# 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 linkage, 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 effiency 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.


*WWW 2016,* April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883030 .

## 1. INTRODUCTION

Thanks to micro-task crowdsourcing platforms such as Amazon Mechanical Turk[1] (AMT) and Crowdflower[2], 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 reposting 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.

[1] `http://mturk.com`
[2] `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 minimize 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,

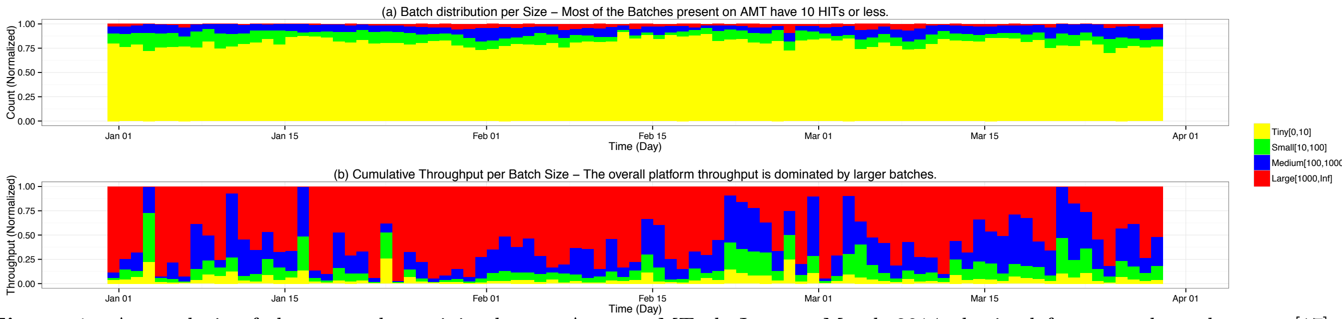


**Figure 1:** An analysis of three months activity log on Amazon MTurk January-March 2014 obtained from mturk-tracker.com [17] The crawler frequency is every 20 minutes, hence it might miss some batches. All HITs considered in this plot are rewarded \$0.01. Throughput measured in HIT/minute for HIT batches of different sizes.

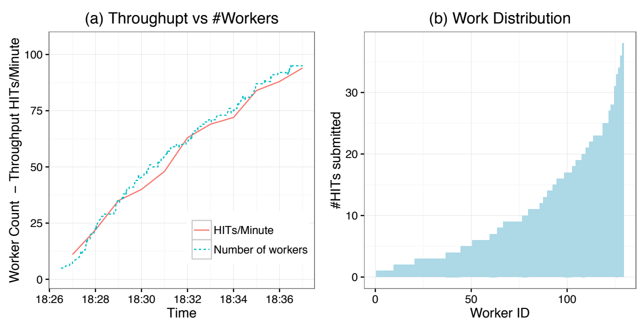


**Figure 2:** Results of a crowdsourcing experiment involving 100+ workers concurrently working on a batch of HITs. (a) Throughput (measured in HITs/minute) increases with an increasing number of workers involved. (b) Work Distribution: the amount of work done by each worker has a long tailed distribution.

since they have a high reward potential, without requiring to search for and select a new type of HIT. This hypothesis is confirmed by our experimental results (see Section 5).

# 3. THE ARCHITECTURE OF MULTI-TENANT CROWD-POWERED SYSTEMS

We now describe a new scheduling architecture that can be integrated to a typical crowd-powered system. We designed this architecture to operate on any crowdsourcing platform, though we will focus the remaining description on AMT as a target platform.

We study scheduling techniques for crowdsourcing on AMT by introducing the notion of `HIT-BUNDLE`, that is, a batch container where heterogeneous HITs of comparable complexity and reward get published continuously by the crowd-powered system on AMT. We show how the notion of `HIT-BUNDLE` not only permits to apply different scheduling strategies but also produces a higher overall throughput in Section 5.2.

Our general framework is depicted in Figure 3. The input to our system comes from the different crowdsourced queries submitted through a crowdsourcing interface. A `Crowdsourcing Decision 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 our `HIT Scheduler`. This contrasts to 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 availble 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 batchs 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 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 5, 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 $B_j$ of HITs. We define a *batch* $B_j = \{h_1, .., h_n\}$ as a set of HITs $h_i$. Each batch

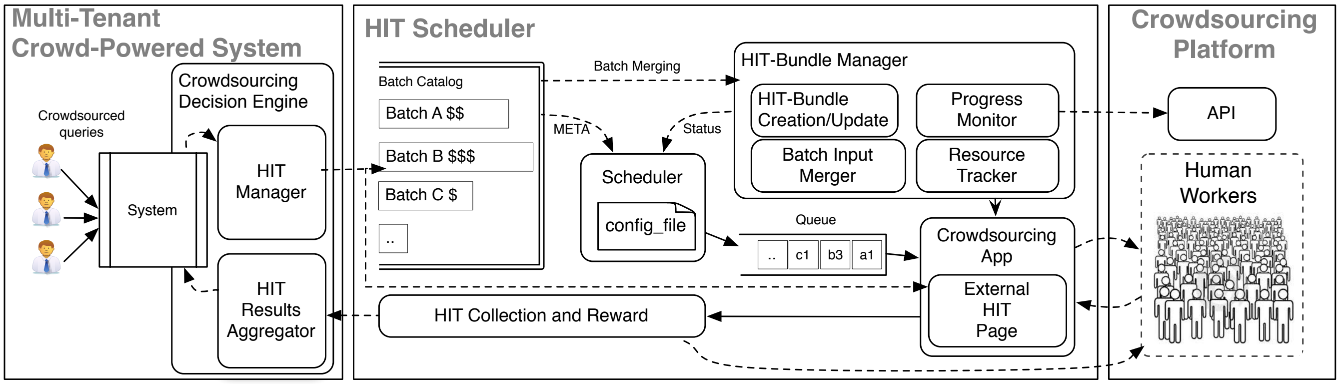


**Figure 3:** The role of the HIT Scheduler in a Multi-Tenant Crowd-Powered System Architecture.

has additional metadata attached to it: A monetary budget $M_j$ to be spent for its execution and a priority score $p_j$ 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 crowd sourcing 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, boringness, 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 crowdsourcing 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 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-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 biases, in the sense that they give an advantage to one batch over the others. Round Robin removes such biases 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 (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 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 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 priorities 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 complex problem that has been tackled by the cluster computing community. One popular approach currently used in Hadoop/Yarn is Fair Scheduling (FS) [16]. In the context 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 assigned 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

**Algorithm 1** Basic Fair Sharing

**Input:** $B = \{b_i < p_1, r_1 >, .., b_n < p_n, r_n >\}$ set of batches currently queued with priority $p_i$, and number of running HITs per batch $r_i$.
**Output:** HIT $h_i$.
1: When a worker is available for a task
2: $B_{Sorted}$ = Sort B by increasing $r_i$
3: $h_i = B_{Sorted}[0]$.getNextHit()
4: **return** $h_i$

from the jobs with the least $running\_tasks/task\_priority$ value. Algorithm1, line 2, gets in that case updated: `Sort B by increasing` $r_i/p_i$. This puts more weight on batches with few running tasks and a high priority.

The following formula gives the *fair share* of resources (i.e., number crowd workers) allocated to a HIT batch $j$ with priority score $p_j$ and concurrent running batches with priority scores $\{p_1..p_N\}$ at any given point in time

$$w_j = \frac{p_j}{\sum_1^N p_i}$$

## 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 Conscious Fair Sharing (WCFS) maximizes the likelihood of a worker receiving a task from a batch he worked on recently, 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 after a scheduling. This approach is the crowd-equivalent of Delay Scheduling [39].

# 5. EXPERIMENTAL EVALUATION

We describe in the following our experimental results obtained by scheduling HITs on the Amazon MTurk (AMT) crowdsourcing platform.

As a general experimental setup, we implemented the architecture 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 potential 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 summary of our dataset and provides a short description and

[3] `https://github.com/XI-lab/HIT-Scheduler`

**Algorithm 2** Worker Conscious Fair Share

**Input:** $B = \{b_i < p_1, r_1, s_i >, .., b_n < p_n, r_n, s_n >\}$ set of batches currently queued with priority $p_i$, $r_i$ number of running HITs, and $s_i$ concessions initialized to 0.
**Input:** K = maximum concession threshold
**Output:** HIT $h_i$.

When a worker $w_j$ is available for a HIT
$b_{last}$ = Last batch that $w_j$ did // null if it's a new worker
$B_{Sorted}$ = Sort B by increasing $r_i/p_i$
**if** $b_{last} == null$ **then**
  $B_{Sorted}[0]$.s = 0
  **return** $B_{Sorted}[0]$.getNextHit()
**end if**
**for** $b$ in $B_{Sorted}$ **do**
  **if** $b == b_{last}$ **then**
    b.s = 0
    **return** $b$.getNextHit()
  **else if** $b.s < K$ **then**
    b.s ++
    continue
  **else**
    b.s = 0
    $b$.getNextHit()
  **end if**
**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 `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 `HIT-BUNDLE`.

| 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. [33] | \$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. [32] | \$0.03 | 100 | 11sec |
| B5 | Spelling Correction | A collection of short paragraphs to spell check from StackExchange. | \$0.03 | 100 | 36sec |
| B6 | Butterfly Classification | Classify a butterfly image to one of 6 species (Admiral, Black Swallowtail, Machaon, Monarch, Peacock, and Zebra). [23] | \$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'). [35] | \$0.01 | 96 | 22sec |

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

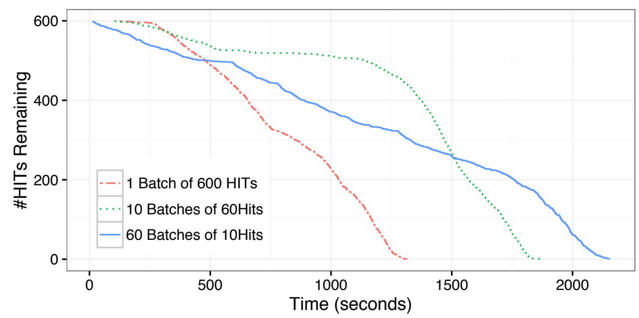


**Figure 4:** A performance comparison of batch execution time using different grouping strategies publishing a large batch of 600 HITs vs smaller batches (From B6).

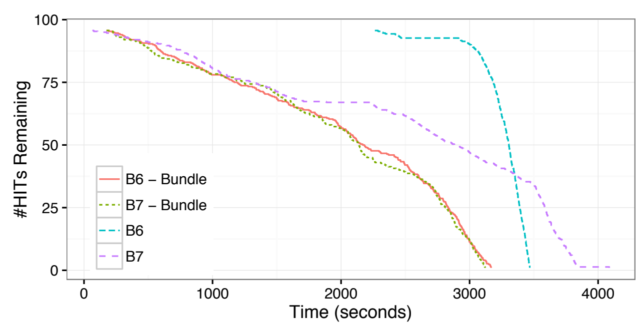


**Figure 5:** A performance comparison of batch execution time using different grouping strategies publishing two distinct batches of 192 HITs separately vs combined inside an `HIT-BUNDLE`.

### *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 `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 within the `HIT-BUNDLE` [4]. The results are depicted in Figure 5.

[4] We observe that AMT randomly selects the input to serve.

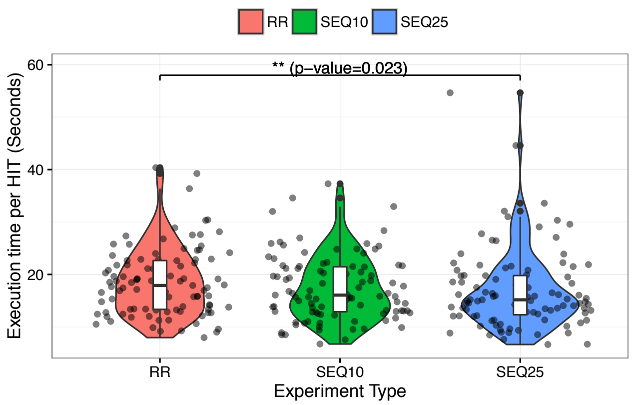


**Figure 6:** Average Execution time for each HIT submitted from the experimental groups RR, SEQ10 and SEQ25.

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.

### 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 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 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`.

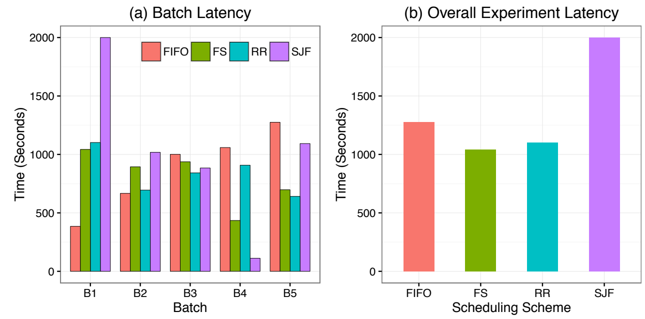


**Figure 7:** Scheduling approaches applied to the crowd.

### 5.3.1 *Controlled Experimental Setup*

In order 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 overtime[5].

In our controlled setting, each experiment that we run involves a number of crowd workers ranging between: $Min_w \leq |workforce| \leq Max_w$, 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 bellow $Min_w$, then the system starts accepting new sessions again.

Unless otherwise stated, we use the following configuration:

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

### 5.3.2 *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 $10 \leq |workforce| \leq 15$. Figure 7 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 just 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 a 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 fair-

[5]We decided not to run simulations, but rather to report the actual results obtained with human workers as part of the evaluated system.

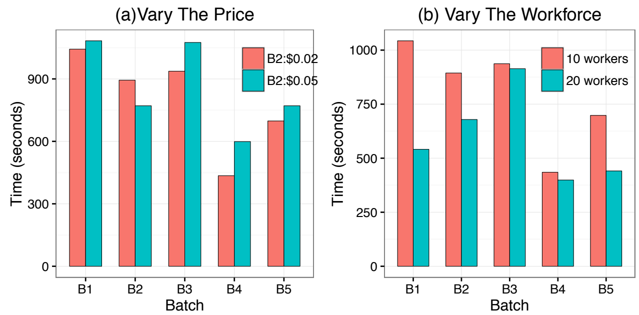


**Figure 8:** (a) Effect of increasing B2 priority on batch execution time. (b) Effect of varying the number of crowd workers involved in the completion of the HIT batches.

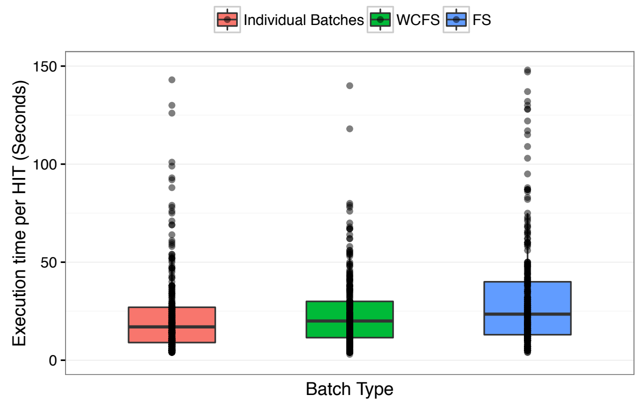


**Figure 9:** Average execution time per HIT under different scheduling schemes.

ness with respect to the different batches, i.e., there was no preferential treatment.

### 5.3.3 *Varying the Control Factors*

In order 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 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.

Another dimension that we vary is the crowd size. Figure 8b shows the batch completion time of two different crowdsourcing experiments when we vary the crowd size from $10 \leq |workforce| \leq 15$ to $20 \leq |workforce| \leq 25$ (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.

## 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

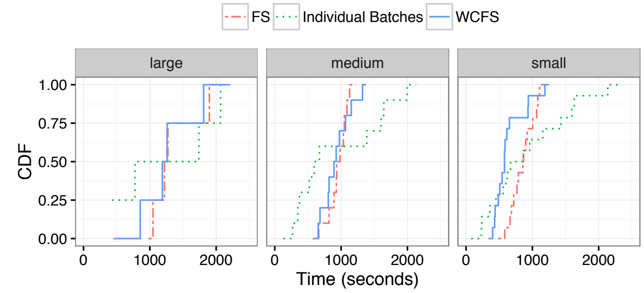


**Figure 10:** CDF of different batch sizes and scheduling schemes.

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 previously described trace as individual batches (standard approach) as well as into two `HIT-BUNDLE`s, 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. 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 spill over the remaining batches in the [11:25 - 11:35] time period. Our

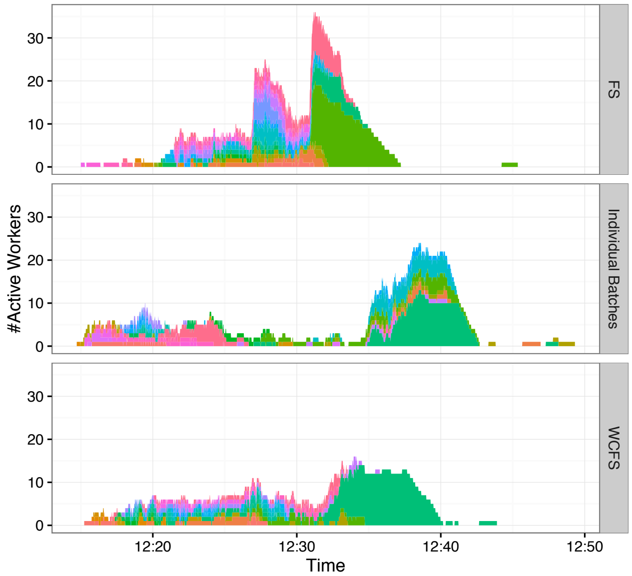


**Figure 11:** Worker allocation with FS, WCFS and classical individual batches in a live deployment of a large workload derived from crowdsourcing platform logs. Each color represents a different batch.

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 criterias 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

[1] O. Alonso and R. A. Baeza-Yates. Design and Implementation of Relevance Assessments Using Crowdsourcing. In *ECIR*, pages 153–164, 2011.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *Proceedings of the 19th*

*ACM international conference on Information and knowledge management*, pages 599–608. ACM, 2010.

[3] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW*, pages 839–848. ACM, 2012.

[4] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, et al. Vizwiz: nearly real-time answers to visual questions. In *UIST*, pages 333–342. ACM, 2010.

[5] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: expert finding in social networks. In *EDBT '13*, pages 637–648. ACM, 2013.

[6] M. J. Crump, J. V. McDonnell, and T. M. Gureckis. Evaluating amazon's mechanical turk as a tool for experimental behavioral research. *PloS one*, 8(3):e57410, 2013.

[7] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, pages 469–478, 2012.

[8] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *The VLDB Journal*, 22(5):665–687, 2013.

[9] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *Proc. SIGSPATIAL/GIS*, pages 324–333. ACM, 2013.

[10] E. Diaz-Aviles and R. Kawase. Exploiting twitter as a social channel for human computation. In *CrowdSearch*, pages 15–19, 2012.

[11] D. E. Difallah, M. Catasta, G. Demartini, and P. Cudré-Mauroux. Scaling-up the crowd: Micro-task pricing schemes for worker retention and latency improvement. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[12] D. E. Difallah, M. Catasta, G. Demartini, P. G. Ipeirotis, and P. Cudré-Mauroux. The dynamics of micro-task crowdsourcing: The case of amazon mturk. In *WWW*, pages 238–247. ACM, 2015.

[13] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell me what you like, and i'll tell you what to do. In *WWW*, pages 367–374, 2013.

[14] S. Faradani, B. Hartmann, and P. G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *Human Computation*, 2011.

[15] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD '11*, pages 61–72. ACM, 2011.

[16] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *NSDI'11*, pages 24–24. USENIX Association, 2011.

[17] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):16–21, 2010.

[18] P. G. Ipeirotis and E. Gabrilovich. Quizz: Targeted crowdsourcing with a billion (potential) users. In *WWW*, pages 143–154. ACM, 2014.

[19] R. Jurca and B. Faltings. Mechanisms for making crowds truthful. *J. Artif. Intell. Res. (JAIR)*, 34:209–253, 2009.

[20] R. Khazankin, H. Psaier, D. Schall, and S. Dustdar. Qos-based task scheduling in crowdsourcing environments. In *Service-Oriented Computing*, pages 297–311. Springer, 2011.

[21] W. S. Lasecki, A. Marcus, J. M. Rzeszotarski, and J. P. Bigham. Using Microtask Continuity to Improve Crowdsourcing. *Technical Report*, 2014.

[22] E. Law and L. v. Ahn. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–121, 2011.

[23] S. Lazebnik, C. Schmid, J. Ponce, et al. Semi-local affine parts for object recognition. In *British Machine Vision Conference (BMVC'04)*, pages 779–788, 2004.

[24] V. Nunia, B. Kakadiya, C. Hota, and M. Rajarajan. Adaptive Task Scheduling in Service Oriented Crowd Using SLURM. In *ICDCIT*, pages 373–385, 2013.

[25] V. Rajan, S. Bhattacharya, L. E. Celis, D. Chander, K. Dasgupta, and S. Karanam. Crowdcontrol: An online learning approach for optimal task scheduling in a dynamic crowd platform. In *ICML Workshop on 'Machine Learning meets Crowdsourcing'*, 2013.

[26] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das. Optimization in knowledge-intensive crowdsourcing. *CoRR*, abs/1401.1302, 2014.

[27] J. M. Rzeszotarski, E. Chi, P. Paritosh, and P. Dai. Inserting micro-breaks into crowdsourcing workflows. In *HCOMP (Works in Progress / Demos)*, volume WS-13-18 of *AAAI Workshops*. AAAI, 2013.

[28] N. Seemakurty, J. Chu, L. von Ahn, and A. Tomasic. Word sense disambiguation via human computation. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 60–63. ACM, 2010.

[29] A. Singla and A. Krause. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *WWW '13*, pages 1167–1178, 2013.

[30] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? *Communications of the ACM*, 53(1):64–71, 2010.

[31] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *SOCC '13*, pages 5:1–5:16. ACM, 2013.

[32] K. Vertanen and P. O. Kristensson. 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*, pages 295–298. ACM, 2011.

[33] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04*, pages 319–326. ACM, 2004.

[34] J. Wang, S. Faridani, and P. Ipeirotis. Estimating the completion time of crowdsourced tasks using survival analysis models. *Crowdsourcing for search and data mining (CSDM 2011)*, 31, 2011.

[35] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

[36] F. L. Wauthier and M. I. Jordan. Bayesian bias mitigation for crowdsourcing. In *NIPS*, pages 1800–1808, 2011.

[37] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question Selection for Crowd Entity Resolution. *Proc. VLDB Endow.*, 6(6):349–360, Apr. 2013.

[38] M. Yin, Y. Chen, and Y.-A. Sun. Monetary Interventions in Crowdsourcing Task Switching. In *Proceedings of the 2nd AAAI Conference on Human Computation (HCOMP)*, 2014.

[39] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys '10*, pages 265–278. ACM, 2010.

[40] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proc. VLDB Endow.*, 6(9):757–768, July 2013.