Scheduling Human Intelligence Tasks in Multi-Tenant Crowd-Powered Systems

Djellel Eddine Difallah†, Gianluca Demartini† and Philippe Cudré-Mauroux†
† eXascale Infolab, University of Fribourg—Switzerland
⇤ Information School, University of Sheffield—United Kingdom

ABSTRACT

Micro-task crowdsourcing has become a popular approach to effectively tackle complex data management problems such as data lineage, missing values, or schema matching. However, the backend crowdsourced operators of crowd-powered systems typically yield higher latencies than the machine-processable operators, this is mainly due to inherent efficiency differences between humans and machines. This problem can be further exacerbated by the lack of workers on the target crowdsourcing platform, or when the workers are shared unequally among a number of competing requesters; including the concurrent users from the same organization who execute crowdsourced queries with different types, priorities and prices. Under such conditions, a crowd-powered system acts mostly as a proxy to the crowdsourcing platform, and hence it is very difficult to provide efficiency guarantees to its end-users.

Scheduling is the traditional way of tackling such problems in computer science, by prioritizing access to shared resources. In this paper, we propose a new crowdsourcing system architecture that leverages scheduling algorithms to optimize task execution in a shared resources environment, in this case a crowdsourcing platform. Our study aims at assessing the efficiency of the crowd in settings where multiple types of tasks are run concurrently. We present extensive experimental results comparing i) different multi-tenant crowdsourcing jobs, including a workload derived from real traces, and ii) different scheduling techniques tested with real crowd workers. Our experimental results show that task scheduling can be leveraged to achieve fairness and reduce query latency in multi-tenant crowd-powered systems, although with very different tradeoffs compared to traditional settings not including human factors.

General Terms

Design; Experimentation; Human Factors.

Keywords

Crowdsourcing; Scheduling; Crowd-Powered System.

1. INTRODUCTION

Thanks to micro-task crowdsourcing platforms such as Amazon Mechanical Turk† (AMT) and Crowdflower‡, it is today possible to build hybrid human-machine systems combining both the scalability of computers with the yet unmatched cognitive abilities of the human brain.

Micro-task crowdsourcing has already been used to power, among others, database systems [15], image search engines [10], or machine-learning algorithms [36]. In these systems, human and machines behave fundamentally differently: While machines can deal with large volumes of data, with real-time streams, and with flocks of concurrent users interacting with the system, crowdsourcing is mostly used as a batch-oriented, offline data processing paradigm, as opposed to cases where achieving low latency is key. The main reason behind this gap lies in the fact that 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 only.

With increased momentum around crowdsourcing for both academic and commercial purposes [12], managing the efficiency of a crowdsourcing platform in delivering results and completing tasks becomes a challenge. Efficiency concerns have so far mostly been tackled by adjusting the price of the Human Intelligence Tasks (HITs) or by repeatedly re-posting them on the crowdsourcing platform [14, 4, 11]. In this work, we propose to take control of the distribution process of tasks originating from multi-tenant crowd-powered systems where multiple, and potentially heterogeneous, HITs are executed. This allows us to apply scheduling techniques to decide which task gets to be sent to the next available worker.

We apply, adapt, and empirically evaluate a series of scheduling techniques that can be used by multi-tenant crowd-powered systems to launch HITs onto the crowdsourcing platform in order to improve their overall efficiency. For that purpose, we propose a system architecture and its AMT-tailored implementation, which have the following system-oriented objectives:

- improve the overall execution time of the generated workload, while
- ensure fairness among the different users of the system by equitably balancing the available workforce, and
- avoid starvation of smaller requests.

†http://mturk.com
‡http://crowdflower.com
From a worker perspective, task scheduling presents new challenges such as context switching and user priming. Hence, we try to answer the following questions: “Does known scheduling algorithms exhibit their usual properties when applied to the crowd?” and “What are the adaptations needed to accommodate the crowd work routine?”

To the best of our knowledge, this paper is the first piece of work focusing on applying scheduling techniques in order to improve the efficiency of crowd-powered systems. While our focus is on efficiency, considering quality constrains to take scheduling decisions is left outside of the scope of this paper (see Section 7 for related work on this topic).

In the following, we experimentally compare the efficiency of various crowd scheduling approaches on real crowds of workers working in a micro-task crowdsourcing platform: We vary the size of the crowd, the ordering and priority of the tasks, as well as the size of the task batches. In addition, we take into account the unique characteristics of the crowd workers such as the effect of context switching and work continuity to design crowd-aware scheduling algorithms. Our experimental settings include i) a controlled setup with a fixed number of workers involved in the experiments, and ii) a real-world setup with varying number of workers, and HIT workloads taken from a commercial crowdsourcing platform log. The results of our experimental evaluation indicate that using scheduling approaches for micro-task crowdsourcing maximize the overall latency of batches of tasks irrespective of their size, while significantly improving the productivity of workers measured as their average execution time.

In summary, the main contributions of this paper are:

- A crowdsourcing system architecture that implements the newly proposed multi-tenant crowd-powered system focusing on HIT scheduling to improve system efficiency;
- A HIT scheduling layer for crowd-powered systems serving multiple users;
- A series of scheduling algorithms customized for the crowd;
- An extensive empirical evaluation comparing the scheduling algorithms over the crowd conducted both in a controlled setting as well as in a real deployment.

The rest of the paper is structured as follows. Section 2 describes the HIT scheduling problem for multi-tenant crowd-powered systems. We introduce our new architecture in Section 3. In Section 4, we present the different scheduling algorithms we implemented for assigning HITs to crowd workers, while Section 5 presents an extensive experimental evaluation of the proposed techniques. We discuss the results and summarize our main findings in Section 6. Section 7 gives an overview of current approaches in crowd-powered systems and micro-task crowdsourcing and how they motivated our investigation. Finally, we conclude the paper in Section 8.

2. MOTIVATION

In this section, we give an overview of crowdsourcing platforms in order to highlight some of their characteristics that motivated our approach. We focus on Amazon Mechanical Turk as i) it is currently the most popular micro-task crowdsourcing platform, ii) there is a continuous flow of workers and requesters completing and publishing HITs on the platform, and iii) its activity logs are available to the public [17].

2.1 The AMT Platform

Amazon Mechanical Turk is an open crowdsourcing marketplace where the crowd is free to choose what to work on. This is desirable as it imposes high HIT standards; for instance, requesters need to pay close attention to their HIT design, documentation and pricing in order to attract and retain workers. On the other hand, this freedom limits the possibilities of the platform to provide any form of service guarantees to the requesters.

2.2 Requesters

In crowdsourcing platforms, businesses that heavily rely on micro-task crowdsourcing for their daily operations end up competing with themselves: If a requester runs concurrent campaigns on a crowdsourcing platform, these will end up affecting each other. For example, a newly posted large batch of HITs is likely to get more attention than a two days old batch waiting to be finished with few HITs remaining (see below for an explanation on that point).

2.3 HITs Execution Patterns in AMT

One of the common phenomena in micro-task crowdsourcing is the presence of long-tail work distributions: In a batch of HITs, the bulk of the work is completed by a few workers who perform most of the tasks while the rest is performed by many different workers who perform just a few HITs each.

We run an initial crowdsourcing experiment to observe the effect that the number of simultaneous workers has on the throughput of our batch (HITs/Minute), and the amount of work done by each worker. We can see in Figure 2a that the overall throughput of the system increases linearly with the number of workers. Figure 2b shows the amount of work (number of HITs submitted during the experiment) performed by each worker. Here, we can observe a long-tailed distribution where few workers perform most of the tasks while many workers perform just a few tasks.

We extend our analysis to the throughput of batches running concurrently on AMT. To that end, we computed the throughput of every batch publicly visible during a three months period, and grouped the results into three categories of batches (tiny, small, medium and large). The results are depicted in Figure 1. We observe that large batches dominate the throughput of a crowdsourcing platform even if the vast majority of the running batches are very small (less than 10 HITs). Large batches are completed at a certain speed by the crowd, up to a certain point when few HITs are left in the batch. Those final few HITs take a much longer time to be completed as compared to the majority of HITs in the batch. Such a batch starvation phenomenon has been observed in a number of recent reports, e.g., in [14, 34] where authors observe that the batch completion time depends on its size and on HIT pricing. HIT completion starts off quickly but then loses momentum. In that sense, large batches of tasks are able to systematically yield higher throughputs as more crowd workers can work on them in parallel.

A similar observation was made in [15], where the authors compared the throughput of different batch sizes and concluded that large batches have the highest throughput, while medium sized batches (50-100 tasks) completed faster. We can conjecture that these phenomena are partially due to the preference of the crowd towards large batches. Indeed, the workers tend to explore new batches with many HITs,
First, we formally define the problem of scheduling HITs. When scheduling a HIT on AMT, 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 forwarding the final results to the end-users.

Workers are allowed to not accept (or return) HITs they prefer not to complete. The workers can also leave the system at any point in time. In these cases, the Scheduler takes responsibility of updating the queue and reschedules uncompleted HITs.

In the following section, we introduce a number of scheduling algorithms that can be used to manage HIT queues.

4. HIT SCHEDULING MODELS

In this section we give a formal problem definition of scheduling with the crowd, and introduce the set of design requirements we accounted for when creating and choosing the scheduling strategies. We will also briefly revisit common scheduling approaches used by existing resource managers in shared environments, and discuss their advantages and drawbacks when applied to the architecture presented in Section 3. As we show in Section 3, using such algorithms presents several new dimensions to be taken into account compared to traditional CPU scheduling, thus, we also propose new scheduling algorithms adapted to the crowd.

4.1 HIT Scheduling: Problem Definition

First, we formally define the problem of scheduling HITs generated by a multi-tenant crowd-based system on top of a crowdsourcing platform.

A query \( r \) submitted to the system and including crowd-powered operators generates a batch \( B_r \) of HITs. We define a batch \( B_j = \{h_1, \ldots, h_n\} \) as a set of HITs \( h_i \). Each batch
has additional metadata attached to it: A monetary budget $M_i$ to be spent for its execution and a priority score $p_i$ with which it should be completed: Batches with higher priority should be executed before batches with lower priority. Thus, 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.

The problem of scheduling HITs takes as input a set of available batches $\{B_1, ..., B_n\}$ and a crowd of workers $\{w_1, ..., w_m\}$ currently active on the platform, and produces as output an ordered list of HITs from $\{B_1, ..., B_n\}$ to be assigned to workers in the crowd by publishing them as a single HIT-BUNDLE. Once a worker $w_i$ is available, the system assigns him/her the first task in the list as decided by the scheduling algorithm.

Scheduling may need to be repeated over time to update the HIT execution queue. Such re-scheduling operations are necessary, for example when a worker fails to complete his/her assigned HIT, or when a new batch of HITs is submitted by one of the clients.

In this way, we obtain some hybrid pull-push behavior on top of AMT as the workers participating in the crowdsourcing campaign are shown HITs computed by the scheduler. Workers are still free to decline the HIT, ask for another one, or simply seek for another requester on AMT.

**Worker Context Switch.** From the 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 etc, and this could be penalizing (see our related work section 7). This overhead is called context switch. One of the goals of this paper is to improve the efficiency of each worker by mitigating her context switches.

### 4.2 HIT Scheduling Requirement Analysis

Next, we describe which requirements should be taken into account when applying scheduling in a crowdsourcing setting. We then use some of these requirement to customize known scheduling techniques for the crowd.

(R1) **Runtime Scalability:** unlike parallel schedulers, where the compiled query plan dictates where and when the operators should be executed [30], crowd-powered systems are bound to adopt a runtime scheduler that a) dynamically adapts to the current availability of the crowd, and b) scales to make realtime scheduling decisions as the work demand grows higher. A similar design consideration is adopted by YARN[31], the new Hadoop resource manager.

(R2) **Fairness:** An important feature that any shared system should provide is fairness across the users of the system. By taking control of the HIT-BUNDLE scheduling, the crowd-powered system acts as the load balancer of the currently available crowd and the remaining HITs in the HIT-BUNDLE. For example, the scheduler should provide a steady progress to large requests without blocking – or starving, the smaller requests.

(R3) **Priority:** in a multi-tenant System, some queries have a higher priority than others. For this reason, HITs generated from the queries should be scheduled accordingly. In a crowdsourcing scheduling setting, as workers are not committed to the platform and can leave at any point in time, a crowd-powered system scheduler should be best-effort, that is, the system should do its best to meet the requester priority requirements without any hard guarantee.

(R4) **Worker Friendly:** Differently from CPUs, people performances are impacted by many factors including training effects, boredom, task difficulty and interestingness. Scheduling approaches over the crowd should whenever possible 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 HITs to workers and providing fair execution to different HIT batches.

### 4.3 Basic Space-Sharing Schedulers

Crowdsourcing platforms usually operate in a non-preemptive mode, that is, they do not allow to interrupt a worker performing a task of low priority to have him perform a task of higher priority with the risk of reneging. In our evaluation we consider common space-sharing algorithms where a resource (a crowd worker in this case) is assigned a HIT until he/she finishes it, or returns it uncompleted to the platform.

**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 job until it is done, FIFO provides the best
throughput per batch one can expect from the platform at a
given moment in time.

The potential shortcomings of this scheme are as follows:
i) short jobs and high priority jobs can get stuck behind long
running tasks, minimizing the overall efficiency of the crowd-
sourcing system, and ii) when a batch has a large number
of tasks, assigned workers can potentially get bored [27].

Shortest Job First (SJF). Other simple scheduling
schemes offer different tradeoffs depending on the
requirements of the multi-tenant system. Shortest Job
First (SJF) offers fast turn-around for short HITs, and
can lead to a minimum of a context switch for part of the
crowd, since the shortest jobs are either quickly finished or
scheduled to the first available workers.

However, SJF is not strategy-proof on current crowd-
sourcing 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-DBMSs). Moreover, these schemes do not
systematically interweave tasks from different batches, and
thus present also the same shortcomings as FIFO.

Round Robin (RR). The previous schemes introduces bi-
ases, in the sense that they give an advantage to one batch
over the others. Round Robin removes such biases by assign-
ing 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 require-
ment (R2) 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 her-
self bouncing across tasks and being forced to continuously
switch context, hence losing time to understand the spe-
cific instructions of the tasks. The negative effect of context
switch is evident from our experimental results (see Section
5) and should be avoided.

4.4 Fair Schedulers
In order to deal with batches of HITs having different pri-
orities while avoiding starvation, we also consider scheduling
techniques frequently used in cluster computing.

Fair Sharing (FS). Sharing heterogeneous resources across
jobs having different demands is a well-known and com-
plex problem that has been tackled by the cluster com-
puting community. One popular approach currently used in
Hadoop/Yarn is Fair Scheduling (FS) [16]. In the con-
text of scheduling HITs on a crowdsourcing platform, we
borrow this approach in order to achieve fair scheduling of
micro-tasks: Whenever a worker is available, he/she gets a
HIT from the batch with the lowest number of currently as-
signed HITs which we call running_tasks. Unlike Round
Robin, this ensures that all the jobs get the same amount
of resources (thus being fair). Algorithm 1 gives the exact
way we considered FS in our context.

Weighted Fair Sharing (WFS). In order to schedule
batches with higher priority first (see R2 in Section 4.2),
weighted fair scheduling can be used, in order assign a task
from the jobs with the least running_tasks/task_priority
value. Algorithm 1, line 2, gets in that case updated: Sort
B by increasing r_i

\[ w_j = \frac{p_j}{\sum_i p_i} \]

4.5 Crowd-aware Scheduling
In addition to the standard scheduling techniques
described above, we also evaluate a couple of approach
aiming at scheduling tasks taking into account the crowd
workers need (see R4 in Section 2). In that sense, we
propose scheduling approaches that offer a tradeoff between
being fair to the batches (by load-balancing the workers)
while also being fair to the workers (by serving HITs with
some continuity, if possible, and with minimal wait time).

Worker Conscious Fair Sharing (WCFS). Worker Con-
scious Fair Sharing (WCFS) maximizes the likelihood of
a worker receiving a task from a batch he worked on re-
cently, thus avoiding that a worker jumps back and forth
between different tasks (i.e., minimizing context switching).
We suggest to achieve this by having top priority batches
concede their positions in favor of one of the next batches in
the queue. Each batch can concede his turn up to K
times, a predefined concession threshold, which is reset af-
after a scheduling. This approach is the crowd-equivalent of
Delay Scheduling [39].

5. EXPERIMENTAL EVALUATION
We describe in the following our experimental results ob-
tained by scheduling HITs on the Amazon MTurk (AMT)
crowdsourcing platform.

As a general experimental setup, we implemented the ar-
chitecture proposed in Section 3 on top of AMT’s API. Our
implementation and datasets are available as an open-source
project for reproducibility purposes and as a basis for po-
tential extensions.3

5.1 Datasets
For our experiments, we used a dataset composed of 7
batches of varying complexity, sizes, and reference prices.
The data was partly created by us and partly collected from
related works; it includes typical tasks that could have been
generated by a crowd-powered system. Table 1 gives a sum-
mary of our dataset and provides a short description and

3https://github.com/XI-lab/HIT-Scheduler
Algorithm 2 Worker Conscious Fair Share

Input: $B = \{b_i \prec p_i, r_i, s_i, \ldots, b_n \prec p_n, r_n, s_n\}$ set of batches currently queued with priority $p_i, r_i$ number of running HITs, and $s_i$ concessions initialized to 0.

Output: $K =$ maximum concession threshold

Output: HIT $b_i$,

1: When a worker $w_j$ is available for a HIT
2: $b_{last} =$ Last batch that $w_j$ did / null if it’s a new worker
3: $B_{Sorted} =$ Sort B by increasing $r_i/p_i$
4: if $b_{last} == null$ then
5: $B_{Sorted}[0].s = 0$
6: return $B_{Sorted}[0].getNextHit()$
7: end if
8: for $b$ in $B_{Sorted}$ do
9: if $b == b_{last}$ then
10: $b.s = 0$
11: return $b.getNextHit()$
12: else if $b.s < K$ then
13: $b.s ++$
14: continue
15: else
16: $b.s = 0$
17: $b.getNextHit()$
18: end if
19: end for

references when applicable. We note that for the purpose of our experiments, we vary the batch sizes and prices according to the setup.

5.2 Micro Benchmarking

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

5.2.1 Batch Split-up

The first question we address is whether smaller or larger batches of homogeneous HITs are more attractive to the workers on AMT. We experimentally check if a single 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 so all the variables like crowd population and size, concurrent requesters, and rewards are the same across the different settings.

Figure 4 shows how the three different batch splitting strategies executed overtime 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 a 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 $\textit{HIT-BUNDLE}$.

<table>
<thead>
<tr>
<th>ID</th>
<th>Dataset</th>
<th>Description</th>
<th>Price per HIT</th>
<th>#HITS</th>
<th>Avg. Time per HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Customer Care</td>
<td>Find the customer-care phone number of a given US-based company</td>
<td>$0.07</td>
<td>50</td>
<td>77sec</td>
</tr>
<tr>
<td>B2</td>
<td>Image Tagging</td>
<td>Type all the relevant keywords related to a picture from the ESP game dataset. [13]</td>
<td>$0.02</td>
<td>50</td>
<td>40sec</td>
</tr>
<tr>
<td>B3</td>
<td>Sentiment Analysis</td>
<td>Classify the expressed sentiment of a product review (positive, negative, neutral).</td>
<td>$0.05</td>
<td>200</td>
<td>22sec</td>
</tr>
<tr>
<td>B4</td>
<td>Type a Short Text</td>
<td>This is a study on short memory, where a worker is presented with text for a few seconds, then he is asked to type it from memory. [32]</td>
<td>$0.03</td>
<td>100</td>
<td>11sec</td>
</tr>
<tr>
<td>B5</td>
<td>Spelling Correction</td>
<td>A collection of short paragraphs to spell check from StackExchange.</td>
<td>$0.03</td>
<td>100</td>
<td>36sec</td>
</tr>
<tr>
<td>B6</td>
<td>Butterfly Classification</td>
<td>Classify a butterfly image to one of 6 species (Admiral, Black Swallowtail, Mormon, Monarch, Peacock, and Zebra). [23]</td>
<td>$0.01</td>
<td>600</td>
<td>15sec</td>
</tr>
<tr>
<td>B7</td>
<td>Item Matching</td>
<td>Unaply identify products that can be referred to by different names (e.g., iPad Two and Padified Generation). [35]</td>
<td>$0.01</td>
<td>96</td>
<td>22sec</td>
</tr>
</tbody>
</table>

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

Figure 4: A performance comparison of batch execution time using different grouping strategies publishing a large batch of 600 HITs vs smaller batches (From B6).

Figure 5: A performance comparison of batch execution time using different grouping strategies publishing two distinct batches of 192 HITs separately vs combined inside an $\textit{HIT-BUNDLE}$.

5.2.2 Merging Heterogenous Batches

We extend the above experiment to compare the execution of two heterogenous batches run separately or within a single $\textit{HIT-BUNDLE}$. 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 $\textit{HIT-BUNDLE}$ 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 within the $\textit{HIT-BUNDLE}$. The results are depicted in Figure 5.

\[\text{We observe that AMT randomly selects the input to serve.}\]
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 involving 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.

### 5.2.3 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 an 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. In order to trigger the context switch, each participant was asked to do at least 10, and up to 100, tasks.

Figure 6 shows the average execution time of all the 100 HITs under each execution group. We observe that the average of execution time of HITs 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 Wilcoxon signed-rank test. SEQ10 has a p-value of 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 longer sequences of HITs of the same type.

### 5.3 Scheduling HITs for the Crowd

Now we move our attention to experimentally comparing the scheduling algorithm that are used to manage the distribution of HITs within a HIT-BUNDLE.
to the input of the batches, we randomly select batches from batches running concurrently. Since we do not have access hour activity on AMT from a uncontrolled crowdsourcing setting using our proposed fair scheduling techniques FS and WCFS in an involved in scheduling HITs over the crowd, we now evaluate 5.4 Live Deployment Evaluation
different batches obtain different levels of improvement. completed faster when more workers are involved. However, in all other settings constant). We can see batches being completed later. This comes at the expense of other batches being completed faster using the FS scheduling approach (gray bar of batch 2 lower than the black one). This comes at the expense of other batches being completed later. Another dimension that we vary is the crowd size. Figure 8b shows the batch completion time of two different crowd- workers varied wildly overtime in each setup. Corroborat-
ing 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 5.3.2, but varying the price attached to B2 and using the FS algorithm only. Figure 8 shows that batches with a higher priority (reward) lead to faster completion times using the FS scheduling approach (gray bar of batch 2 lower than the black one). This comes at the expense of other batches being completed later.

5.3.3 Varying the Control Factors

In order to test our priority control mechanism across different batches, we create a workload that mimics a 1- workforce with respect to the different batches, i.e., there was no preferential treatment.

Other parameters like requester name and reward are similar.

5.4 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 real 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.

5.4.1 Live Deployment Experimental Setup

We publish concurrently the 28 batches from the previ-
ously described trace as individual batches (standard ap-
proach) 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.

5.4.2 Average Execution Time

Figure 9 shows the average HIT execution time obtained by the different setups. Confirming the results from Section 5.2.3, we observe that workers perform better when working on individual batches because of the missing context switch effect (though the performance difference is minimal). 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.

5.4.3 Results of the Live Deployment Run

We plot the CDFs of HIT completion per category in Figure 10. For example, 25% of small batches completed in 500 seconds when run 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., long period without progress); here, we clearly see the benefits of both FS and WCFS at load balancing.

The final plot (Figure 11) 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 overtime in each setup. Corroborat-

Figure 8: (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.

Figure 9: Average execution time per HIT under different scheduling schemes.

Figure 10: CDF of different batch sizes and scheduling schemes.
main observation is that FS and WCFS i) achieve their desired property of load balancing the batches when there are sufficient number of workers, ii) they finish all the jobs well before the individual execution (10-15 minutes considering the 95th percentile).

6. DISCUSSION

In Section 4.2, we introduced a set of requirements for scheduling HITs. The different scheduling techniques that we propose meet these requirements as follows:

(R1) Scalability: we choose not to tackle scheduling as a high-complexity multivariate optimization problem but rather as a more scalable, HIT sorting task; this property is desirable since our scheduling algorithms are ought to serve large numbers of crowd workers—"potentially billions of users" [18];

(R2/R3) Fairness and Priority: thanks to FairSharing and weighted FS, we equitably load balance HITs and express priority of HITs as a function of the price and schedule them accordingly; (R4) Worker-Consciousness: thanks to WCFS and CGS, we are able to adapt scheduling to the crowd.

We presented above the results of a series of empirical crowdsourcing experiments where we varied different dimensions.

Starting with a set of micro benchmarks, the first observation we make is the attractiveness of larger batches to the crowd. A possible explanation of this observation is related to the overhead of searching for new batches to work on, thus the preference given to larger batches. Next, we optimized for reducing context switches, as it is a well-studied problem and has a direct impact on workers’ efficiency.

From our experimental results, we conclude that the most appropriate scheduling technique among the ones we considered is the WCFS variant, which allows HIT batches to be fairly treated on the crowdsourcing platform and also takes into account the needs of workers to have some continuity in the HIT they focus on rather than potentially causing them to constantly switch context.

On the worker side, we identify two key features that make crowd workers execute tasks very differently compared to machines: i) crowd workers suffer from context switch after changing the type of task they work on, and ii) they are attracted by large HIT batches that guarantee a continuous stream of HITs and, thus, of revenue.

We note that to obtain better execution times it is possible, for example, to increase the monetary reward attached to the HITs. However, such reward increases would make the crowdsourcing cost rise thus hindering the scalability of the approach. Moreover, increasing the monetary reward opens up the door to spammer workers who are exclusively interested in the monetary reward and not in honestly completing the HITs.

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

- Large HIT batches are preferred by crowd workers;
- Thus, large batches attract a larger workforce which implies a higher throughput;
- Individual workers perform slightly better when working on homogeneous batches (compared to batches regrouping different types of HITs);
- HIT-BUNDLE have overall a positive impact on task latency as they tend to attract bigger workforces;
- Scheduling techniques make it possible to prioritize HIT batches as needed while being fair with all running batches and all involved workers; In particular, FS and WCFS equally distribute the available workforce over the different batches;
- The techniques we evaluated can be applied on top of existing micro-task crowdsourcing platforms in a scalable fashion without the need of new push crowdsourcing mechanisms or systems, thus leveraging large crowds of people already engaged on existing platforms.

7. RELATED WORK

Micro-task Crowdsourcing. Paid micro-task crowdsourcing has been used for a wide range of applications including entity resolution [35, 37], schema matching [40], entity linking and instance matching [7, 8], word sense disambiguation [28], relevance judgements [1] etc.

We can distinguish two types of crowdsourcing paradigms: pull-crowdsourcing and push crowdsourcing [22]. 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 [13, 5], for example, authors leverage online social network profiles and activities to find better suited candidates and push tasks to them.

In this paper, we instead propose the use of a HIT-BUNDLE, that is, a unique batch of heterogeneous tasks generated by a multi-tenant system. This allows to apply task scheduling techniques within the HIT-BUNDLE and to decide which task should be served to the next available worker. In this way, we rather focus on improving the crowd efficiency without the need of deploying a dedicated crowdsourcing platform but rather allowing us to reuse popular crowdsourcing platforms (e.g., Amazon MTurk).
In our work, we have observed that latency and throughput can be controlled with the crowd size and pricing dimensions. Optimal payment strategies, reward schemes, and incentive mechanisms for crowdsourcing have been studied [19, 29] and may also be applied in combination to crowdsourcing scheduling techniques in order to maximize throughput as well as the quality of the results.

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 [25], authors propose a scheduling approach to assign tasks to workers based on their history and how they learn doing 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, [25] 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 [26] 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: [20] shows, by means of simulations, how approaches that take into account worker skills outperform standard scheduling approaches, while [24] suggests scheduling tasks according to the required skills and the previous feedback from the requesters.

A different type of scheduling has been addressed in [9], 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.

The Effect of Switching Tasks. When scheduling tasks for the crowd, it is necessary to take the human dimension into account. Recent work [21] 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, [6]). In addition to this, in our work we show how context switch leads to an overall larger latency in work completion (Section 5.2) and propose scheduling techniques that take this human factor into account. The authors of [38] 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.

8. CONCLUSIONS

In a shared crowd-powered system environment, multiple users (or tenants) periodically issue queries that trigger predefined crowd-operators, resulting in independent crowdsourcing tasks published on the target crowdsourcing platform. In this paper, we pose and experimentally show that the divide strategy is not optimal, and that the crowd-powered system can increase its overall efficiency by bundling requests into a single batch that we call: HIT-BUNDLE, and then taking control of the distribution process of the tasks i.e., scheduling. 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.

Fairness is an important feature that shared environments (including multi-tenant crowd-powered systems) should support. Thus, we explored the problem of scheduling HITs using weighted Fair Scheduling algorithms, where priority is expressed as a function of price. However, human individuals behave very differently from machines, they are sensitive to the context switch that a regular scheduler might cause. The negative effects of context switching were visible in our experiments and are also supported 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 jobs.

To the best of our knowledge, this is the first piece of work giving crowd-powered systems control over their HIT execution schedule, with the goal of improving their overall efficiency. Our architecture, and its AMT-tailored implementation, can be leveraged in a number of ways for query optimization, and for powering complex SLAs.

9. ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation under grant number PP00P2 153023, by a Google Research Award, and by the UK EPSRC grant number EP/N011589/1.

10. REFERENCES
