# Scaling-Up the Crowd: Micro-Task Pricing Schemes for Worker Retention and Latency Improvement

**Djellel Eddine Difallah**[†], **Michele Catasta**[*], **Gianluca Demartini**[†], **Philippe Cudré-Mauroux**[†]
[†] eXascale Infolab, University of Fribourg, Switzerland
[*]EPFL, Switzerland

## Abstract

Retaining workers on micro-task crowdsourcing platforms is essential in order to guarantee the timely completion of batches of Human Intelligence Tasks (HITs). Worker retention is also a necessary condition for the introduction of SLAs on crowdsourcing platforms. In this paper, we introduce novel pricing schemes aimed at improving the retention rate of workers working on long batches of similar tasks. We show how increasing or decreasing the monetary reward over time influences the number of tasks a worker is willing to complete in a batch, as well as how it influences the overall latency. We compare our new pricing schemes against traditional pricing methods (e.g., constant reward for all the HITs in a batch) and empirically show how certain schemes effectively function as an incentive for workers to keep working longer on a given batch of HITs. Our experimental results show that the best pricing scheme in terms of worker retention is based on punctual bonuses paid whenever the workers reach predefined milestones.

## Introduction

*"Companies that do a better job of attracting, developing, exciting, and retaining their talent will gain more than their fair share of this critical and scarce resource and will boost their performance dramatically"*

— Michaels et al., *The War for Talent*

Crowdsourcing is increasingly used in order to obtain large-scale human input for a wide variety of information management tasks. Common examples include relevance judgements (Carvalho, Lease, and Yilmaz 2011; Hosseini et al. 2012), image search (Yan, Kumar, and Ganesan 2010), data integration (Demartini, Difallah, and Cudré-Mauroux 2013; Wang et al. 2012). Such tasks are typically published on a crowdsourcing platform like Amazon MTurk[1], which acts as a market for online labor. The crowd workers are free to pick any of the publisher's tasks on the platform and submit their answers in exchange of micro-payments.



[1] http://www.mturk.com

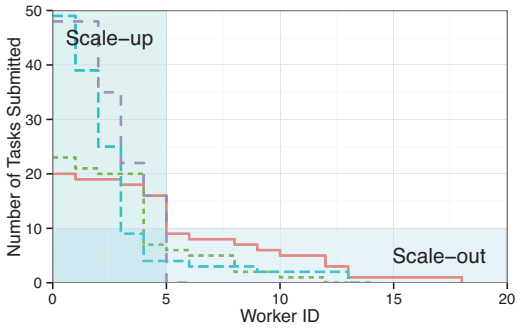


Figure 1: The classic distribution of work in crowdsourced tasks follows a long-tail distribution where few workers complete most of the work while many workers complete just one or two HITs.

The timely completion of a crowdsourcing campaign is however hardly guaranteed, as many factors influence its progression pace, including: the crowd availability and time-of-day (Ross et al. 2010; Ipeirotis 2010), the amount of the micro-payments (Faradani, Hartmann, and Ipeirotis 2011), the number of remaining tasks in a given batch, concurrent campaigns, or the reputation of the publisher (Irani and Silberman 2013). A common observation that is often made when running a crowdsourcing campaign on micro-task crowdsourcing platforms is the long-tail distribution of work done by people (Franklin et al. 2011; Demartini, Difallah, and Cudré-Mauroux 2012; Ipeirotis 2010): Many workers complete just one or a few HITs while a small number of workers do most of the HITs in a batch (see Figure 1). While this distribution has been repeatedly observed in a variety of settings, we argue in the following that it is hardly the optimal case from a batch latency point of view.

As shown in previous work (Faradani, Hartmann, and Ipeirotis 2011), long batches of Human Intelligence Task (HITs) submitted to crowdsourcing platforms tend to attract more workers as compared to shorter batches. A consequence of the long tail distribution of the workers, however, long batches tend to attract less workers towards their end—that is, when only a few HITs are left—as fewer workers are willing to engage with the almost-completed batch.

In this case, it is particularly important that current workers continue to do as much work as possible before they drop out and prompt the hiring of new workers for the remaining HITs. In addition, when workers become scarce (e.g., when the demand is high), such turnovers can become a serious obstacle to rapid batch completion.

Graceful *scalability* is an essential property of IT systems aiming at minimizing latency while answering increasing numbers of jobs concurrently. Some systems can be scaled *up* by introducing better hardware of software in order to cope with increasing demand. Many distributed systems, on the other hand, often support horizontal scalability (they scale *out*) by supporting the addition of additional nodes to handle more requests in parallel.

In an analogous way, we introduce in this paper two ways of scaling the execution of a crowdsourcing campaign:

- *scaling-out* the crowd: In this model, a large number of workers complete the tasks in parallel and compete for the next tasks. Under the assumption that a large crowd of workers is always available to handle the different tasks at hand, this model can minimize the batch execution time by increasing the competition among workers.
- *scaling-up* the crowd: A different way of scaling a crowdsourcing campaign is to focus on attaining higher worker retention rates such that they keep working longer on a given batch. This model potentially presents two advantages: It minimizes the down times incurred when waiting for new workers, and yields potentially better workers having more experience handling a given task.

Both approaches are valid ways to reduce the latency of the crowd, reduce the overall execution time, and improve the global efficiency of an hybrid human-machine information system. It is worth noting that under both regimes the system could reach its optimum, that is, with a sufficiently large crowd where each individual is highly motivated to finish a batch that he engages with, the execution time converges to a minimum.

Scaling out the crowd is difficult on current micro-task crowdsourcing platforms where the number of workers is strictly limited[2]. Instead, we propose to focus on scaling up the crowd in the following and introduce a series of new incentive schemes towards that goal. Our fundamental goal is to retain workers working on a specific batch of HITs as long as possible in order to minimize the overall batch execution time. The set of schemes that we experimentally compare include a uniform pricing scheme baseline that assigns a fixed price to every HIT in a batch, as well as a set of bonus pricing schemes that vary the amount of money paid for a HIT over the duration of the crowdsourcing campaign.
Experiments run with crowd workers from the Amazon MTurk platform over different task types show that our proposed pricing strategies perform differently depending on the task: For example, for tasks that require some training and are tedious at first, overpaying workers at the beginning of a batch works well in practice as it encourages the workers to persevere and continue beyond the initial learning phase.

In summary, the main contributions of this paper are:

- A novel crowdsourcing optimization problem focusing on *scaling-up* the crowd and retaining workers longer in order to minimize the execution time of long batches.
- A set of new incentives schemes focusing on making individual workers more engaged with a given batch of HITs in order to improve worker retention rates.
- An open-source software library to embed the proposed schemes inside current HIT interfaces[3].
- An extensive experimental evaluation of our new techniques over different tasks on a state-of-the-art crowdsourcing platform.

The rest of the paper is structured as follows: In the next section we review related work focusing on contributions related to pricing schemes for crowdsourcing platforms and their effects on the behavioral patterns of the workers. Next, we formally define the problem and introduce different pricing schemes to retain workers longer on a set of HITs given a fixed monetary budget. The following section presents empirical results comparing the efficiency of our different pricing schemes and discussing their effect on crowd retention and overall latency. Finally, we discuss the main findings from our experimental evaluation and conclude the paper.

## Related Work

A number of recent contributions studied the effect of monetary incentives on crowdsourcing platforms. In (Mao et al. 2013), Mao *et al.* compared crowdsourcing results obtained using both volunteers and paid workers. Their findings show that the quality of the work performed by both populations is comparable, while the results are obtained faster when the crowd is financially rewarded.

Wang *et al.* (Wang, Ipeirotis, and Provost 2013) looked at pricing schemes for crowdsourcing platforms focusing on the quality dimension: The authors proposed methods to estimate the quality of the workers and introduce new pricing schemes based on the expected contribution of the workers. While also proposing an adaptive pricing strategy for micro-task crowdsourcing, our work focuses instead on retaining the crowd longer on a given batch of tasks in order to improve the efficiency of individual workers and to minimize the overall batch execution time.

Another recent piece of work (Lasecki et al. 2014) analyzed how task interruption and context switching decreases the efficiency of workers while performing micro-tasks on a crowdsourcing platform. This motivates our own work, which aims at providing new incentives to convince the workers to keep working longer on a given batch of tasks.

Chandler and Horton (Chandler and Horton 2011) analyzed (among others) the effect of financial bonuses for crowdsourcing tasks that would be ignored otherwise. Their results show that monetary incentives worked better than non-monetary ones given that they are directly noticeable

[2]MTurk has for example been declining new workers for some time in order to keep a healthy balance between the number of workers and the number of tasks posted on the platform.

[3]A library based on the Django framework available at: https://github.com/XI-lab/BonusBar

by the workers. In our own work, we display bonus bars on top of the task to inform the worker on his/her hourly rate, fixed pay, and bonuses for the current HITs.

Recently also, Singer *et al.* (Singer and Mittal 2013) studied the problem of pricing micro-tasks in a crowdsourcing marketplace under budget and deadline constraints. Our approach aims instead at varying the price of individual HITs in a batch (i.e., by increasing or decreasing the monetary rewards) in order to retain workers longer.

Faradani *et al.* (Faradani, Hartmann, and Ipeirotis 2011) studied the problem of predicting the completion of a batch of HITs and at its pricing given the current marketplace situation. They proposed a new model for predicting batch completion times showing that longer batches attract more workers. In comparison, we experimentally validate our work with real crowd workers completing HITs on a micro-task crowdsourcing platform (i.e., on Amazon MTurk).

In (Mao, Kamar, and Horvitz 2013), Mao *et al.* looked into crowd worker engagement. Their work is highly related to ours as it aims to characterize how workers perceive tasks and to predict when they are going to stop performing HITs. The main difference with our work is that (Mao, Kamar, and Horvitz 2013) looked at a volunteer crowdsourcing setting (i.e., they used data from Galaxy Zoo where people classify pictures of galaxies). This is a key difference as our focus is specifically on finding the right pricing scheme (i.e., the correct financial reward) to engage workers working on a batch of HITs.

Another setting where retaining workers is critical is push crowdsourcing. Push crowdsourcing (Difallah, Demartini, and Cudré-Mauroux 2013) is a special type of micro-task platform where the system assigns HITs to selected workers instead of letting them do any available HIT on the platform. This is done to improve the effectiveness of the crowd by selecting the right worker in the crowd for a specific type of HIT based on the worker profile which may include previous HITs history, skills and preferences. Since attracting the desired workers is not guaranteed, keeping them on the target task is essential.

On a separate note, this piece of work was also inspired from studies on talent management in corporate settings. Companies have long realized the shortage of highly qualified workers and the fierce competition to attract top talents. In that context, retaining top-performing employees longer constitutes an important factor of performance and growth (Bartlett and Ghoshal 2013; Michaels, Handfield-Jones, and Axelrod 2001). Although our present setting is radically different from traditional corporate settings, we identified many cases where the crowdsourcing requesters (acting as a virtual employer) could use common human resources practices. In the following, we particularly investigate practices such as: training cost, bonuses, and attribution of qualifications (Huselid 1995; Arthur 2001).

## Worker Retention Schemes

Our main retention incentive is based on compensation of workers engaged in a batch of tasks, using monetary bonuses and qualifications. We start this section by formally characterizing our problem below. We then introduce our various pricing schemes, before describing the visual interface we implemented in order to inform the workers of the monetary rewards, and the different types of tasks we considered for the HITs.

### Problem Definition

Given a fixed retention budget $B$ allocated to pay workers $w_1, \ldots, w_m$ to complete a batch of $n$ analogous tasks $H = \{h_1, \ldots, h_n\}$, our task is to allocate $B$ for the various HITs in the batch in order to maximize the average number of tasks completed by the workers. More formally, our goal is to come up with a function $b(h)$ which, for each $h_j \in H$ gives us the optimal reward upon completion of $h_j$ such as to maximize the average number of tasks completed by the workers, i.e.,

$$b(h)_{opt} = \underset{b(h)}{\operatorname{argmax}}\, m^{-1}n^{-1} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \mathbb{1}_{C(w_i,h_j,b(h_j))}$$

where $\mathbb{1}_{C(w_i,h_j,b(h_j))}$ is an indicator function equal to 1 when worker $w_i$ completed task $h_j$ under rewarding regime $b(h)$, and to 0 otherwise. For simplicity, we assume in the following that workers complete their hits sequentially, i.e., that $\forall h_i, h_j \in H$ , $h_i$ is submitted before $h_j$ if $i < j$, though they can drop out at any point in time in the batch.

### Pricing Functions

**Fixed Bonus** The standard pricing scheme used in micro-task crowdsourcing platforms like Amazon MTurk is *uniform pricing*. Under this regime, the worker receives the same monetary bonus for each completed task in the batch:

$$b(h_i) = \frac{B}{|H|} \; \forall \; h_i \in H \tag{1}$$

**Training Bonus** Instead of paying the same bonus for each task in a batch, one might try to overpay workers at the beginning of the batch in order to make sure that they do not drop out early as is often the case. This scheme is especially appealing for more complex tasks requiring the workers to learn some new skill initially, making the first HITs less appealing due to the initial overhead. This scheme allows the requester to compensate the implicit training phase by initially fixing a high hourly wage despite the low productivity of the worker. Many different reward functions can be defined to achieve this goal. In our context, we propose a linearly *decreasing pricing* scheme as follows:

$$b(h_i) = \frac{B}{|H|} + \left( \left\lceil \frac{|H|}{2} \right\rceil - i \right) \cdot \left( \frac{B}{|H|} \cdot \frac{2}{|H|} \right) \tag{2}$$

where we add to the average HIT reward $B/|H|$ a certain *bonus payment increment* (i.e., the last term of the equation) a certain number of times based on the current HIT in the batch. The general idea behind this scheme is to distribute the available budget in a way that HITs are more rewarded

at the beginning and such that the bonus incrementally decreases after that. One potential advantage of this pricing scheme is the possibility to attract many workers to the batch due to the initial high pay. On the other hand, retention may not be optimal since workers could drop out as soon as the bonus gets too low.

**Increasing Bonus** By flipping the $(+)$ sign in Equation 2 into a $(-)$, we obtain the opposite effect, that is, a pricing scheme with *increasing reward* over the batch length. That way, the requesters are overpaying workers towards the end of the batch instead of at the beginning. This approach potentially has two advantages: First, as workers get increasingly paid as they complete more HITs in the batch, they might be motivated to continue longer in order to complete the most rewarding HITs at the very end of the batch. Second, workers get rewarded for becoming increasingly trained in the type of task present in the batch. On the other hand, a possible drawback of this scheme is the fairly low initial appeal of the batch due to the low bonuses granted at first.

**Milestone Bonus** In all the previous schemes, bonuses are attributed after each completed HIT. However, depending on the budget and the exact bonus function used, the absolute value of the increments can be very small. To generate bigger bonuses, one could instead try to accumulate increments over several HITs and distribute bonuses occasionally only. Following this intuition, we introduce in the following the notion of *milestone bonuses*. Under this regime, an accumulated bonus is rewarded punctually after completing a specific number of tasks. For a fixed interval $I$, $I <= n$, we formulate this scheme using the following function:

$$b(h_i) = \begin{cases} \left\lceil \frac{B \cdot I}{|H|} \right\rceil & \text{if } i \bmod I = 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

**Qualifications** In addition to the monetary reward that is offered at each interval, the requester can define a qualification level that can be granted after each milestone. Qualifications are a powerful incentive as they constitute a promise on exclusivity for future work.

**Random Bonus** An additional scheme that we consider is the attribution of a bonus drawn once at random[4], from a predefined distribution of the total retention budget $B$. In particular, we consider the *Zipf* distribution in order to create a lottery effect so that a worker can get a high bonus at any point while progressing through the batch.

## Visual Reward Clues

On current micro-task crowdsourcing platforms such as Amazon Mechanical Turk, one can implement the above rewarding schemes by allocating bonuses for the HITs. Hence, workers complete HITs in a batch in exchange of to the usual fixed reward, but get a bonus that possibly varies from one HIT to another. In order to make this scheme clear to the workers, we decided to augment the HIT interface with a *Bonus Bar*, an open-source toolkit that requesters can easily integrate with their HITs[5]. Figure 2 gives a visual rendering of the payment information displayed to a worker completing one of our HITs.

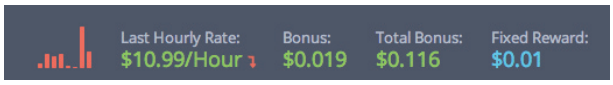


Figure 2: Screenshot of the Bonus Bar used to show workers their current and total reward.

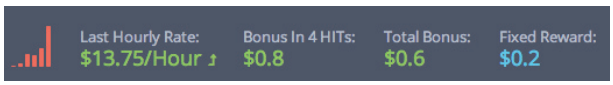


Figure 3: Screenshot of the Bonus Bar with next milestone and bonus.

## Pricing Schemes for Different Task Types

We hypothesize that the pricing schemes proposed above perform differently based on the task at hand. In that sense, we decided to address three very different types of tasks and to identify the most appropriate pricing scheme for each type in order to maximize worker retention.

The first distinction we make for the tasks is based on their length: Hence, we differentiate short tasks that only require few seconds each (e.g., matching products) and longer tasks that require one minute or more (e.g., searching the Web for a customer service phone number). Note that in any case we only consider micro-tasks, that is, tasks requiring little effort to be completed by individuals and that can be handled in large batches.

The second distinction we make is based on whether or not the task require some sort or initial training. The example we decided to pick for this paper is the classification of butterfly images in a predefined set of classes. We assume that at the beginning of the batch the worker is not confident in performing the task and repeatedly needs to check the corresponding Wikipedia pages in order to correctly categorize the various butterflies. After a few tasks, however, most worker will have assimilated the key differentiating features of the butterflies and will be able to perform the subsequent tasks much more efficiently. For such tasks, we expect the training bonus scheme to be particularly effective since it overpays the worker at the beginning of the batch as he/she is spending a considerable amount of type to complete each HIT. After the worker gets trained, one can probably lower the bonuses while still maintaining the same hourly reward rate.

[4]The attributed bonus value is removed from the distribution's list to insure that the budget limit is met.

[5]Specifically, MTurk requesters can use the toolkit by means of the *ExternalQuestion* data structure: That is, an externally hosted Web form which is embedded in the MTurk webpages.

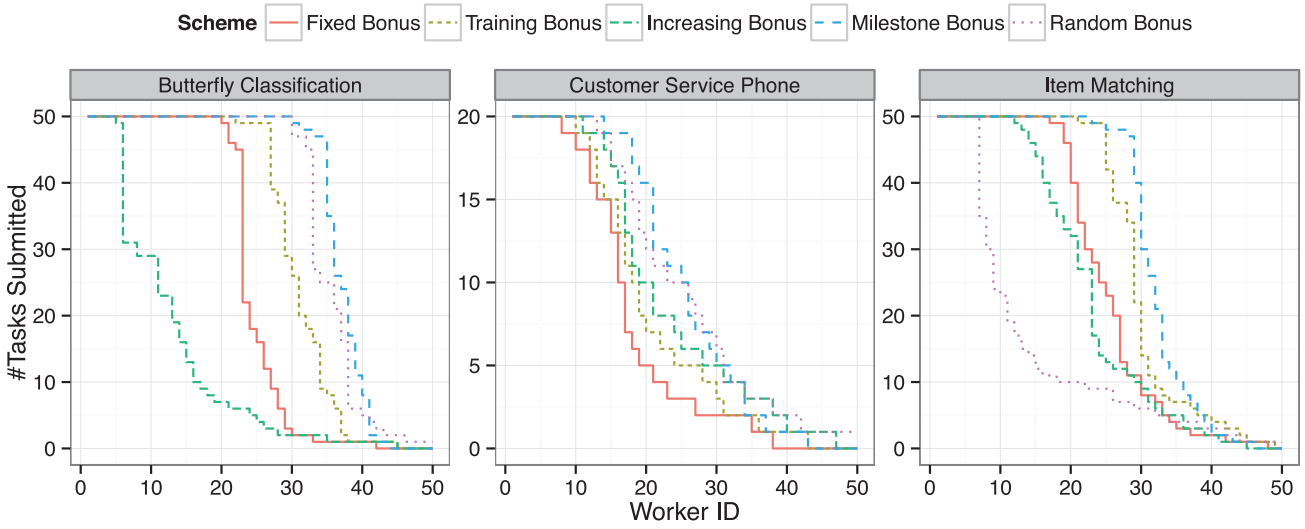


Figure 4: Effect of different bonus pricing schemes on worker retention over three different HIT types. Workers are ordered by the number of completed HITs.

Table 1: Statistics for the three different HIT types.

| Batch Type | #Workers | #HITs | Base Budget | Bonus Budget | Avg. HIT Time | Avg. Hourly Rate |
|---|---|---|---|---|---|---|
| Item Matching | 50 | 50 | $0.5 | $0.5 | 22sec | $5.3/hr |
| Butterfly Classification | 50 | 50 | $0.5 | $0.5 | 15sec | $9.4/hr |
| Customer Care Phone Number Search | 50 | 20 | $0.2 | $0.4 | 78sec | $2.2/hr |

# Experimental Evaluation

## Experimental Setup

In order to experimentally compare the different pricing schemes we introduced above, we consider three very different tasks:

- *Item Matching*: Our first batch is a standard dataset of HITs (already used in (Wang et al. 2012)) asking workers to uniquely identify products that can be referred to by different different names (e.g., 'iPad Two' and 'iPad 2nd Generation').
- *Butterfly Classification*: This is a collection of 619 images of six types of butterflies: Admiral, Black Swallowtail, Machaon, Monarch, Peacock, and Zebra (Lazebnik et al. 2004). Each batch of HITs uses 50 randomly selected images from the collection that are presented to the workers for classification.
- *Customer Care Phone Number Search*: In this batch, we ask the workers to find the customer-care phone number of a given US-based company using the Web.

Our first task is composed of relatively simple HITs that do not require the workers to leave the HIT page but just to take a decision based on the information displayed. Our second task is more complex as it requires to classify butterfly images into predefined classes. We assume that the workers will not be familiar with this task and will have to learn about the different classes initially. In that sense, we provide workers with links to Wikipedia pages describing each of the butterfly species. Our third task is a longer task that requires no special knowledge but rather to spend some time on the Web to find the requested piece of information.

Table 1 gives some statistics for each task, including the number of workers and HITs we considered for each task (always set to 50 for both), the base and bonus budgets, and the resulting average execution times and hourly rates. All the tasks were run on the Amazon MTurk platform.

Our main experimental goals are i) to observe the impact of our different pricing schemes on the total number of tasks completed the workers in a batch (worker retention) and ii) to compare the resulting batch execution times. Hence, the first goal of our experiments is not to complete each batch but rather to observe how long workers keep working on the batch. Towards that goal, we decided to recruit exactly 50 distinct workers for each batch, and do not allow the workers to work twice on a given task. We build the backend such that each worker works on his/her HITs in isolation without any concurrency. This is achieved by allowing 50 repetitions per HIT and recording the worker Id the first time the HIT is accepted, once the count of Ids reaches 50, any new comer is asked not to accept the HIT. All batches were started at random times during the day and left online long enough to alleviate any effect due to the timezones.

## Experimental Results

**Worker Retention** Figure 4 shows the effect of the different pricing schemes on worker retention for the different types of HITs we consider in this work. The first observa-

tion we can make is that the pricing scheme based on the Milestone Bonus that grants rewards when reaching predefined goals performs best in terms of worker retention: more workers complete the batch of tasks as compared to other pricing schemes over all the different task types.

Another observation is that in the Butterfly Classification task the training bonus pricing scheme retains workers better than the increasing or the fixed bonus scheme. This supports our assumption that overpaying workers at the beginning of the batch while they are learning about the different butterfly classes helps them feeling rewarded for the learning effort and helps keeping them working on the batch longer.

On the other hand, the increasing pricing scheme performed worse both for the Item Matching and the Butterfly Classification batches. This is probably the case as workers did feel underpaid for the work they were doing and preferred to drop the batch before its end.

The final comment is about the fixed pricing scheme: This shows bad performance in terms of worker retention over all the task types we have considered. Note that this is the standard payment scheme used in paid micro-task crowdsourcing platforms like Amazon MTurk where each HIT in a batch is rewarded equally for everyone independently on how many other HITs the workers has performed in the batch.

**Learning Curve** We report on how the execution time varies across the different task types in Figure 5. We group the results based on three different classes of workers: a) the **Short** category, which includes workers having completed 25% or less tasks in the batch, b) the **Medium** category, which includes workers having completed between 25% and 75% of the HITs in the batch, and c) the **Long** category, which includes those workers who completed more than 75% of the tasks.

From the results displayed in Figure 5, we observe a significant learning curve for the Butterfly Classification batch: On average, the first tasks in the batch require workers a substantially longer time to complete as compared to the final ones. For the Customer Care Phone Number Search batch, we see that the task completion time varies from HIT to HIT. We also note that workers who remained until the end of the batch are becoming slightly faster over time. The Item Matching batch shows a similar trend, where tasks submitted towards the end of the batch require on average less time than those submitted initially. Across the different types of tasks, we also note that workers who are categorized as Short always start slower than others on average (i.e., workers dropping out early are also slower initially). This is hence an interesting indicator of potential drop-outs.

These results are particularly important for our goal of improving latency, since the retained workers tend to get faster with new HITs performed. This gain is expected to have a direct impact on the overall execution time of the batch. Next, we check whether this has an impact on the quality of the submitted HITs.

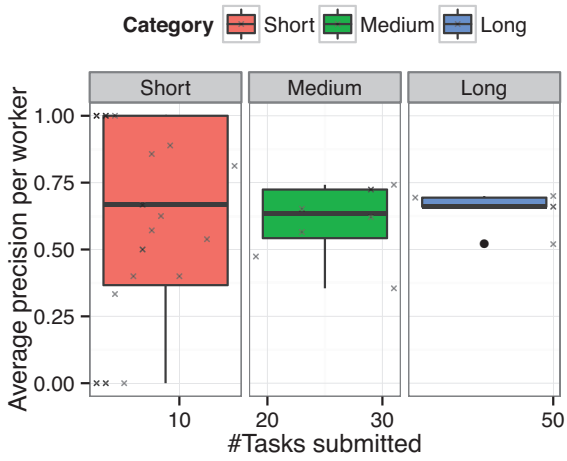


Figure 6: Overall precision per worker and category of worker for the Butterfly Classification task (using Increasing Bonus).

**Impact on Work Quality** We report on the quality of the crowdsourced results in Figure 6. We observe that the average precision of the results does not vary across workers who perform many or few tasks. We observe however that the standard deviation is higher for the workers dropping early than for those working longer on the batches. In addition, those workers who perform most of the HITs in the batch never yield low precision results (the bottom right of the plot is empty). This could be due to a self-selection phenomenon through which workers who perform quite badly at the beginning of the batch decide to drop out early.

## Efficiency Evaluation

In this final experiment, we evaluate the impact of our best approach (Milestone Bonus) on the end-to-end execution of a batch of HITs, and we compare with a) the classical approach with no bonus, and b) using the bonus budget to increase the base reward. In order to get independent and unbiassed results, we decided to create a new task for this experiment[6], which consists in correcting 10 english essays from the ESOL dataset (Yannakoudakis, Briscoe, and Medlock 2011). We run the three batches on MTurk, each having 10 HITs and requiring 3 repetitions, that is, 3 entries are required from different workers for each HIT. A summary of our setting is shown in Table 2. The three setups differ as follows:

- **Batch A(Milestones)**: Workers who select Batch A are presented with the interface displaying the Bonus Bar configured with a bonus at 3, 6 and 10 HITs milestones offering respectively ($0.2), ($0.4),($0.8) bonuses for a maximum retention budget of $1.4*3=$4.2.
- **Batch B(Classic)**: Workers who select Batch B are presented with a classical interface and receive a fixed reward of $0.2 for each submission they make.

[6]In the previous set of experiments, we hired more than 450 distinct workers.


Figure 5: Average of the HITs execution time with standard error ordered by their sequence in the batch. Results are grouped by worker category (long, medium and short term workers). In many cases, the *Long* term workers improve their HIT time execution. This is expected to have a positive impact on the overall batch latency.

- **Batch C(High Reward)**: Workers who select Batch C are presented with a classical interface. Here, we use the bonus budget to increase the base reward, thus, workers will receive a fixed reward of $0.34 for each submission they make.

We perform 5 repeated runs as follows: a) we start both batches A and B at the same time and let them run concurrently – this measures the sole effect of retention, b) batch C was launched separately since it offers a higher base reward and might influence A and B[7].

Figure 7 shows the results of 5 repeated experiments of the above settings. We report the overall execution time after each batch finishes (i.e., when all the 3*10 HITs are submitted), the budget used by each run, the number of workers involved and how many HITs each worker submitted. We can observe the effects of retention in batch A as it involves less workers who submit a greater number of HITs on average as compared to batches B and C. From a latency perspective, batch A consistently outperforms batch B's execution time, on average by 33%, thanks to the retention budget in use. While batch C is faster overall – which can be explained by the fact that it attracts more workers due to its high reward – it uses the entirety of its budget, as compared to A that only uses $2.44 on average.

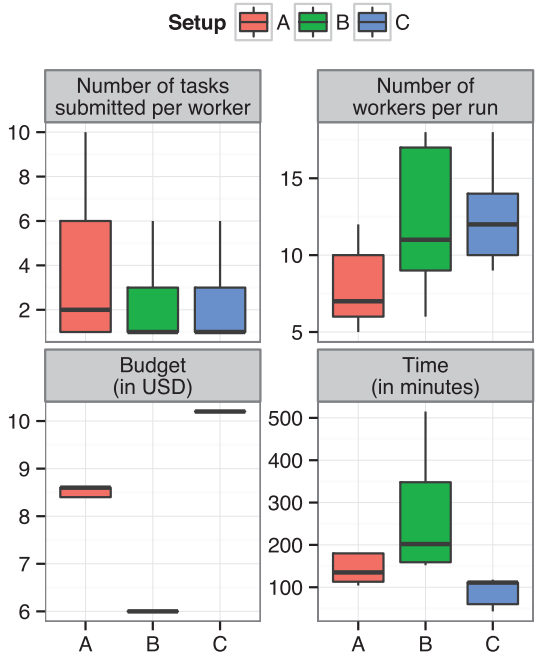


Figure 7: Results of five independent runs of A, B and C setups. Type A batches include the retention focused incentive while Type B is the standard approach using fixed pricing, Batch C uses a higher fixed pricing – but leveraging the *whole* bonus budget.

## Discussion

To summarize, the main findings resulting from our experimental evaluation are:

- Giving workers a punctual bonus for reaching a predefined objective defined as a given number of tasks im-

[7]To minimize timezones effects we run the batch at a similar time of the day as A and B

Table 2: Statistics of the second experimental setting – English Essay Correction

| Batch Type | #HITs | #Repetitions | Reward | Base Budget | Bonus Budget | Avg. HIT Time | Avg. Hourly Rate |
|---|---|---|---|---|---|---|---|
| A (Milestones) | 10 | 3 | $0.2 | $6 | $4.2 | 268sec | $5.7/hr |
| B (Classic) | 10 | 3 | $0.2 | $6 | N/A | 310sec | $2.4/hr |
| C (High Reward) | 10 | 3 | $0.34 | $10.2 | N/A | 302sec | $3.9/hr |

proves worker retention.

- Overpaying workers at the beginning of a batch is useful in case the tasks require an initial training: Workers feel rewarded for their initial effort and usually continue working for a lower pay after the learning phase.
- While retention comes at a cost, it also improves latency. Based on our experiments comparing different setups over multiple runs, we observe that the bonus scheme involved less workers who perform more tasks on average. This property is particularly important when the workforce is limited on the crowdsourcing platform.

## Conclusions

In this work, we addressed the problem of scaling-up the crowd, that is, of incentivizing workers such that they keep working longer on a given batch of HITs. Increased worker retention is valuable in order to avoid the problem of batch starvation (when only a few remaining HITs are left in a batch and no worker selects them), or if the workforce is limited on the crowdsourcing platform (a requester tries to keep the workers longer on his batch). We defined the problem of worker retention and proposed a variety of bonus schemes in order to maximize retention, including fixed, random, training, increasing, and milestone-based schemes. We performed an extensive experimental evaluation of our approaches over real crowds of workers on a popular microtask crowdsourcing platform. The results of our experimental evaluation show that the various pricing schemes we have introduced perform differently depending of the type of tasks. The best performing pricing scheme in terms of worker retention is based on milestone bonuses, which are punctually given to the workers who reach a predefined goal in terms of completed number of HITs.

We also observe that our best bonus schemes consistently outperform the classic fixed pricing scheme, both in terms of worker retention and efficient execution. The main finding of this paper is hence that it is possible to adopt new pricing schemes in order to make workers stick to their batch of tasks longer and obtain results faster back from the crowdsourcing platform. We believe that this is a key finding in the context of hybrid human-machine systems, where most of the latency is incurred through human computation, and a step towards providing crowdsourcing SLAs.

## Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work was supported by the Swiss National Science Foundation under grant number PP00P2_128459.

## References

Arthur, D. 2001. *The employee recruitment and retention handbook*. AMACOM Div American Mgmt Assn.

Bartlett, C., and Ghoshal, S. 2013. Building competitive advantage through people. *Sloan Mgmt. Rev* 43(2).

Carvalho, V. R.; Lease, M.; and Yilmaz, E. 2011. Crowdsourcing for search evaluation. In *ACM Sigir forum*, volume 44, 17–22. ACM.

Chandler, D., and Horton, J. J. 2011. Labor Allocation in Paid Crowdsourcing: Experimental Evidence on Positioning, Nudges and Prices. In *Human Computation*.

Demartini, G.; Difallah, D. E.; and Cudré-Mauroux, P. 2012. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*, 469–478. ACM.

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

Difallah, D. E.; Demartini, G.; and Cudré-Mauroux, P. 2013. Pick-a-crowd: tell me what you like, and i'll tell you what to do. In *Proceedings of the 22nd international conference on World Wide Web*, 367–374. International World Wide Web Conferences Steering Committee.

Faradani, S.; Hartmann, B.; and Ipeirotis, P. G. 2011. What's the Right Price? Pricing Tasks for Finishing on Time. In *Human Computation*.

Franklin, M. J.; Kossmann, D.; Kraska, T.; Ramesh, S.; and Xin, R. 2011. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, 61–72. New York, NY, USA: ACM.

Hosseini, M.; Cox, I. J.; Milić-Frayling, N.; Kazai, G.; and Vinay, V. 2012. On aggregating labels from multiple crowd workers to infer relevance of documents. In *Advances in information retrieval*. Springer. 182–194.

Huselid, M. A. 1995. The impact of human resource management practices on turnover, productivity, and corporate financial performance. *Academy of management journal* 38(3):635–672.

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

Irani, L. C., and Silberman, M. 2013. Turkopticon: Interrupting worker invisibility in amazon mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 611–620. ACM.

Lasecki, W. S.; Marcus, A.; Tzeszotarski, J. M.; and Bigham, J. P. 2014. Using Microtask Continuity to Improve Crowdsourcing. In *Carnegie Mellon University Human-Computer Interaction Institute - Technical Reports - CMU-HCII-14-100*.

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

Mao, A.; Kamar, E.; Chen, Y.; Horvitz, E.; Schwamb, M. E.; Lintott, C. J.; and Smith, A. M. 2013. Volunteering Versus Work for Pay: Incentives and Tradeoffs in Crowdsourcing. In *HCOMP*.

Mao, A.; Kamar, E.; and Horvitz, E. 2013. Why Stop Now? Predicting Worker Engagement in Online Crowdsourcing. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Michaels, E.; Handfield-Jones, H.; and Axelrod, B. 2001. *The war for talent*. Harvard Business Press.

Ross, J.; Irani, L.; Silberman, M.; Zaldivar, A.; and Tomlinson, B. 2010. Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, 2863–2872. ACM.

Singer, Y., and Mittal, M. 2013. Pricing Mechanisms for Crowdsourcing Markets. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, 1157–1166. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee.

Wang, J.; Kraska, T.; Franklin, M. J.; and Feng, J. 2012. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment* 5(11):1483–1494.

Wang, J.; Ipeirotis, P. G.; and Provost, F. 2013. Quality-Based Pricing for Crowdsourced Workers. In *NYU Stern Research Working Paper - CBA-13-06*.

Yan, T.; Kumar, V.; and Ganesan, D. 2010. Crowdsearch: Exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, 77–90. New York, NY, USA: ACM.

Yannakoudakis, H.; Briscoe, T.; and Medlock, B. 2011. A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 180–189. Association for Computational Linguistics.