A 10-Month-Long Deployment Study of On-Demand Recruiting for Low-Latency Crowdsourcing

Ting-Hao (Kenneth) Huang **Jeffrey P. Bigham**
Carnegie Mellon University, Pittsburgh, PA, USA.
{tinghaoh,jbigham}@cs.cmu.edu

Abstract

A number of interactive crowd-powered systems have been developed to solve difficult problems out of reach for automated solutions. To work interactively, such systems need access to on-demand labor. To meet this demand, workers can be *(i)* recruited when needed directly from the crowd marketplace, or *(ii)* recruited in advance and asked to wait in a *retainer pool* until they are needed. Most of the evaluations of these systems have been over a short time period, even though we know that marketplaces change and adapt over time. In this paper, we present the results of a 10-month deployment of a crowd-powered system that uses a hybrid approach to fast recruitment of workers that we call *Ignition*. We describe the Ignition approach and the observed times required to recruit workers from the marketplace and retainer over this long period of time. Our results demonstrate that it is possible to recruit workers with low latency even over long periods, and suggest a number of opportunities for future work for recruitment strategies and modeling that may further improve on-demand recruitment for deployed systems.

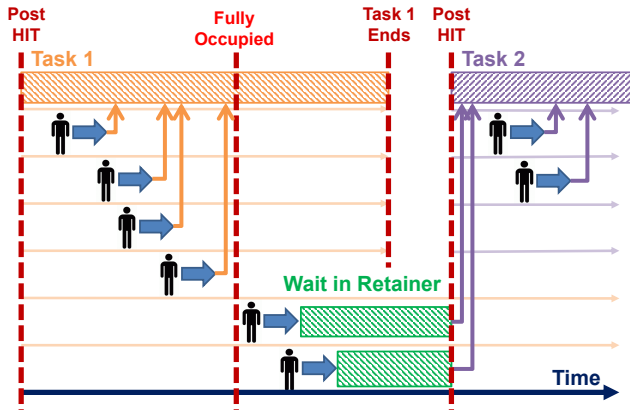

Figure 1: Ignition combines on-demand recruiting of workers with a retainer to affordably recruit workers quickly to power deployed crowd systems.

Introduction

A number of interactive crowd-powered systems have been developed to solve difficult problems out of reach for automated solutions. For instance, Soylent used the crowd to edit and proofread text (Bernstein et al. 2010); Chorus recruited a group of workers to hold sophisticated conversations (Lasecki et al. 2013); and Legion allowed a crowd of workers to interact with an UI-control task (Lasecki et al. 2011). A primary challenge for such interactive systems is to decrease latency because crowdsourcing can be slow. At a high level, there are at least three sources of latency for such systems: *(i)* time required to get workers to show up to the task, *(ii)* time required for the workers to do the work, and *(iii)* time for the system to integrate the work that they did into the output given to users. While *(ii)* and *(iii)* are domain-specific, all crowd-powered systems share the challenge of recruiting workers quickly.

Two main approaches have been used to recruit workers from crowd marketplaces, such as Amazon Mechanical Turk (MTurk), quickly: *(i)* **on-demand recruiting**, in which workers are recruited when they are needed (task starts) (Bigham et al. 2010), usually by simply posting HITs (Human Intelligence Tasks) on MTurk marketplace, and *(ii)* **retainers**, which workers are recruited into a retainer pool and can be called up on quickly (Bernstein et al. 2011). However, recruiting by posting HITs is inexpensive but slower; whereas, a full-time retainer is expensive but faster. On-demand recruiting is known to be robust, as it has been used in deployed systems such as VizWiz (Bigham et al. 2010), however, its recruiting time is reportedly longer than a minute, which may be too long for many interactive applications. In the VizWiz system (Bigham et al. 2010), Bigham *et al.* use pre-recruiting to reduce the experienced response time of users – workers are recruited when users begin using the application, which gives the system a lead time of approximately one minute. However, pre-recruiting is not always possible. For applications such as crowd-powered conversational agents (Huang et al. 2016b), the time between users opening the application and sending their first message is very short, which makes pre-recruting less effective. On the other hand, the retainer model, which holds workers in a waiting pool and calls workers back when tasks come in, has a fast response time (less than 10 seconds). However, the retainer model can be expensive for real-world deployments, especially for small or medium size of deployment,

in which most of the money is used for waiting time.

To tackle this cost-latency tradeoff, we proposed *Ignition*, which makes low-latency crowd-powered systems feasible over long deployments by balancing two recruiting methods: *(i)* simply posting HITs, and *(ii)* maintaining a worker waiting pool (*i.e.,* a retainer.) As shown in Figure 1, Ignition starts recruiting workers when a task begins, and the trick is that the model recruits *slightly more* workers than each task needs. The workers who arrive after a task is fully occupied are retained in a waiting page, and then be directed to the next available task.

Ignition is particularly useful for supporting the following types of tasks:

- Applications with an expected response time between **30 seconds to 2 minutes.**
- Tasks with **dynamic length.**
- Tasks that are better with **multiple workers,** but can start when the first worker arrives.
- Systems that are deployed at small or medium scale.

In this paper, we report on our experience developing Ignition and our experience with a 10-month deployment with 648 workers collaboratively completing 745 tasks. We deployed Ignition in support of our on-going deployment of a crowd-powered conversational assistant, Chorus (Huang et al. 2016b). Each Chorus task can start with 1 worker, and could be collectively operated by a maximum of 5 workers. We report on its response time (both of workers arriving to Ignition, and to the supported task), its stability over time, and worker response rates. Our experience may inform future efforts to deploy low-latency crowd-powered systems and develop the underlying infrastructure supporting them.

Related Work

Human computation (von Ahn 2005) has been shown to be useful in many areas, such as story generation (Huang et al. 2016a), writing and editing (Bernstein et al. 2010), image description and interpretation (von Ahn and Dabbish 2004), and protein folding (Cooper et al. 2010). Existing abstractions obtain quality work by introducing redundancy into tasks to verify results (Little et al. 2010; Kittur, Smus, and Kraut 2011). Early crowdsourcing systems leveraged human intelligence through batches of tasks that were completed over hours or days. VizWiz (Bigham et al. 2010) was one of the first systems to elicit nearly real time response from the crowd. It introduced a queuing model to recruit workers on-demand with a latency of around two minutes. Bernstein et al. (Bernstein et al. 2011) showed that the latency to direct a worker to a task can be reduced to below a couple of seconds by combining the concepts of queuing and waiting to recruit crowds (groups) from existing sources of crowd workers. Further work has used queuing theory to show that this latency can be reduced to under a second and has also established reliability bounds on using the crowd in this manner (Bernstein et al. 2012).

Why Using Low-Latency Crowdsourcing

It is generally more expensive and more challenging to utilize crowdsourcing than to use automated approaches when the tolerated response time is extremely short. The common practice of human-in-the-loop architectures is to use the crowd for annotating data or providing human feedback to algorithms, which usually do not require instant turnaround. While some prior works aimed at improving the general speed of crowdsourcing, such as optimizing crowd component's response time (Yan, Kumar, and Ganesan 2010), understanding the sources of the crowdsourcing latency and tackling them (Haas et al. 2015), or inventing new mechanisms for humans to label data quickly (Krishna et al. 2016), the uses of low-latency crowdsourcing often serve specialized purposes. In this section we described three of these specialized purposes: synchronizing with user, other workers, or automated systems. It is noteworthy that these usages are not mutually exclusive. Complex crowd-powered systems such as Evorus (Huang et al. 2017) could have crowd workers to interact with all of these at the same time.

Synchronizing with Users The exploration of low-latency crowdsourcing started with focus of providing fast responses directly to end-users. VizWiz (Bigham et al. 2010) utilized crowd workers to answer visual questions quickly for blind people; Scribe (Lasecki et al. 2012) had non-experts to caption speech for deaf and hard of hearing users with a per-word latency of under 3 seconds; and Adrenaline used the crowd to pick "the best moment" from a short video in a second after the film was shoot (Bernstein et al. 2011). A variance of this type of system is the real-time "polling" system, in which workers or target audiences are asked to provide their opinions about a given question or item (*e.g.,* poster design) instantly. For instance, the APP "1Q"[1] uses smartphones' push notifications to ask poll questions and collect responses from target audiences within few minutes.

Enabling instant responses to end-users also opened up the era of *interactive* crowd-powered systems, where the end-users can synchronously receive and give feedback with crowd workers during multiple turns of interactions. Soylent had crowd workers edit and proofread text in real-time inside user's text editor (Bernstein et al. 2010); Chorus recruited a group of workers to collectively hold a synchronous conversation with the end-user (Lasecki et al. 2013; Huang et al. 2016b); and IdeaGens enabled expert to provide real-time guidance to crowd workers (ideators) to generate new ideas (Chan, Dang, and Dow 2016).

Synchronizing with Other Workers Prior systems have shown that multiple workers can be recruited for collaboration by having workers wait until a sufficient number of workers have arrived (Chilton 2009). While this approach do not provide low-latency responses for any individual labels nor to the end-user, workers are often expected to respond quickly to *other workers.* For example, the ESP Game paired workers synchronously to allow them to play an interactive

[1] 1Q: https://1q.com/

image-label guessing game (von Ahn and Dabbish 2004), and Revolt coordinated crowd workers to collaboratively identify ambiguous items in the data (Chang, Amershi, and Kamar 2017). Similar mechanism have also been adopted by researchers who recruited groups of Amazon Mechanical Turk workers for studying collaborative learning (Wang, Wen, and Rose 2017) and intelligent agents' behaviors inside human groups (Azaria, Richardson, and Kraus 2015). Some crowd-powered systems such as Chorus also allows workers to communicate with each other in real-time, even though it is not an essential component of their workflows.

Synchronizing with Fast-paced Automated Systems Lasecki et al. (Lasecki et al. 2011) introduced continuous real-time crowdsourcing in Legion, a system that allowed a crowd of workers to interact with an UI-control task, such as driving a toy robot, with a latency typically under 1 second. Following this paradigm, real-time crowd has been used to control various automated systems that requires a short response time. CrowdDrone used real-time crowd to orientate an unmanned aerial vehicles in an unknown environment (Salisbury, Stein, and Ramchurn 2015b), and CrowdAR had crowd to identify and track targets in a live video feed (Salisbury, Stein, and Ramchurn 2015a). One variance of this type of systems is the crowd-powered "surveillance" system, in which the system does not need in-the-moment guidance all the time, but instead requires fast responses when the target incidents occur. For example, Zensors utilized continuous real-time crowdsourcing to monitor live surveillance video feed and notified end-users when the target event (*e.g.,* it starts raining) occurs (Laput et al. 2015).

Recently, more systems also started exploring the fast-paced collaboration between crowd workers and automated components. Evorus recruited a group of crowd workers to collaborate with automated bots to hold a conversation (Huang et al. 2017); Guardian first had crowd workers to extract key information from a running dialog (Huang, Chen, and Bigham 2017), and then used these information to query Web APIs, and finally had workers to convert the API responses back into natural languages (Huang, Lasecki, and Bigham 2015); and CRQA system used a human-in-the-loop architecture to answer questions within 60 seconds (Savenkov and Agichtein 2016).

Ignition Framework

In this section we describe our implementation of Ignition. This implementation has been deployed for supporting an on-demand crowd-powered conversational agent, Chorus, in-the-wild since June 2016. We iteratively improved the design of the system through multiple empirical evaluations and feedback from workers and our collaborators.[2]

Worker's Workflow

From the workers' perspective, Ignition is composed of a sequence of web pages. As shown in Figure 2, the workflow of workers are as follows: First, a worker reaches to the landing page. The landing page uses five slides to briefly introduce the task, and also have the worker sign the consent form when necessary. Each new worker who has never submitted our HIT before is also required to finish the one-minute interactive tutorial. Second, the worker then clicks a button to enter to the waiting page. The interface of the waiting page is shown in Figure 3. The worker is instructed to keep the browser tab open to wait for the task. The system grants the worker with an retainer reward (2 points) per second for his/her waiting time. Reward points are later converted to bonus pay for workers. The accumulated reward points are displayed in the middle of the page, with the remaining waiting time and estimated bonus amount listed below.

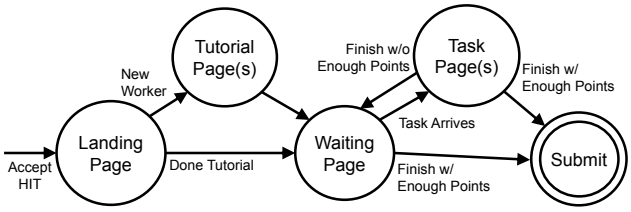

Figure 2: Transition graph for workers in Ignition. A worker first reaches the landing page for the introduction and tutorial, and then goes to a waiting page (also known as retainer page) to wait for tasks. When a task arrives for the worker, the worker is called back to perform the task using a pop-up alert and a sound notification.

When a task arrives, the waiting page uses a pop-up alert and a bird sound notification to call the worker back, and the worker is required to respond within 20 seconds. If the worker responds in time, he/she will be then directed to the task page to perform the task; If the worker reaches to 4000 points (estimatedly 33 minutes) without any tasks, the waiting page will also call the worker back to confirm that he/she is still available, and then automatically submits the HIT if the worker responds within 20 seconds. The workers who wait in the retainer pool promise to respond within a specific amount of time. We recognize these promises and the time spent by the workers as valuable contributions to keep a deployed crowd-powered system stable. Therefore, we believe that a requester should pay for workers' waiting time regardless of whether they eventually are assigned with a task or not. On the other hand, if a worker does not respond in time, the interface will be wiped out and become unable to submit. An instruction will further be displayed to ask the worker to return this HIT as a penalty.

Furthermore, as shown in Figure 2, if the task utilizes a comparable reward point system as that of Ignition, the worker can be sent back to the waiting page to continue accumulating points if he/she did not make sufficient contribution to the task (R_{task}.) This is particularly useful for the tasks that have dynamic length such as conversation tasks (Huang et al. 2016b). If a worker enters a conversation task right before it ends and thus earns insufficient reward, Ignition allows the worker to go back to retainer with his/her accumulated reward points to continue waiting. This design

[2]Our implementation of Ignition is available at: https://github.com/windx0303/ignition-model

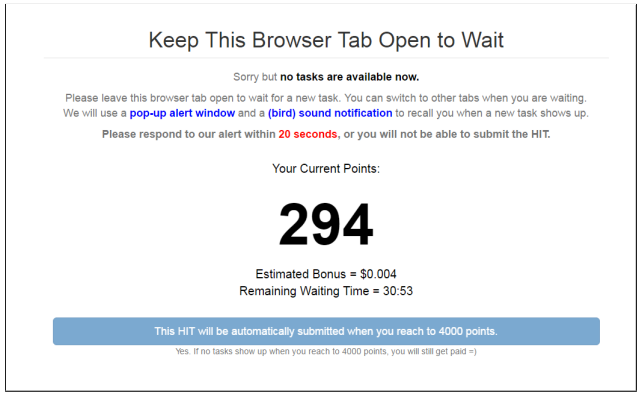

Figure 3: The worker interface. Workers are instructed to keep the browser tab open to wait for tasks. Their earned reward points are displayed in the middle of the page, with the remaining time and estimated bonus amount listed below.

also allows workers to focus only on the amount of reward points they have earned, instead of keep track of both waiting time and reward points at the same time.

Recruiting Strategy & Worker Routing

When a task arrives, Ignition recruits slightly more workers than needed, and uses a retainer to hold the extra workers for future tasks. In our deployed system, the task can start with 1 worker, but is better with 4 to 6 workers. Therefore, for each incoming task, we aim to recruit 8 workers in total. The recruiting process can be break down into two parts: Ignition first recalls $C_{retainer}$ workers from retainer, and then aims to recruit C_{market} workers from Amazon Mechanical Turk marketplace. Namely, $C_{retainer}$ and C_{market} sums up to 8, as Equation (1) shows.

$$C_{retainer} + C_{market} = 8 \quad (1)$$

The underlying assumption of Ignition is that a larger C_{market} has a faster recruiting time from the market, and thus compensates the speed decrease when no (or few) workers are waiting in the retainer ($C_{retainer} = 0$.) The detailed process is as follows.

1. Upon the arrival of a new task, Ignition checks if any workers are currently waiting in the retainer If yes, the system greedily calls at most 6 workers back from retainer to do the task ($0 \leqslant C_{retainer} \leqslant 6$).
2. Ignition then tries to recruit C_{market} workers from mturk marketplace by posting 1 HIT with an average of C_{market} assignments. We introduced randomness into the system that Ignition has a 30% chance to add one assignment ($C_{market} + 1$), 30% chance to subtract one assignment ($C_{market} - 1$), and 10% to post 10 assignments. Randomness was added here to allow us to collect data about different numbers of assignments, in case worker response rate or time is dependent on the number of workers already recruited (which may be true if we are pulling from an especially small pool), and to explore latency effects potentially introduced by the platform itself (anecdotally, HITs with different numbers of assignments seem to appear with different latencies on MTurk).

When a task has 5 or more workers (either from retainer or marketplace,) the task is labeled as "fully-occupied" and stops taking more workers, and the workers recruited via the same HIT who arrives later will start waiting in the retainer. However, if some workers left before the task ends and thus the task has less than 5 workers, the task will be open to workers again.

Instant, Retained, and No Tasks

When a worker reaches to the waiting page and starts waiting, one of the following events will occur.

- **[Instant Task]** The task has been created and remains open when the worker arrives to the waiting page. In this case the worker does not need to wait and will be called immediately when reaching to the waiting page.
- **[Retained Task]** When the worker arrives to the waiting page, the original task is fully-occupied or over. The worker opens the browser tab to wait, and then the next task arrives.
- **[No Task]** Similar to the Retained Task, worker arrives to the waiting page and starts waiting. However, no tasks appear till the end of his/her waiting time. The workers can submit the HIT at the end and just get paid with mturk price and waiting bonus.

It is noteworthy that workers are not allowed to "double waiting." If a worker is currently waiting in the retainer and reaches to the waiting page again via a different HIT, the system will block him/her on the second browser tab.

Long-term Deployment Study

The current version of Ignition was initially launched in June, 2016 for supporting the on-demand crowd-powered conversational agent, Chorus[3], that was deployed to public (Huang et al. 2016b). To date[4], 122 users used the conversational agent during 745 conversational sessions. Each session was one task, which lasted an average of 10.87 minutes (SD = 15.26.) The assignment and task distribution is shown in Figure 5. Totally 6,823 assignments by 648 workers were recorded, in which 45.04% were of Instant tasks, 8.75% were of Retained tasks, and 46.21% were of no tasks. As for the task distribution, 50.07% of HITs were posted when no workers were in the retainer, 18.12% of HITs were posted when 1 worker was in the retainer, and 11.54% of HITs were posted when 2 workers were in the retainer. This distribution suggests that in Ignition, the speed of recruiting from marketplace and of recalling workers from the retainer is equally important. We will analyze the performance of these two parts in the follosing subsections.

[3]Chorus: http://talkingtothecrowd.org/

[4]All results presented in this paper are based on data recorded from July 1st, 2016 to April 27th, 2017.

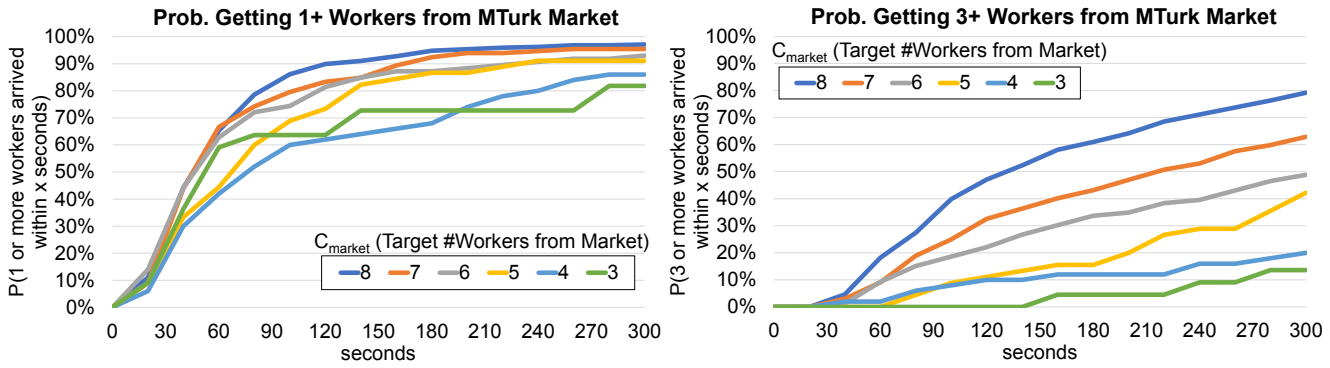

Figure 4: The probability of a posted HIT that have at least 1 (and at least 3) workers in the retainer at x seconds after the HIT was posted. When Ignition posted a HIT with an average of 5 assignments, 40% of the time the first worker will reach to the retainer under 2 minutes. ($N = 346, 132, 86, 45, 50, 22$)

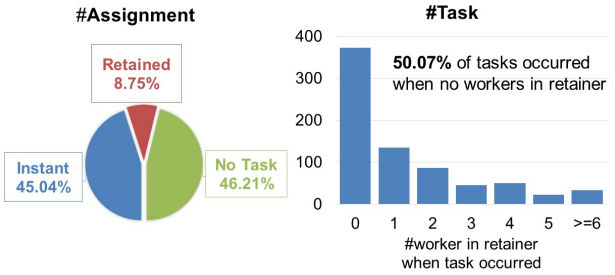

Figure 5: The distribution of assignments and tasks. 50.07% of HITs were posted when no workers were in the retainer. Amongst totally 6,823 assignments, 45.04% of assignments were of Instant tasks, 8.75% were of Retained tasks, and 46.21% were of no tasks.

Recruiting from the Marketplace

In the deployed Ignition, majority of assignments were of the [Instant Task] case, in which workers do not need to wait and are directed to tasks immediately after they reach to the retainer. In the case of [Instant Task], most of the recruiting time are spent in waiting for workers to find the HIT, accept the HIT, start doing the HIT, and finish the tutorial. To the best of our knowledge, no prior works reported how fast a HIT will be taken on Amazon Mechanical Turk marketplace. Therefore, we calculated the probability of a posted HIT that have at least 1 (and at least 3) workers in the retainer at x seconds after the HIT was posted. The results are shown in Figure 4. The x axis is the cut-off time (x seconds,) and the y axis is the probability at x seconds.

The results suggest that when a HIT posted to mturk marketplace with more assignments, it is more likely workers will arrive to the task earlier. When Ignition posted a HIT with an average of 5 assignments, 40% of the time the first worker will reach to the retainer under 2 minutes; when Ignition posted a HIT with an average of 8 assignments, 79% of the time the first worker will reach to the retainer under 80 seconds. With a larger C_{market}, Ignition's recruiting strategy can get workers faster. This results confirm the

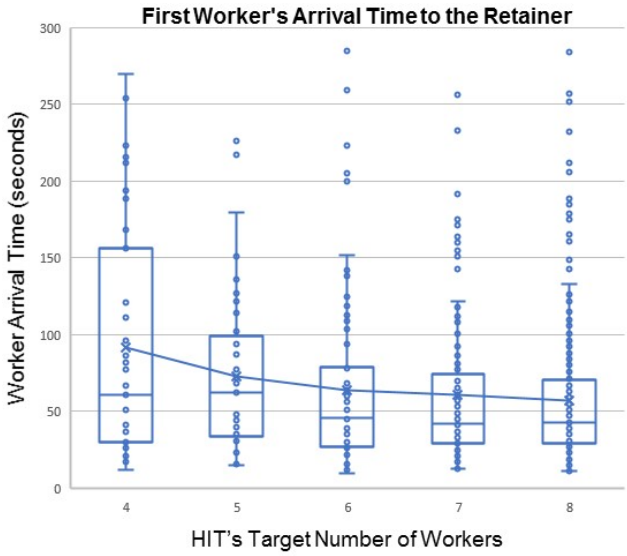

Figure 6: The distribution of the first workers' arrival times when the system aimed to recruit various number of workers from mturk marketplace. A larger C_{market} results in not only a smaller mean of arrival time, but also a smaller standard deviation.

underlying assumption of Ignition that a larger C_{market} results in a faster recruiting time from market, which can compensate the speed decrease when no workers waiting in the retainer. Furthermore, the distribution of the first workers' arrival times when the system aimed to recruit various number of workers from mturk marketplace is shown in Figure 6 (workers who arrived after 5 minutes are excluded.) A larger C_{market} results in not only a smaller mean of arrival time, but also a smaller standard deviation.

The positive correlation between C_{market} and recruiting speed could be caused by several factors: First, a HIT with more assignments has a longer lifetime on Mturk marketplace before all assignments are taken, and thus has better visibility. Second, a HIT with more assignments is more ro-

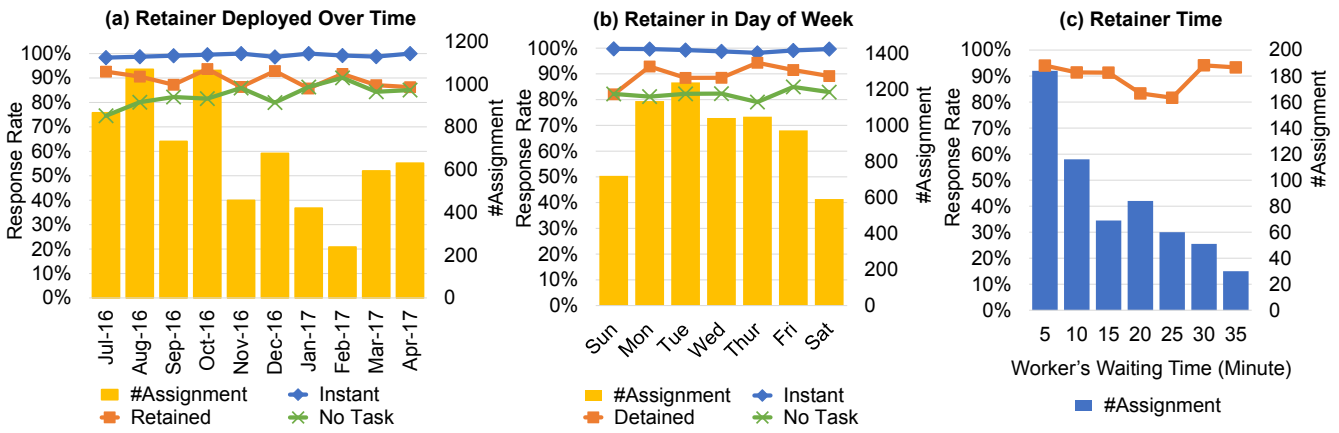

Figure 7: Worker recall rate from the retainer *(a)* by months, *(b)* by the day of the week, and *(c)* by time spent waiting. Overall, results demonstrate that the recall rate was reasonably stable during our deployment, although this rate did vary.

bust to the workers who hold the accepted tasks for a while instead of doing the task immediately. Finally, given that we had a relatively small group of active workers who took most of our HITs, a smaller $C_{retainer}$ could indicate that more of these active workers are still on the marketplace, and are thus easier to be recruited from the market. Workers use web browser extensions to alert them when new HITs are posted by favored requesters, and therefore some of the results we have seen could be influenced by the use of these tools.

Recruiting from the Retainer

The retainer model has shown to be able to recall 75% of workers within a couple of seconds (Bernstein et al. 2011). Our deployment study echoes this findings, and further demonstrates the long-term dynamics of a deployed retainer. We calculated the response rate (*i.e.*, the probability that a worker responded to the task within 20 seconds) of [Instant Task], [Retained Task], and [No Task][5] during each month of our deployment. As Figure 7(a) shows, the response rate of [Instant Task] cases were nearly perfect. We believe that it is because workers do not need to wait. Except for the first two months, the response rates of [Retained Task] and [No Task] were all higher than 80% during the entire deployment. Interestingly, we found that the response rate of [Retained Task] cases slightly dropped on Sunday (Figure 7(b).) Within the [Retained Task] cases, we also calculated the response rate with respect to each worker's waiting time in the retainer (Figure 7(c).) We found that workers have the lowest response rate when waited in the retainer between 15 to 25 minutes. It is possible that workers learned to pay more attention at the end of their waiting time (*i.e.*, 33 minutes), perhaps because they were aware of their chances of getting paid without actually doing the task.

It is noteworthy that our implementation has a more relaxed response time constraint (20 seconds) for workers than that of Bernstein *et. al*'s work ($\leqslant$ 5 seconds.) Therefore the response rate reported in this section is higher. During our deployment, the average response time from the retainer is 7.703 seconds (SD = 4.679, all cases included.)

In sum, during our deployment, the retainer was shown to be able to stably recall 80% to 90% workers when tasks comes in, which suggests its mechanism is reliable to real-world use (although potentially expensive).

Recruiting by Ignition (Retainer & Marketplace)

Finally, for understanding the end-to-end performance of Ignition, we analyzed the probability of a started task that have at least 1 (and at least 3) workers reaching to the task (*not* retainer), regardless of the sources of workers (either from marketplace or from retainer.) The result is shown in Figure 8. The x-axis is the cut-off time (x seconds) after the HIT was posted, and the y-axis is the probability at x seconds. Figure 4 and Figure 8 demonstrates how our hybrid approach works inside a real-world deployed system. It is noteworthy that the lines of "$C_{market} = 8$" in Figure 4 (navy blue color) and the lines of "$C_{retainer} = 0$" in Figure 8 (navy blue color) are nearly the same (because $C_{market} + C_{retainer} = 8$.) When no workers are waiting in the retainer pool, Ignition posts more assignments to mturk marketplace to recruit workers to have a better recruiting speed (Figure 6); when some workers are waiting in the retainer pool, Ignition recalls workers back from retainer and thus results in a much shorter response time (Figure 8.)

As shown in Figure 5, half of tasks occurred when no workers were waiting in the retainer pool. Ignition dynamically decides the number of assignments to post based on the number of workers in retainer, and thus helps to balance monetary cost and response speed.

Worker Survey

We believe that understanding workers' opinion and behaviors could help us further improve the design of future low-latency crowd-powered systems. While low-latency crowdsourcing has been proposed and developed for many years, literatures had little to say about workers' perspective of these technologies, nor how they work with these tasks.

[5]For [No Task], workers also need to respond at the end of their waiting time to confirm that they are still available.

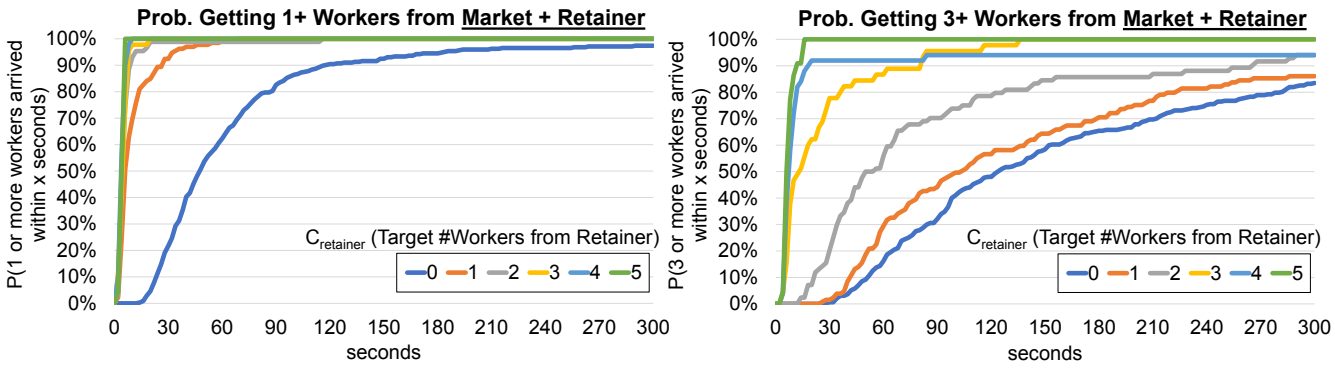

Figure 8: The probability that a task will have at least 1 (or 3) new worker(s) reach to the task over time, regardless of the sources of workers, given the number of workers that need to be recruited.

Therefore, we designed a questionnaire to collect opinions and self-reported behavior for 156 workers who have completed at least 10 HITs in the Ignition system. We posted the survey as a $2.0 HIT on Amazon Mechnical Turk for two weeks, and contacted these workers to participate in the survey. A total of 101 workers finished the survey, *i.e.*, the response rates is 64.74%. Each worker on average took 10 minutes 20 seconds to finish to the questionnaire.

Workers' Opinion about Retainer HITs

In the survey, we asked workers to rate if they like *(i)* "doing HITs on MTurk in general" and *(ii)* "doing retainer HITs,"[6] which we referred to as *likability* score. Responses were collected on a five-point Likert scale (strongly disagree = 1, and strongly agree = 5). As a result, the average likability rating of general HITs is significantly higher than that of the retainer HITs ($p = 0.0058$). Workers reported an average of 4.47 points (SD = 0.81) for general HITs, and 4.12 points (SD = 1.00) for retainer HITs. The distributions of rating points are shown in Figure 9.

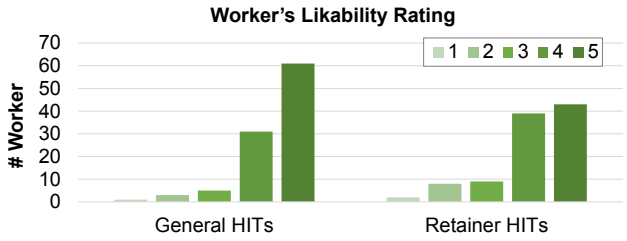

Figure 9: Answers to Likert scale questions on our survey indicating that workers like doing *(i)* HITs on MTurk in general and *(ii)* retainer HITs. The average likability rating of general HITs is significantly higher ($p = 0.0058$).

[6]In the questionnaire we defined a "retainer HIT" as follows: *Unlike a typical HIT that rewards a worker for completing a task, a HIT served by a retainer system pays a worker for "waiting for a task to appear" in addition to completing the actual task.*

What Workers Disliked Based on our experience of deploying Ignition, we expected that workers would prefer regular HITs over retainer HITs. In the survey, we further asked workers to rate the extent to which they *disliked* the four main aspects that workers have complained about: *(i)* needing to wait until work is available, *(ii)* committing their next 30 minutes, *(iii)* responding to the recall alert with 20 seconds, and *(iv)* believing the payment for waiting time is too low. Workers reported feeling most disliking the payment rate for their waiting time, for which the average dislikability rating is 3.64 (SD = 1.22). The next factor was the requirement to respond quickly, in which the average rating was 3.09 (SD = 1.41). The third is the requirement that workers need to commit 30 minutes, for which the average rating was 2.76 (SD = 1.31). The least dislikable factor was needing to wait, in which the average rating was 2.52 (SD = 1.24). The rating distributions are shown in Figure 10. We also asked workers to rate on a five-point Likert scale how *easy* it is for them to commit 30 minutes and to respond within 20 seconds. Workers on average find it slightly easier to commit their time (3.75, SD = 1.13) than to respond to the recall alert quickly (3.70, SD = 1.24).

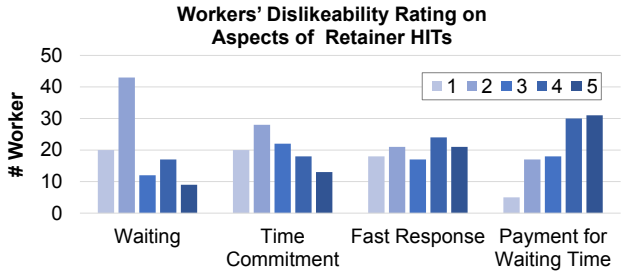

Figure 10: Answers to Likert scale questions on our survey indicating that workers dislike *(i)* waiting, *(ii)* committing time, *(iii)* responding to the recall alert fast, and *(iv)* getting paid lower for their waiting time of retianer HITs. As a result, workers feel most unsatisfied about the lower payment for their waiting time, and least concern about the fact that they need to wait.

Furthermore, we asked workers to provide some other dislikable factors which were not covered in these four items. 5 workers mentioned that the retainer HITs are likely "buggier" than regular HITs. One worker said that while our HITs were fine, "I've seen the issue with other retainer hits." Another worker also said "Maybe it didn't always operate smoothly." Meanwhile, most workers echoed their thoughts on these four aspects, especially about the lower wage for their waiting time, instead of mentioning new dislikable factors. Workers suggested the ideal wage they expected for waiting. One worker said, "it's hardly \$0.20 + 0.05 to wait. It should be at least \$0.40 +0.10"; a worker mentioned another requester provided \$0.50 for 10-minute waiting time.

What Workers Liked We asked workers to answer in free text what they *like* about the retainer HITs. The following three themes emerged in the collected responses: *(i)* getting paid for simply waiting, *(ii)* the waiting page have a large clear timer to show the amount of accumulated bonus and remaining time, and *(iii)* being able to work on other HITs or do other things in parallel.

34.7% of workers mentioned they like to get paid for just waiting. As one worker said, "...(Your HITs) give a sense of 'not waiting in the dark' since they pay you to wait (not even having focus to your tab). This is a form of respect to the turkers and predisposes me to do my best to complete your projects." Workers also pointed out that some other requesters did not pay for their waiting time if they did not encounter any tasks. For example, one worker said, "I liked that even if there was no work i was still getting paid something just to wait, most other requesters don't give that courtesy." 29.7% of workers mentioned that they like the design of our waiting page, especially it has a big timer showing the bonus and remaining time. As one worker put it, "...the way the system shows points increasing with the time and the minimum amount of time one needs to wait to submit the hit is useful information and helps keep ones attention to the task." Another workers said, "I like that the countdown and the bonus totals change in real time and I can see how much time is left." One other worker also said "I think it is really great that you have a waiting page, I wish more requesters did." 25.7% of workers mentioned that they like the fact that they are free to work on other HITs or do other things during waiting. For instance, one worker commented, "We can work or do other things as well if there is no task assigned, so we can utilize our time effectively because of the recall alert and waiting page feature."

How Do Workers Work with Retainer HITs

We asked workers the browsers[7] and tools (*e.g.,* browser extensions) they used to keep track of our HITs. 33.7% of worker said they used browser extensions to subscribe to our HITs on Amazon Mechanical Turk. We then asked these workers which tools or extensions they have been using. The following are all the browser extensions mentioned, along with the number of workers mentioned it: HIT Scraper (12), Turkmaster (6), JR Mturk Panda Crazy (5), Mturk Suite (3), Hit Finder (2), HIT Notifier (2), Openturk (1), Overwatch for worker.mturk (1), HIT Monitor (1), "Greasemonkey scripts" (1), and "a auto reload tool" (1).

We also asked workers "Is there any forum or community you use to keep track of our HITs?" 28.7% of workers said yes and reported the forum they use. The followings are the on-line communities worker reported using, along with the number of workers who mentioned using it: MturkCrowd.com (10), "worker forums" (4), TurkerNation (4), "Hits Worth Turking For" Reddit group (3), TurkerHub (3), TurkOpticon (3), mturkgrin (1), Turkalert (1), and "a whatsapp group" (1).

Furthermore, we asked workers "What do you usually do when waiting on the countdown timer page?". 79.2% of workers usually do other HITs in parallel; 15.8% of workers usually do something else on their computers in parallel instead of doing other HITs; and 2% of workers do not use computer in parallel, but do something else (*e.g.,* watching TV) instead. While the majority of workers take other HITs when they are waiting, around 20% of workers do something else instead, even not in front of their computers. Allowing them to choose a louder or more aggressive notification is likely helpful to recall them back.

We also asked workers "If something unexpected happen during your waiting time and you have to leave, what do you usually do?". 46.5% of workers said they usually leave the browser open, just in case he/she could be back soon; 44.6% of workers usually return the HIT and leave; and 5.9% of workers just close the browser directly. Since many workers are willing to come back to continue waiting after they were interrupted, enabling them to pause and resume on the waiting page could be helpful. However, since the task distribution over retainer time is not uniformly distributed (Figure 7(c)), a more sophisticated mechanism might be needed for preventing workers from abusing a pausing feature (*e.g.*, disallowing workers to pause during the first 5 minutes, or setting a maximum pause time.)

Discussion

Delay-Cost Trade-offs As expected, there are trade-offs between response speed and cost when recruiting workers on MTurk. Maintaining a retainer pool can result in a very short response time, however, is also expensive. Given our current rate, which is \$0.25 per 33 minutes, a base rate of running a full-time retainer can be calculated as follows. If we maintain a 5-worker retainer for 24 hours, it would cost \$65.45 per day (including MTurk's 20% fee), \$458.18 per week, or approximately \$2,000 per month. This price does not consider the possible refillings of retainer when the system requires more than 5 workers at the same time. Ignition' cost is basically a function of task numbers, getting rid of the basic rate that needed to maintain a retainer pool.

It may be possible to learn optimal policies for recruiting based on a budget. Our results suggest that variables such as the number of workers waiting, the number of workers already recruited, the time workers have already spent in the retainer, each worker's prior response rate/time, etc. could

[7] 87.2% of workers used Google Chrome, and 18.8% of workers used Mozilla Firefox.

be inputs to such a model. Furthermore, given workers generally expect a higher wage for their waiting time, the hourly wage of the retainer and task also likely play roles. For systems that will be deployed over long periods, it may be useful to model the observed latency recruiting workers from the marketplace – some applications may not even need to use a retainer, relying instead on the natural latency afforded by the marketplace itself.

Task-Dependent Factors It is noteworthy that in this paper we only measure workers' *arrivals* to the waiting page and tasks, but not their completions nor performances on tasks. In other words, even if Ignition is able to recruit workers quickly with reasonable financial cost, workers can still return the HIT or not complete the HIT until it expires, even when they reach the HIT quickly. In the early stage of deploying Chorus (Huang et al. 2016b), many workers return our HITs not because of the Ignition recruiting system, but the unfamiliarity of Chorus tasks.

Conclusion & Future Work

We have introduced *Ignition*, an approach that combines both on-demand recruiting and the retainer model to bring workers to tasks from Amazon Mechanical Turk. We have explored deployment of Ignition over 10 months to support a medium-sized crowd-powered system deployment, finding that it reasonably balanced the cost and latency of recruiting workers. The paper discusses the observed stability, timing, and retention of workers using the model, demonstrating the feasability of on-demand recruitment over time. In the future, it would be interesting to explore how our findings are affected by increased load on the crowd marketplace, and how these results might change if we instead consider a large, active deployment in which many more workers are involved. Furthermore, prior workers have explored using old tasks (Bigham et al. 2010) or micro diversion breaks (Dai et al. 2015) to engage workers longer, which could be incorporated in future Ignition frameworks.

Future research may also consider the effect of continuity on worker response time and recruitment, as prior work has found that this can affect quality (Lasecki et al. 2015). For instance, it may be that workers are more easily brought back to work on another task after they have finished a prior one. It is also likely that the variables we measure change over time, as workers learn of the task and adapt to it, or as the underlying market changes. It would thus be interesting to compare the evolution of more than one system, on more than one platform, over a long period of time.

Acknowledgements

This research was supported by the Yahoo! InMind Project (Azaria and Hong 2016). We thank Joseph Chee Chang, who came up with the name "Ignition," Kotaro Hara, Saiph Savage, and Daniel S. Weld for their feedback and assistance. We also thank the workers on Mechanical Turk who participated in our studies and survey.

References

Azaria, A., and Hong, J. 2016. Recommender system with personality. In *Proceedings of the 10th ACM conference on Recommender systems*. ACM.

Azaria, A.; Richardson, A.; and Kraus, S. 2015. An agent for deception detection in discussion based environments. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 218–227. ACM.

Bernstein, M. S.; Little, G.; Miller, R. C.; Hartmann, B.; Ackerman, M. S.; Karger, D. R.; Crowell, D.; and Panovich, K. 2010. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, 313–322. New York, NY, USA: ACM.

Bernstein, M. S.; Brandt, J.; Miller, R. C.; and Karger, D. R. 2011. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, 33–42. New York, NY, USA: ACM.

Bernstein, M. S.; Karger, D. R.; Miller, R. C.; and Brandt, J. R. 2012. Analytic methods for optimizing realtime crowdsourcing. In *Proceedings of Collective Intelligence*, CI 2012, to appear.

Bigham, J. P.; Jayant, C.; Ji, H.; Little, G.; Miller, A.; Miller, R. C.; Miller, R.; Tatarowicz, A.; White, B.; White, S.; and Yeh, T. 2010. Vizwiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, 333–342. New York, NY, USA: ACM.

Chan, J.; Dang, S.; and Dow, S. P. 2016. Improving crowd innovation with expert facilitation. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 1223–1235. ACM.

Chang, J. C.; Amershi, S.; and Kamar, E. 2017. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2334–2346. ACM.

Chilton, L. 2009. Seaweed: A web application for designing economic games. Master's thesis, MIT.

Cooper, S.; Khatib, F.; Treuille, A.; Barbero, J.; Lee, J.; Beenen, M.; Leaver-Fay, A.; Baker, D.; Popovic, Z.; and Players, F. 2010. Predicting protein structures with a multiplayer online game. *Nature* 466(7307):756–760.

Dai, P.; Rzeszotarski, J. M.; Paritosh, P.; and Chi, E. H. 2015. And now for something completely different: Improving crowdsourcing workflows with micro-diversions. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 628–638. ACM.

Haas, D.; Wang, J.; Wu, E.; and Franklin, M. J. 2015. Clamshell: Speeding up crowds for low-latency data labeling. *Proc. VLDB Endow.* 9(4):372–383.

Huang, T.-H. K.; Ferraro, F.; Mostafazadeh, N.; Misra, I.; Agrawal, A.; Devlin, J.; Girshick, R.; He, X.; Kohli, P.; Batra, D.; et al. 2016a. Visual storytelling. In *Proc. the 15th*

Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2016). NAACL.

Huang, T.-H. K.; Lasecki, W. S.; Azaria, A.; and Bigham, J. P. 2016b. "is there anything else i can help you with?": Challenges in deploying an on-demand crowd-powered conversational agent. In *Proceedings of AAAI Conference on Human Computation and Crowdsourcing 2016 (HCOMP 2016)*. AAAI.

Huang, T.-H. K.; Chang, J. C.; Swaminathan, S.; and Bigham, J. 2017. Evorus: A crowd-powered conversational assistant that automates itself over time. In *Poster track of the 20th ACM Symposium on User Interface Software and Technology (UIST Poster 2017)*.

Huang, T.-H. K.; Chen, Y.-N.; and Bigham, J. P. 2017. Real-time on-demand crowd-powered entity extraction. In *Proceedings of the 5th Edition Of The Collective Intelligence Conference (CI 2017)*. Oral presentation.

Huang, T. K.; Lasecki, W. S.; and Bigham, J. P. 2015. Guardian: A crowd-powered spoken dialog system for web apis. In Gerber, E., and Ipeirotis, P., eds., *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2015, November 8-11, 2015, San Diego, California.*, 62–71. AAAI Press.

Kittur, A.; Smus, B.; and Kraut, R. 2011. Crowdforge: Crowdsourcing complex work. Technical Report CMUHCII-11-100, Carnegie Mellon University.

Krishna, R. A.; Hata, K.; Chen, S.; Kravitz, J.; Shamma, D. A.; Fei-Fei, L.; and Bernstein, M. S. 2016. Embracing error to enable rapid crowdsourcing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, 3167–3179. New York, NY, USA: ACM.

Laput, G.; Lasecki, W. S.; Wiese, J.; Xiao, R.; Bigham, J. P.; and Harrison, C. 2015. Zensors: Adaptive, rapidly deployable, human-intelligent sensor feeds. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '15. New York, NY, USA: ACM.

Lasecki, W. S.; Murray, K. I.; White, S.; Miller, R. C.; and Bigham, J. P. 2011. Real-time crowd control of existing interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 23–32. ACM.

Lasecki, W. S.; Miller, C. D.; Sadilek, A.; Abumoussa, A.; Borrello, D.; Kushalnagar, R.; and Bigham, J. P. 2012. Real-time captioning by groups of non-experts. In *In Proceedings of the Symposium on User Interface Software and Technology (UIST 2012)*, 23–34.

Lasecki, W. S.; Wesley, R.; Nichols, J.; Kulkarni, A.; Allen, J. F.; and Bigham, J. P. 2013. Chorus: A crowd-powered conversational assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, 151–162. New York, NY, USA: ACM.

Lasecki, W. S.; Rzeszotarski, J. M.; Marcus, A.; and Bigham, J. P. 2015. The effects of sequence and delay on crowd work. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, 1375–1378. New York, NY, USA: ACM.

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2010. Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, 57–66. New York, NY, USA: ACM.

Salisbury, E.; Stein, S.; and Ramchurn, S. 2015a. Crowdar: augmenting live video with a real-time crowd. In *Third AAAI Conference on Human Computation and Crowdsourcing*.

Salisbury, E.; Stein, S.; and Ramchurn, S. 2015b. Real-time opinion aggregation methods for crowd robotics. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 841–849. International Foundation for Autonomous Agents and Multiagent Systems.

Savenkov, D., and Agichtein, E. 2016. Crqa: Crowd-powered real-time automatic question answering system. In *Proc. the Fourth AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2016)*.

von Ahn, L., and Dabbish, L. 2004. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, 319–326. New York, NY, USA: ACM.

von Ahn, L. 2005. *Human Computation*. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA.

Wang, X.; Wen, M.; and Rose, C. 2017. Contrasting explicit and implicit support for transactive exchange in team oriented project based learning. In *Proc. 12th International Conference on Computer Supported Collaborative Learning (CSCL) 2017*. Philadelphia, PA: International Society of the Learning Sciences.

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.