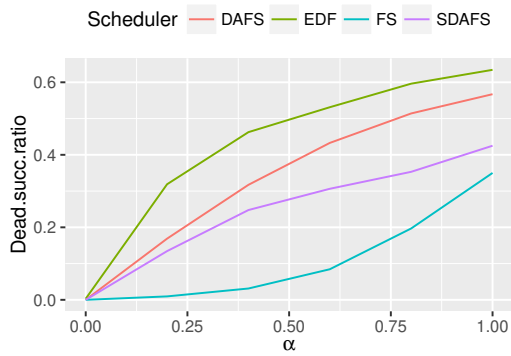


Fig. 13. Schedulers performance for synthetic scenarios varying deadline times parameter α.

deadlines satisfied during the given time window. This is a design choice, as we want to capture the dynamic behavior of batch arrivals. We ensure a fair comparison by repeating the experiments for all different scheduling algorithms with the same workers and batch arrivals for each scenario.

We use the following metrics to evaluate the different schedulers: for the batches deadlines, we consider the ratio of batches that are completed in time and the standard deviation of the completion of all batches as an indication of the completion disparity. For best-effort batches, we consider the mean and standard deviation of the completion level. The latter is a good indicator of global fairness. The instantaneous fairness index $J_\infty(x)$ is instead an indicator of how one scheduler behaves over time. We analyze such behavior in Section 4.6.2.

4.6.1 Performance. The performance of the various schedulers are shown in Figure 14, for the case where $\alpha = 0.4$; intermediate values of this parameter are especially interesting as they show how the schedulers behave when a solution is hard to find. We focus on fair schedulers first. The median deadline success ratio of the DAFS is ten times higher than the value of FS. Interestingly, this is obtained without sacrificing too much the performance of the best effort batches: using DAFS corresponds to a median loss of 5% in term of best effort batches completed compared to FS.

We notice that while EDF obtains the best performance in term of batches with deadlines, it is at the expense of a loss in performance for the best effort batches. Regarding fairness, the EDF scheduler has a standard deviation for best effort batches completion that is 8 times worse than the

	mean dead.	std dead.	mean b. effort	std b. effort	sim. time [s]
DAFS	0.968	0.001	0.943	0.018	22983.559
EDF	0.988	0.001	0.760	0.082	795.865
FS	0.972	0.115	0.999	0.012	682.937
SDAFS	0.883	0.001	0.888	0.050	750.643

Table 3. Schedulers batch completion statistics for synthetic scenarios: Mean completion ratio and standard deviation for deadline and best-effort batches together with the scheduling computation cost in seconds.

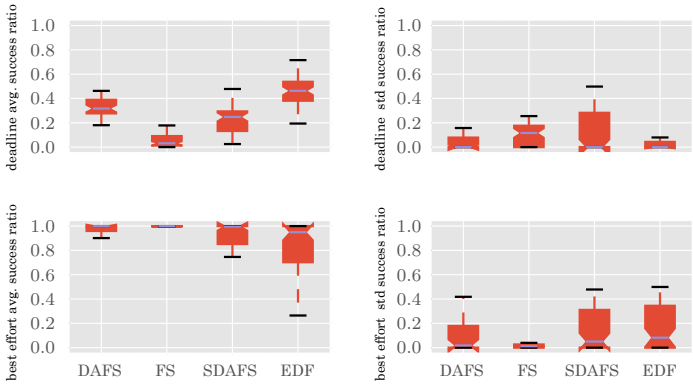


Fig. 14. Schedulers performance with $\alpha = 0.4$.

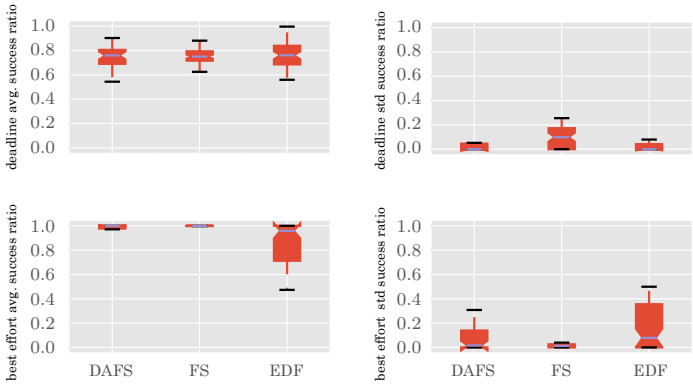
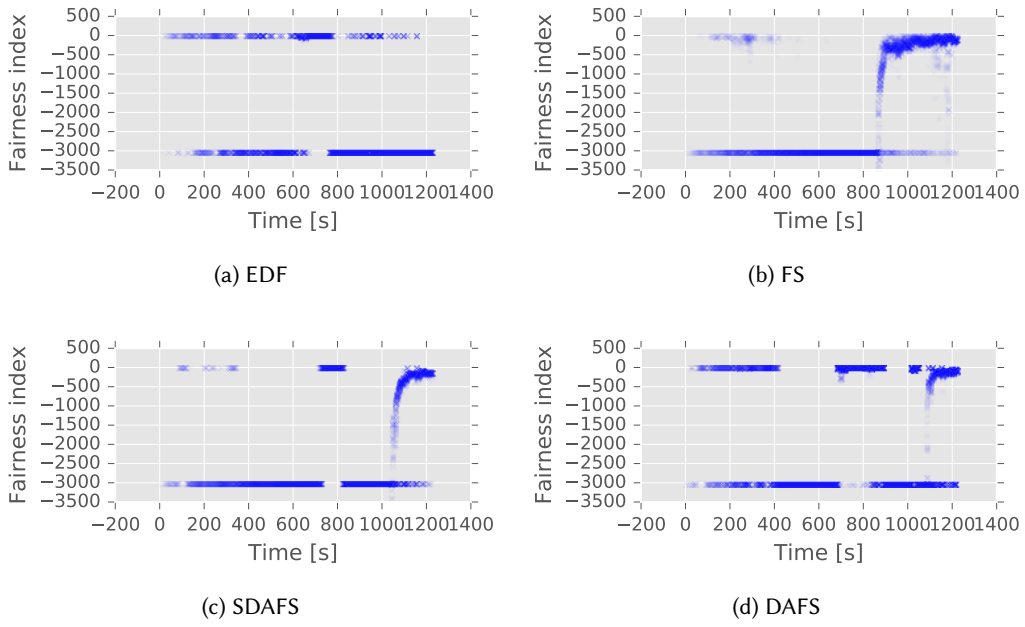


Fig. 15. Schedulers performance with $\alpha = 2$.

one of FS or DAFS, while FS performs poorly on the completion of batches with a deadline. This is even more apparent in Figure 15, for $\alpha = 2$, where the deadlines are easy to satisfy, but where the EDF scheduler wastes a lot of resources focusing also on late batches that are not satisfiable.



(a) EDF (b) FS

(c) SDAFS (d) DAFS

Fig. 16. Fairness index evolution for different schedulers over 1200 seconds.

The results are summarized in Figure 13 and Table 3. Regarding SDAFS, while the loss in performance compared to DAFS are significant (50% for deadline success ratio), it may still be preferred in situations where the additional computational power required cannot be met. Table 3 also shows the computation cost (in seconds): DAFS needs at least one order of magnitude more time than the other schedulers to be executed.

4.6.2 Fairness Index Dynamics. We visualize how the different schedulers behave in term of the fairness index $J_\infty(x)$ in Figure 16. We note that DAFS is the best in keeping the index high for most of the time. Interestingly, all algorithms except EDF show a trend towards favoring fairness as time passes and the number of new arrivals decreases.

4.7 Discussion

We presented above the results of a series of empirical crowdsourcing experiments and simulations where we varied different parameters to have a realistic scale experiment of production and best effort batches.

We started by identifying two features that impact the crowd work in a task scheduling environment i) crowd workers prefer larger batches that guarantee a continuous stream of tasks, and ii) their productivity is sensitive to context switch. We also show that to obtain better execution times, it is possible to increase the monetary reward of a batch. However, this solution is not scalable and might have a negative impact on the quality of the results.

Based on these observations, we make a number of design choices that are aimed to reduce context switch while at the same time tackling batch starvation and deadline issues without incurring higher costs. From our results, we conclude that the proposed Deadline-Aware Fair Scheduler

allows batches to be fairly treated, boosts production batches, and also takes into account work continuity to minimize the context switch for a given worker.

To summarize, the main observations that we draw from our experiments are:

- Large batches attract a larger workforce which yields a higher throughput;
- Using a `HIT-BUNDLE` has a positive impact on task latency as it tends to attract larger workforces;
- Individual workers perform slightly better when working on homogeneous batches (compared to batches regrouping different types of HITs);
- Fair Scheduling techniques make it possible to prioritize production batches while being fair with all running batches and all involved workers;
- Worker Conscious Fair Scheduling (WCFS) uses delayed scheduling to reduce the effects of context switch for the workers while maintaining the benefits of fair scheduling;
- Our Deadline-Aware Fair Scheduler prevents common pathological cases that are problematic with other deadline schedulers, e.g., the disruption of existing batches in the queue when a new batch is introduced.
- The techniques we evaluated are beneficial even on top of existing micro-task crowdsourcing platforms.

5 RELATED WORK

5.1 Micro-task Crowdsourcing

Paid micro-task crowdsourcing has been used for a wide range of applications including entity resolution [50], schema matching [53], entity linking and instance matching [13, 15], word sense disambiguation [45], relevance judgements [1] etc.

We can distinguish two types of crowdsourcing paradigms: pull-crowdsourcing and push-crowdsourcing [35]. The key difference is that pull-crowdsourcing platforms allow the workers to browse and choose among available tasks posted by the requesters, while push-crowdsourcing assigns tasks to workers by considering selection criteria such as skills, location or interests in order to assign tasks to the best available workers. In [8, 22], for example, authors leverage online social network profiles and activities to find better suited candidates and push tasks to them.

In this article, we focus on push-crowdsourcing which in order to apply task scheduling techniques and to decide which task should be served to the next available worker. Hence, our focus is on improving the crowd *efficiency* without the need of deploying a dedicated crowdsourcing platform but rather allowing us to reuse an existing crowdsourcing platforms such as AMT.

In our work, we have observed that latency and throughput can be controlled with the crowd size and pricing dimensions. The workforce size is generally a dynamic factor that influences the performance of a scheduling system, especially when we have to prioritize production batches. Methods for estimating the size of a dynamic crowd population have been studied in [21]. Optimal payment strategies, reward schemes, and incentive mechanisms for crowdsourcing have been studied [19, 31, 46] and may also be applied in combination to crowdsourcing scheduling techniques in order to maximize throughput as well as the quality of the results.

5.2 Low-latency Crowdsourcing

Some methods aim at to reducing the latency of crowdsourced batches, which is an objective related to ours. In [5], authors propose methods to have workers always available to synchronously work on tasks and achieve faster task completion time. This is done by means of retainer pools where workers actively wait for tasks to be available for them [6]. In [28] authors show how combining workers retained to wait for tasks with on-demand workers on a crowdsourcing platform can

provide low-latency results in the long term. Also looking at retainer pools of workers, [27] tackles different causes of latency in crowdsourcing and propose active learning methods to speed up task execution. Similarly to our work, Cioppino [26] looks at considering human factors when load balancing a crowd workforce across tasks that need to be completed. Their experimental results show similar improvements in platform throughput and cost reductions as we do but do not consider the aspect of deadline-aware approaches. Compared to these approaches, in our work we build on traditional micro-task crowdsourcing platforms and build a task scheduling layer on top instead of forming a dedicated pool of workers that we can notify on demand.

5.3 Task Assignment and Scheduling

Scheduling tasks for the crowd has been recently discussed in the context of work *quality* mostly, while we focus on efficiency. In CrowdControl [41], authors propose a scheduling approach to assign tasks to workers based on their history and how they learn by completing tasks. Instead, we focus on the requester needs for scheduling and look at priorities of batches, while still taking into account the human dimension of crowdsourcing. Moreover, [41] evaluates the proposed approaches by means of simulation while in our work we assess the effectiveness of the proposed algorithms over a real deployment over the crowd.

Similarly, SmartCrowd [43] considers task assignment as an optimization problem based on worker skills and their reward requirements. As compared to this, we rather focus on the system-side requirements for scheduling, by making sure that all competing batches are completed appropriately by the crowd.

Further pieces of work recently studied scheduling approaches focused on work quality: [32] shows, by means of simulations, how approaches that take into account worker skills outperform standard scheduling approaches, while [40] suggests scheduling tasks according to the required skills and the previous feedback from the requesters.

A different type of scheduling has been addressed in [17], where authors look at crowdsourcing tasks that need to take place in a specific real-world geographical location. In this case, it is necessary to schedule tasks for workers in order to minimize spatial movements by taking into account their geographical location.

Task allocation in teams has been studied in [2], where authors defined the problem, studied its complexity, and proposed greedy methods to allocate tasks to teams and accordingly adjust their size. Team formation given a task has been studied in [3] looking at worker skills. In our work, we rather focus on assigning tasks to individual workers to balance the load on the crowdsourcing platform.

Allocation of crowdsourcing tasks to individual workers has been studied in [38] where authors looked at optimal task allocation solutions based on available and required skills. Contrary to them, our scheduling problem focuses on optimizing the execution time of tasks that may have an execution deadline.

5.4 The Effect of Switching Tasks

When scheduling tasks for the crowd, it is necessary to take the human dimension into account. Recent work [34] showed how disrupting tasks continuity degrades the efficiency of crowd workers. Taking this result into account, we designed worker-conscious scheduling approaches that aim at serving tasks of the same type in sequence to crowd workers in order to leverage training effects and to avoid the negative effects of context switching.

Studies in the psychology domain have shown that switching between different tasks types has a negative effect on worker reaction time and on the quality of the work done (see, for example, [12]). In addition to this, in our work we show how context switch leads to an overall larger latency in

work completion (Section 4.2) and propose scheduling techniques that take this human factor into account. The authors of [51] study the effect of monetary incentives on task switching concluding that providing such incentives can help in motivating quality work in a task switching situation. In our work, we rather aim at reducing task switching by consciously scheduling tasks to workers. Related to this is the study of multi-tasking in crowdsourcing [42] where studies show how different types of tasks can be effectively completed in parallel without extending their individual execution time.

5.5 Similarities with the Cloud

Our system draws a parallel with scheduling and sharing costly resources in the cloud. A cloud-based clusters manage tens of thousands of computers with varying storage capacities, CPUs and GPUs. A internal resource managers operate using scheduling algorithms to orchestrate the allocations of the different resources efficiently. Our system is designed in the same spirit, and thus we can apply similar techniques to be resistant to adversarial behavior. Instead of having multi-tenants running computing jobs in the cloud, we have multiple users (or systems) launching crowdsourcing microtasks on AMT. For instance, the concept of weighted priority is achieved through dynamic pricing or admission control. While this is beyond the present work, we refer the reader to recent literature in this area in the context of cloud computing [4].

Compared to our previous work [20], here we additionally consider the existence of *production batches*. The key difference of this paper is looking at a deadline-aware setting were production batches get additional priority boost toward the finish line. However, we do not support soft or hard deadlines mostly due to the unpredictability of the workforce availability. We propose novel methods and experimental evidence that show how such requests can be satisfied by also considering the presence of other batches running concurrently on the crowdsourcing platform. Previous research described in Section 5.2 looked at ways to speed-up task execution using techniques such as worker pools available in real time and active learning methods. Instead, we look at scheduling techniques that can deal with different batch priorities. Most of the task assignment methods proposed in the literature (Section 5.3) consider the problem of matching worker skills with task requirements. Compared to them, rather than skill/requirement matching, we focus on batch priorities to schedule tasks and balance the workload to optimize their execution. Work described in Section 5.4 looks at human factors that affect efficient task completion. We build on top of these findings to design worker-aware scheduling algorithms that can deal with such human factors while scheduling tasks and still optimize for task completion time.

6 REAL-WORLD DEPLOYMENTS CONSIDERATIONS

Task-scheduling can readily be utilized in three different types of crowdsourcing environments i) Existing platforms with a large crowd and multipurpose tasks, e.g., AMT, ii) Platforms with niche applications and self-recruited crowds, e.g., ScaleAPI, and iii) Enterprise crowdsourcing, where only the employees of a company can perform the published tasks. Each deployment would require adaptations, for instance in this paper we showcased the use of a `HIT-BUNDLE` to publish and schedule tasks on Mturk. However, smaller platforms can support scheduling natively, which might create new challenges as they grow to a broader user and crowd base. We summarize the key differences of using task scheduling across the three types of platforms types in Table 4. In particular, we note the possibility of adversarial requester behavior where the users can game the system by strategically regrouping and repricing their tasks. This is an aspect that we do not tackle directly in this work as we assume a stable enterprise-like crowdsourcing environment with an administrator. Nevertheless, as mentioned in the section 5.3, strategy-proof research for resource allocation can be easily extended to the case of the crowd, especially since the concept of priority is

constrained to the price a requester is willing to pay or to bid for. Moreover, since we centralize the task scheduling and distribution process, we allow the integration of future strategy proof methods.

Finally, to facilitate further developments in this line of research, we have set up a project for simulating crowdsourcing demand and supply along with the current source code of the simulations we have conducted in this paper at https://github.com/dedcode/CrowdScheduler.

Table 4. Limitations and challenges of task-scheduling when used in existing crowdsourcing environments.

	Task Scheduling Integration	Scalability	Strategy Proofness
Large Platforms	Add-on Scheduling (with a redirect to an external page)	Large pool of workers that contribute to the throughput of the system	Requester are not restricted to the scheduling algorithms and can game the system via regular tasks
Specialized Platforms	Built-In Scheduling	Crowd size depends on growth and efforts of the platform to offer new services	New pricing schemes and strategy-proof algorithms are needed to handle a growing user-base
Enterprise Platforms	Built-In Scheduling	Limited to the company employees	Usually supervised by an Admin

7 CONCLUSIONS

In this paper, we extend our previous work on scheduling crowdsourced tasks by supporting best-effort batches as well as production batches with a deadline. We explored solutions for scheduling crowdsourcing tasks that support both fairness and deadline-awareness. We derived variants of the Fair Scheduling algorithms, where the deadline and the number of incomplete tasks give priority to production batches.

We experimentally show that a crowdsourcing system can increase its overall efficiency by bundling requests into a single batch we name `HIT-BUNDLE`, and then taking control of the distribution process of the tasks. Our experiments show that this approach has two benefits i) it creates larger batches that have a higher throughput, and ii) it gives to the system control on what HIT to push next—a feature that we leverage to push high-priority requests for example. Moreover, controlling the task execution makes it possible to develop more sophisticated crowdsourcing operators e.g., workflow execution, collaborative tasks.

We also showed how workers could be sensitive to context switch that arbitrary task assignment causes. The adverse effects of context switching were visible in our experiments and are corroborated by related studies in psychology. In that context, we proposed a Worker Conscious Fair Scheduling (WCFS), a new scheduling variant that strikes a balance between minimizing the context switches and the fairness of the system.

We experimentally validated our algorithms over real crowds of workers on a popular paid micro-task crowdsourcing platform running both controlled and uncontrolled experiments. Our results show that it is possible to achieve i) a better system efficiency—as we reduce the overall latency of a set of batches—while ii) providing fair executions across batches, resulting in iii) non-starving small batches. In addition, we validated our deadline-aware algorithms through simulations, showing that we can obtain results comparable to earlier deadline schedulers in terms of number of batches completed by their deadline (more than 90% of the one achieved by EDF in all cases), with the advantage of not sacrificing fairness amongst the best effort batches and avoiding pathological instances in which EDF can fail.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation under grant number PP00P2 153023, the European Union's Horizon 2020 research and innovation programme under grant agreement No 732328, a Google Research Award, and by the European Science Foundation by sponsoring the Science Meeting SM 7091 under the ELIAS Research Networking Program.

REFERENCES

[1] Omar Alonso and Ricardo Baeza-Yates. 2011. Design and Implementation of Relevance Assessments Using Crowdsourcing. In *Advances in Information Retrieval*, Paul Clough, Colum Foley, Cathal Gurrin, Gareth J. F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 153–164.

[2] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. 2010. Power in unity: forming teams in large-scale community systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 599–608.

[3] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. 2012. Online Team Formation in Social Networks. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA, 839–848.

[4] Moshe Babaioff, Yishay Mansour, Noam Nisan, Gali Noti, Carlo Curino, Nar Ganapathy, Ishai Menache, Omer Reingold, Moshe Tennenholtz, and Erez Timnat. 2017. ERA: A Framework for Economic Resource Allocation for the Cloud. In *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017*. 635–642.

[5] Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. 2011. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *UIST '11*. ACM, 33–42.

[6] Michael S Bernstein, David R Karger, Robert C Miller, and Joel Brandt. 2012. Analytic methods for optimizing realtime crowdsourcing. *arXiv preprint arXiv:1204.2995* (2012).

[7] Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, et al. 2010. Vizwiz: nearly real-time answers to visual questions. In *UIST*. ACM, 333–342.

[8] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Matteo Silvestri, and Giuliano Vesci. 2013. Choosing the right crowd: expert finding in social networks. In *EDBT '13*. ACM, 637–648.

[9] Edwin Chen. 2013. Improving Twitter Search with Real-Time Human Computation. https://goo.gl/EWT7Kt. (2013). Last accessed: 2018-12-12.

[10] Houssine Chetto and Maryline Chetto. 1989. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on software engineering* 10 (1989), 1261–1269.

[11] Lydia B. Chilton, John J. Horton, Robert C. Miller, and Shiri Azenkot. 2010. Task Search in a Human Computation Market. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '10)*. ACM, New York, NY, USA, 1–9.

[12] Matthew J. C. Crump, John V. McDonnell, and Todd M. Gureckis. 2013. Evaluating Amazon's Mechanical Turk as a Tool for Experimental Behavioral Research. *PLOS ONE* 8, 3 (03 2013), 1–18. https://doi.org/10.1371/journal.pone.0057410

[13] Gianluca Demartini, Djellel Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*. 469–478.

[14] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: Leveraging Probabilistic Reasoning and Crowdsourcing Techniques for Large-scale Entity Linking. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA, 469–478. https://doi.org/10.1145/2187836.2187900

[15] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2013. Large-scale Linked Data Integration Using Probabilistic Reasoning and Crowdsourcing. *The VLDB Journal* 22, 5 (Oct. 2013), 665–687. https://doi.org/10.1007/s00778-013-0324-z

[16] Gianluca Demartini, Beth Trushkowsky, Tim Kraska, Michael J Franklin, and UC Berkeley. 2013. CrowdQ: Crowdsourced Query Understanding.. In *CIDR*. www.cidrdb.org, Paper 137.

[17] Dingxiong Deng, Cyrus Shahabi, and Ugur Demiryurek. 2013. Maximizing the Number of Worker's Self-selected Tasks in Spatial Crowdsourcing. In *Proc. SIGSPATIAL/GIS*. ACM, 324–333.

[18] Ernesto Diaz-Aviles and Ricardo Kawase. 2012. Exploiting Twitter as a Social Channel for Human Computation. In *CrowdSearch*. 15–19.

[19] Djellel Difallah, Michele Catasta, Gianluca Demartini, and Philippe Cudré-Mauroux. 2014. Scaling-up the Crowd: Micro-Task Pricing Schemes for Worker Retention and Latency Improvement. In *Second AAAI Conference on Human*

Computation and Crowdsourcing. 50–58.

[20] Djellel Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. 2016. Scheduling Human Intelligence Tasks in Multi-Tenant Crowd-Powered Systems. *Proceedings of the 25th International Conference on World Wide Web, WWW 2016* (2016), 855–865.

[21] Djellel Difallah, Elena Filatova, and Panos Ipeirotis. 2018. Demographics and Dynamics of Mechanical Turk Workers. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, New York, NY, USA, 135–143.

[22] Djellel Eddine Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. 2013. Pick-a-crowd: Tell Me What You Like, and I'll Tell You What to Do. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. ACM, New York, NY, USA, 367–374.

[23] Siamak Faridani, Björn Hartmann, and Panagiotis G. Ipeirotis. 2011. What's the Right Price? Pricing Tasks for Finishing on Time. In *Proceedings of the 11th AAAI Conference on Human Computation (AAAIWS'11-11)*. AAAI Press, 26–31. http://dl.acm.org/citation.cfm?id=2908698.2908703

[24] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: answering queries with crowdsourcing. In *SIGMOD '11*. ACM, 61–72.

[25] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: fair allocation of multiple resource types. In *NSDI'11*. USENIX Association, 24–24.

[26] Daniel Haas and Michael J Franklin. 2017. Cioppino: Multi-Tenant Crowd Management. In *Proceedings of the fifth AAAI Conference on Human Computation and Crowdsourcing (HCOMP)*. AAAI Press, 41–50.

[27] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J Franklin. 2015. Clamshell: Speeding up crowds for low-latency data labeling. *Proceedings of the VLDB Endowment* 9, 4 (2015), 372–383.

[28] Ting-Hao K Huang and Jeffrey P Bigham. 2017. A 10-Month-Long Deployment Study of On-Demand Recruiting for Low-Latency Crowdsourcing. In *In Proceedings of The fifth AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2017). AAAI, AAAI*. AAAI Press, 61–70.

[29] Panagiotis G. Ipeirotis. 2010. Analyzing the Amazon Mechanical Turk Marketplace. *XRDS* 17, 2 (Dec. 2010), 16–21. https://doi.org/10.1145/1869086.1869094

[30] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. *Throughput fairness index: An explanation.* Technical Report. Tech. rep., Department of CIS, The Ohio State University.

[31] Radu Jurca and Boi Faltings. 2009. Mechanisms for making crowds truthful. *Journal of Artificial Intelligence Research* 34 (2009), 209–253.

[32] Roman Khazankin, Harald Psaier, Daniel Schall, and Schahram Dustdar. 2011. QoS-Based Task Scheduling in Crowdsourcing Environments. In *Service-Oriented Computing*, Gerti Kappel, Zakaria Maamar, and Hamid R. Motahari-Nezhad (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 297–311.

[33] T. Lan, D. Kao, M. Chiang, and A. Sabharwal. 2010. An Axiomatic Theory of Fairness in Network Resource Allocation. In *2010 Proceedings IEEE INFOCOM*. 1–9. https://doi.org/10.1109/INFCOM.2010.5461911

[34] Walter S Lasecki, Adam Marcus, Jeffrey M Rzeszotarski, and Jeffrey P Bigham. 2014. Using Microtask Continuity to Improve Crowdsourcing. *Technical Report* (2014).

[35] Edith Law and Luis von Ahn. 2011. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 5, 3 (2011), 1–121.

[36] Svetlana Lazebnik, Cordelia Schmid, Jean Ponce, et al. 2004. Semi-local affine parts for object recognition. In *British Machine Vision Conference (BMVC'04)*. 779–788.

[37] Adam Marcus and Aditya Parameswaran. 2015. Crowdsourced Data Management: Industry and Academic Perspectives. *Foundations and TrendsÂ® in Databases* 6, 1-2 (2015), 1–161. https://doi.org/10.1561/1900000044

[38] Panagiotis Mavridis, David Gross-Amblard, and Zoltán Miklós. 2016. Using Hierarchical Skills for Optimized Task Assignment in Knowledge-Intensive Crowdsourcing. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 843–853. https://doi.org/10.1145/2872427.2883070

[39] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. 2014. Scaling Up Crowdsourcing to Very Large Datasets: A Case for Active Learning. *Proc. VLDB Endow.* 8, 2 (Oct. 2014), 125–136. https://doi.org/10.14778/2735471.2735474

[40] Vikram Nunia, Bhavesh Kakadiya, Chittaranjan Hota, and Muttukrishnan Rajarajan. 2013. Adaptive Task Scheduling in Service Oriented Crowd Using SLURM.. In *ICDCIT*. 373–385.

[41] Vaibhav Rajan, Sakyajit Bhattacharya, L Elisa Celis, Deepthi Chander, Koustuv Dasgupta, and Saraschandra Karanam. 2013. CrowdControl: An online learning approach for optimal task scheduling in a dynamic crowd platform. In *ICML Workshop on 'Machine Learning meets Crowdsourcing'*. PMLR, 1–9.

[42] Akshay Rao, Harmanpreet Kaur, and Walter S. Lasecki. 2018. Plexiglass: Multiplexing Passive and Active Tasks for More Efficient Crowdsourcing. In *Proceedings of the Sixth AAAI Conference on Human Computation and Crowdsourcing,*

HCOMP 2018, Zürich, Switzerland, July 5-8, 2018. AAAI Press, 145–153.
[43] Senjuti Basu Roy, Ioanna Lykourentzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. 2014. Optimization in Knowledge-Intensive Crowdsourcing. *CoRR* abs/1401.1302 (2014).
[44] Jeffrey M. Rzeszotarski, Ed Chi, Praveen Paritosh, and Peng Dai. 2013. Inserting Micro-Breaks into Crowdsourcing Workflows.. In *HCOMP (Works in Progress / Demos) (AAAI Workshops)*, Vol. WS-13-18. AAAI Press.
[45] Nitin Seemakurty, Jonathan Chu, Luis von Ahn, and Anthony Tomasic. 2010. Word sense disambiguation via human computation. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '10)*. ACM, 60–63.
[46] Adish Singla and Andreas Krause. 2013. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *WWW '13*. ACM, 1167–1178.
[47] Keith Vertanen and Per Ola Kristensson. 2011. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*. ACM, 295–298.
[48] Luis von Ahn and Laura Dabbish. 2004. Labeling Images with a Computer Game. In *CHI '04*. ACM, 319–326.
[49] Jing Wang, Siamak Faridani, and P Ipeirotis. 2011. Estimating the completion time of crowdsourced tasks using survival analysis models. *Crowdsourcing for search and data mining (CSDM 2011)* 31 (2011), 31–38.
[50] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB Endow.* 5, 11 (July 2012), 1483–1494. https://doi.org/10.14778/2350229.2350263
[51] Ming Yin, Yiling Chen, and Yu-An Sun. 2014. Monetary Interventions in Crowdsourcing Task Switching. In *Proceedings of the 2nd AAAI Conference on Human Computation (HCOMP)*. AAAI Press, 234–241.
[52] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys '10*. ACM, 265–278.
[53] Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Chen Caleb Cao. 2013. Reducing Uncertainty of Schema Matching via Crowdsourcing. *Proc. VLDB Endow.* 6, 9 (July 2013), 757–768. https://doi.org/10.14778/2536360.2536374

Received Sept 2017; revised December 2018; accepted December 2018