Task Scheduling on Crowdsourcing Platforms for Enabling Completion Time SLAs

Matthias Hirth*, Florian Steurer†, Kathrin Borchert†, Dan Dubiner‡
* *TU Ilemnau*, Ilmenau, Germany, matthias.hirth@tu-ilmenau.de
† *University of Würzburg*, Würzburg, Germany, kathrin.borchert@informatik.uni-wuerzburg.de
‡ *ScaleHub AG*, Norderstedt, Germany, dan.dubiner@scalehub.com

***Abstract*—Today a diverse crowdsourcing economy has established, and similar to other service industries, service level agreements (SLAs) are a crucial part of this ecosystem. However, the uncertainty introduced by human workers makes it harder to estimate and set completion time guarantees than in a system that is solely technology based. In this work, we, therefore, analyze an exemplary task scheduling strategy that enables operators of crowdsourcing-based services to meet completion time SLAs. In particular, we use the real-world crowdsourcing-based text digitalization platform ScaleHub as the foundation for our system model and a dataset provided by ScaleHub to obtain a realistic parametrization of the model. We derive an analytical model that enables us to illustrate the potential of scheduling mechanisms to meet SLA constraints and that helps platform providers to optimize their systems. Further, we use a more detailed simulation model to illustrate the challenges arising from erroneous worker submission in the context of task schedulers.**

***Index Terms*—crowdsourcing, task scheduling, performance evaluation**

I. Introduction

The continuously growing number of smart and Internet-connected devices leads to the generation of an enormous amount of data every day. In conjunction with the newly available computation power and storage capabilities, this data was one of the main drivers for the recent advances in the fields of big data analysis, machine learning, and data mining. However, despite these new possibilities still, a significant number of tasks exist that require human judgments and input, e.g., creating labeled data for training machine learning algorithms. The crowdsourcing paradigm tries to solve this by utilizing web-based platforms to distribute tasks to a global workforce of Internet users.

Based on the crowdsourcing paradigm a diverse industry has established today, ranging from low-level work provider platforms like Amazon Mechanical Turk (MTurk) [1] or Microworkers [2] that enable their users to easily access a global workforce, to full-service platforms like Figure Eight [3] or Streetspotr [4] that offer their customers crowdsourcing-based services while completely abstracting the actual crowdsourcing related challenges. Mainly these service-oriented platforms now start facing challenges like SLAs concerning guaranteed completion times that have been present in the service domain for decades but are relatively new in the context of crowdsourcing.

In this work, we analyze how task scheduling strategies can enable operators of crowdsourcing-based services to meet completion time SLAs and how these strategies affect the costs of the service providers. In particular, we use the real-world crowdsourcing-based text digitalization platform ScaleHub [5] as the foundation for an abstract analytical and a detailed simulation model. The parametrization of both models is based on the analysis of a large-scale dataset provided by ScaleHub.

The remainder of the paper is structured as follows. Section II provides the necessary background on crowdsourcing and the different types of crowdsourcing platforms currently available. Further, we review related works. In Section III, we describe the dataset obtained from ScaleHub and derive realistic input parameters for the analytical model proposed in Section IV. The simulation model described in Section V extends our analytical model to feature a more realistic scenario including batch arrivals of tasks and invalid submissions of crowd-workers. Section VI concludes this paper.

II. Background and Related Work

In the following, we give a brief introduction to the concept of crowdsourcing, and commercial crowdsourcing providers and services in general. Further, we review related work in the field of mathematical and simulative modeling of crowdsourcing platforms.

A. Crowdsourcing

Jeff Howe originally introduced the term crowdsourcing as outsourcing a task and functions to a network of people (the crowd) in the form of an open call [1]. A more recent definition by Estellés-Arolas et al. [2] tries to convey the various aspects of this topic more precisely. Here, the authors define crowdsourcing as the proposal of a task to a group of individuals, which results in mutual benefit for both, the proponent and the individual undertaking this task.

The outsourced task can take a variety of forms. One possible way to categorize tasks is to distinguish between complex, creative task, like creating content such as stock

[1] https://mturk.com Accessed Apr. 2019
[2] https://www.microworkers.com Accessed Apr. 2019
[3] https://www.figure-eight.com Accessed Apr. 2019
[4] https://streetspotr.com Accessed Apr. 2019
[5] https://www.scalehub.com Accessed Apr. 2019

photos, and routine task, like data capture [3]. However, to crowdsource a large task, like digitizing a long document, it is often necessary to decompose the task into smaller units, which are acceptable for the crowd workers. The results of those smaller tasks need to be recombined to obtain the final result. Buettner calls this the Analysis-Synthesis approach [4]. Especially for complex tasks with interdependencies and multiple types of expertise needed, designing reliable workflows using analysis-synthesis is not well understood yet [5].

Crowdsourcing platforms act as mediators, providing requesters with access to an on-demand human workforce for processing tasks. The platforms differ in the services they offer. Platforms like MTurk or Microworkers are mainly concerned with supplying a large and general purpose workforce. Accordingly, they offer functionality to select and filter workers, but due to their general purpose, lack sophisticated quality control mechanisms [6]. Other, specialized, crowdsourcing providers help requesters to decompose tasks and offer predefined workflows for specialized tasks. Meta-platform like Figure Eight provide crowdsourcing services by utilizing other platforms, without necessarily maintaining a crowd on their own.

B. Crowdsourcing Service Providers

Users that want to leverage the crowd need to provide governance, which includes finding a strategy to provide incentive for the workers, breaking tasks down to appropriate subtasks and finding a mechanism to assure the quality of the work. All of these problems are non-trivial and are still subject to ongoing reseach [7].

The complexity of governance opens the possibility for meta-platforms or service providers to help customers to utilize the crowd. Some examples of such service providers are Figure Eight, which uses the crowd to create training data for machine learning systems and ScaleHub, which provides managed services to process hand- or machine-written documents. These providers act as an intermediary between one or many crowdsourcing platforms and the customer.

A critical point of providing managed service is guaranteeing SLAs, which enable the customer to effectively manage outsourced workflows and enable business-partnerships with high levels of trust and commitment [8]. For guaranteeing SLAs, it is a crucial capability of the service provider, to be able to manage the uncertainty of crowd work [9]. Without a service provider, the customer has to break the tasks down to appropriate-sized subtasks, which the customer send to a crowdsourcing platform, providing some form of incentive. The crowd workers process the task, and the customer has to control the quality of the results and recombine them. If the customer instead decides to outsource this workflow to a meta-platform, the customer only needs to settle on SLAs, mainly to specify the desired quality, time and throughput. Now the meta-platform handles the governance, whereas the customer can rely on the agreed SLA. Therefore, using a service provider can result in a more reliable, more comfortable, and even cheaper workflow for the customer, since the customer does not need to exercise governance anymore and also gains the benefits of SLA from the meta-platform.

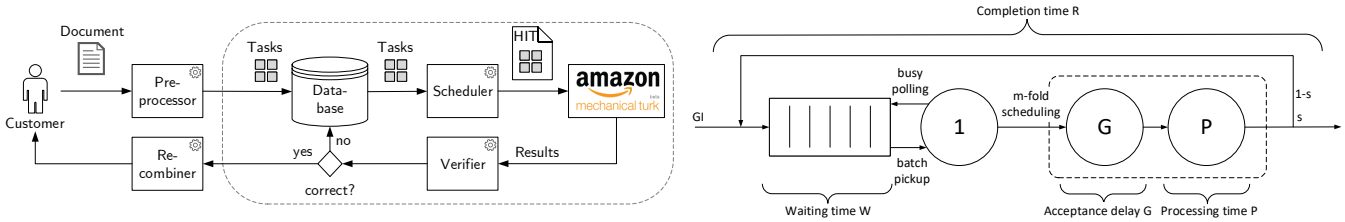
For providing and guaranteeing SLAs, meta-platform operators need to have a thorough understanding of their system, but also detailed knowledge of the processes within the crowdsourcing platforms they use. Therefore, this work presents a first model of a meta-platform including the underlying crowdsourcing platform that enable meta-platform operators to define SLAs and make trade-offs between cost- and time-constraints.

C. Related Work

Most of the existing crowdsourcing literature focuses on complex tasks and idea generation as well as concrete case-studies [4]. In contrast, the body of work dealing with mathematical models for crowdsourcing, especially from queueing theory, is relatively small [10]. Faridani et al. [11] model the arrival of workers using a non-homogeneous Poisson-process and derive strategies for pricing tasks to be completed on time. Bernstein et al. [12] analyze the problem of real-time crowdsourcing by modeling platforms as a $M/M/c$ queue. Specifically, they analyze a retainer approach, where workers are paid to be on call to achieve interactive response times. In [13], crowdsourcing platforms are modeled as a $M/M/c-\infty$ queue to investigate the relationships between the number of active workers, the worker utilization and the pre-processing delay, which is the time until a task is picked up by a worker.

Iperiotis et al. [14] focus on the analysis of existing platforms, rather than on the modeling, and use a crawled dataset from MTurk to investigate characteristics of the platform. The authors note that most activity on MTurk is concentrated around small tasks with small rewards. They also observe a heavy-tail completion time distribution, which poses a problem for requesters that need a high degree of reliability. Similar observations were also made on other crowdsourcing platforms like Microworkers [15].

In this work, queueing theory is used to model crowdsourcing systems, where an intermediary service provider manages the complexity of crowdsourcing for its clients. Scheduling strategies for the service provider are investigated by well-established analytical and simulative methods. Gebert et al. [16] provide a discrete-time queueing model for evaluating the performance of virtualized network functions. Their work provides analytical solutions for the batch size and the waiting time in a system, where incoming requests wait for a fixed interval and are then forwarded. This work uses a similar model for analyzing scheduling algorithms, where the service provider batches incoming tasks and then forwards them to a crowdsourcing platform. However, we extend the model of Gebert et al. by an additional maximum batch size and provide analytical solutions for the time-continuous case.



(a) Real-world system description

(b) Queueing model

Fig. 1: ScaleHub platform model and queuing model

III. System Model and Dataset

In the following, we derive the model that is later used in the evaluation from the commercial platform ScaleHub. The model parameters are extracted from a dataset provide by the platform operators.

A. Crowdsourcing Service Provider

1) ScaleHub: ScaleHub is a service provider that uses crowdsourcing for the large-scale processing of documents where optical character recognition fails. The platform is schematically depicted by Figure 1a. Customers sent documents to ScaleHub and a *preprocessor* decomposes the document to small tasks, e.g., capturing alphanumerical or numerical data. The small tasks are temporarily stored in a *database*, and later picked up by a *scheduler* that combines multiple tasks to a Human Intelligence Task (HIT) and schedule this HIT to MTurk. Crowd-workers on MTurk can then decide from available HITs, not only those submitted by ScaleHub, which HIT they want to work on. Ultimately, each HIT is completed by at most one crowd-worker. Therefore, the *scheduler* also schedules each task at least twice, so that the results provided by the crowd-workers can be verified by comparison. If the results are found to be incorrect, the incorrect tasks return to the *database*, where they can be scheduled again. Otherwise the results are recombined and the processed document is sent back to the customer. The dashed region surrounds the region of interest of this work. The preprocessing and recombination steps are not considered because they can be considered as fixed delay components and do not add value for the question on how to schedule tasks to crowdsourcing platforms. Therefore, we treat the tasks arriving from the preprocessor as the arrival process.

2) Queuing Model: Figure 1b shows a general queueing model describing the ScaleHub system. Tasks arrive, constituting an arbitrary arrival process with batch arrivals. The tasks wait in the queue, which is repeatedly polled by the scheduler. The scheduler can decide to pick up tasks, put them into a batch and schedule m copies of that batch to the crowdsourcing platform. Since we can not measure the number of workers available, the platform is modeled using two delay resources with infinite capacity. The first delay resource represents the acceptance delay, which is the time until a worker decides to start the work. The second resource represents the processing time, which is the time the worker needs to actually process the batch. The processing time may depend on the batch size. After the batch leaves the platform, each individual task has a chance of s to be correct. Correct tasks leave the system. Incorrect tasks return to the queue, where the scheduler can decide how to proceed. This model leaves room for decisions of the scheduling algorithm, while still being modest in complexity. Further, it allows analytical solutions for basic scheduling strategies and arrival processes.

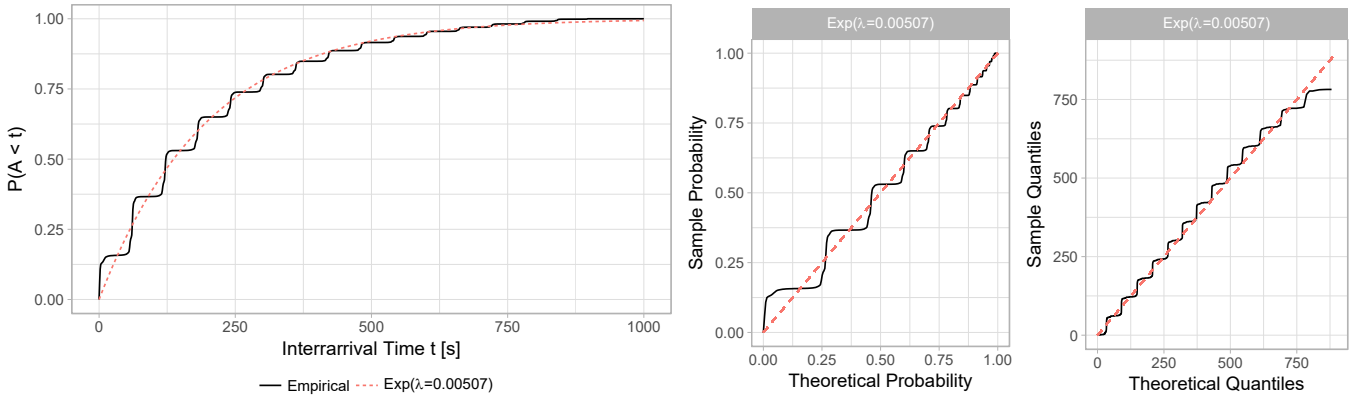
B. Model Parameters

We analyse a data set from the ScaleHub platform to derive realistic model parameters. The provided dataset consists of 20,000 batch (documents) arrivals at ScaleHub, containing almost 3,000,000 individual tasks (data capture fields) during December 2017 to March 2018. The tasks are recombined to 230,000 HITs at MTurk. Using special recombination logic, ScaleHub creates HITs of similar complexity and ensures the confidentiality of sensitive information in the tasks. Therefore, the HITs in this dataset are actually composed of a multitude of several small tasks.

The entire data analysis has been conducted using the R programming language [17] and the `fitdistrplus`-package [18]. We additionally used PP- and QQ-plots for the evaluation of the fitted distributions, as hypotheses tests become very sensitive to small deviations from the empirical data [19] if the data set gets large.

1) Arrival Process and Batch Sizes: First we analyze the multi-type batch arrival process at ScaleHub. For a description of the process we examine two properties: The interarrival times and the batch sizes.

The empirical cumulative distribution function (CDF) of the batch interarrival times at ScaleHub is visualized in Figure 2a. While the complete data range is shown in the figure, we remove outliers by cutting off data above the 95% quantile to impove the quality of the data fitting in the following. After the filtering, the observations range from 1s, which is the resolution of the time measurement, to 897s, the longest interarrival time. The steps of the empirical function substantiate because the arriving tasks are polled periodically in intervals of one minute and transferred into the task database. The dashed curve shows the CDF of an exponential distribution with $\lambda = 0.00507$, which is obtained by conducting a maximum-likelihood fit. We observe that the



(a) Interarrival time distribution (b) PP-Plot of fitting (c) QQ-Plot of fitting

Fig. 2: Interarrival time analysis

fit follows the general trend of the empirical CDF, but of course, does not reproduce the polling behaviour.

We use a PP- and QQ-plot to evaluate the quality of the fitting. First, we have a look at the PP-plot that shows the theoretical probability of the fitted distribution on the x-axis against the empirical probability from the data on the y-axis. The dashed line indicates a perfect fit. The black line oscillates around the dashed line because the exponential fit can not capture the steps in the empirical function. As the steps of the empirical function become smaller, the amplitude of that oscillation decreases. The QQ-plot shows the quantiles of the fitted function on the x-axis against the quantiles of the sampled distribution on the y-axis. The curve for the empirical data shows a behavior similar to the PP-plot. The empirical value oscillates around the perfect fit, although the amplitude is smaller. We can see a deviation for the high quantiles because the maximum of the sampled data is 897s, whereas the theoretical quantiles of the fit never reach 100%.

The plots indicate that the exponential fit is a sensible choice. The empirical data deviate from the fit because of the steps in the empirical CDF, but these deviations are relatively small and also show no clear trend.

Each arrival at ScaleHub is a batch and may contain multiple tasks. To obtain a comprehensive description of the arrival process, we analyze the distribution of the batch sizes and inspect possible dependencies between batch sizes and interarrival times. To visualize the distribution of batch sizes, the empirical CDF is shown by the black line in Figure 3. The chart shows the batch size of an arrival on the x-axis and the value of the CDF on the y-axis. The curve reaches a cumulative probability of 1 at 58, the maximum number of items that occurred in the data set. The red curve shows a maximum-likelihood fit of a negative binomial distribution with parameters $r = 5.9$ and $p = 0.34$. The fitted curve appears to match the empirical data very nicely. To validate this assumption, a χ^2 goodness of fit test is used. The null-hypothesis states that the empirical values follow a negative binomial distribution with $r = 5.9$ and $p = 0.34$. Even though

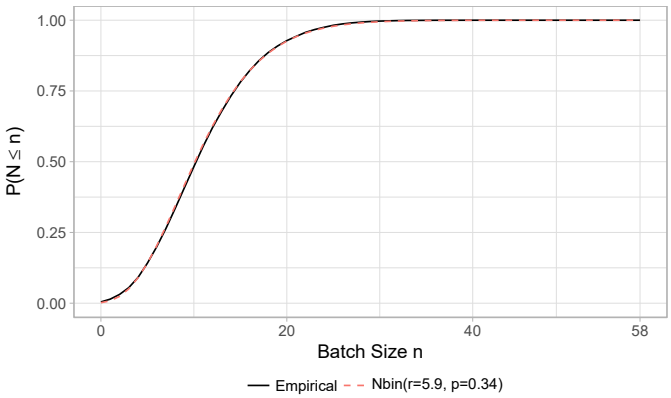


Fig. 3: Batch size distribution

the sample contains slightly more than 20,000 data points, the test can not reject the null-hypothesis (p-value 0.2764). Therefore, we can assume that the fit is reasonably good.

For simulating an arrival process, batch sizes and interarrival times need to be drawn from the fitted distributions. Drawing these values independently is only appropriate, if the batch sizes and interarrival times are independent. To check whether the assumption of independence is reasonable, the 95% confidence interval of the Pearson correlation is calculated. The interval is $[-0.021, 0.007]$ and contains 0. Therefore, we can assume that there is no linear dependency between the batch sizes and interarrival times. To capture non-linear dependencies, a Hoeffding D-test is conducted [20]. The test results in D-values of 0, therefore not rejecting the hypothesis of independence. Considering the results of both tests, it is reasonable to assume independence of batch sizes and interarrival times.

2) Crowdsourcing Acceptance and Processing Time: When a HIT is posted at MTurk, it goes through different states until it is finished. A HIT starts as *open* and gets eventually *accepted* by a crowd-worker. Sometimes workers change their minds and decide not work on a HIT after they accepted

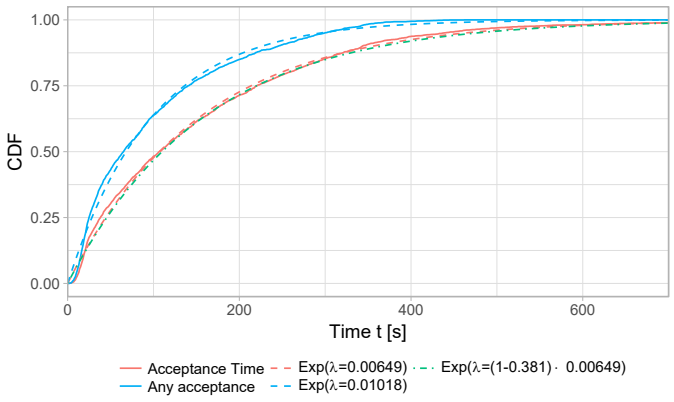


Fig. 4: CDFs of the acceptance times at MTurk

it. Therefore, HITs are returned with a probability r. Since HITs are returned with probability r, the number of returns R is geometrically distributed with $R \sim Geom(1-r)$. With probability $1-r$ HITs are processed and thus *finished*. The tasks in our evaluation have a return rate r of about 0.381. After the HIT is finished, the results are examined and either *approved* or *disapproved*.

Figure 4 shows the acceptance time t in seconds on the x-axis and the probability of waiting t seconds on the y-axis. First, we look at the blue curve, which depicts the empirical time until a HIT is *accepted*. The time ranges from a minimum of 1 to a maximum of 462 seconds. Since HITs can be returned and accepted again by another worker, this is not enough to fully describe the acceptance process. The red curve displays the time until the last acceptance, that is the acceptance after which the task is actually processed. The maximum value of the combined time is 1546s, which is not shown in the figure.

If a HIT is returned and accepted R times, we have R phases of acceptance. Since we use an exponential distribution to fit the time until a task is accepted, we can calculate the time until the last acceptance by $Exp((1-r)\cdot 0.00686)$ with $r = 0.381$ [21]. In Figure 4 this distribution is depicted by the green curve. It corresponds well to the direct fit of the time until the last acceptance, and therefore verifies the task acceptance model. In the remainder of this work, we use the direct fit of the time until the last acceptance and refer to that as the acceptance time G.

After a HIT is accepted, the worker begins processing it. A Pearson correlation analysis provides evidence that the processing time is influenced by the batch size. Intuitively, if a worker begins processing a HIT, the worker first needs some time to read the job description and time, dependent on the number of tasks in the HIT, to do the actual processing. Therefore, we use a linear model $P = \alpha B + \epsilon$ to describe the processing time P, where B is the batch size and ϵ is the error term. Using a least-square regression executed in `R`, we obtain $\epsilon = 18.90s$ and $\alpha = 6.28s$.

To evaluate the validity of the model, we check the normality of the residuals using a QQ-plot. The plot indicates, that the residuals are approximately normal, even though there are small deviations for very low and high quantiles. A scale-location plot is used to check the residuals for homoscedasticity, revealing that the variance is slightly increasing for larger batch sizes. Overall, the diagnostic plots do not show critical violations of the assumptions of a linear model, therefore indicating the model is a feasible choice. For the simulation, we calculate the processing time using the linear model and add normally distributed residuals. In the event of a negative processing time, we re-roll.

3) Success Rate: Each HIT is a composition of several tasks, i.e., while completing a HIT a worker has to transcribe small pieces of different documents. In particular, each task is actually part of two HITs and therefore two instances are scheduled. We can use this fact to calculate the success probability s for a single task.

Let A be the event that two results agree. We assume that there is no agreement if the instances are false, that is, both instances do not contain the exact same mistake. Further, we assume the success of two instances to be independent, therefore we get $P(A) = s^2$. Based on the data we calculate the success probability of a single task $s_t = \sqrt{P(A_t)}$ with $P(A_t) = 0.925$.

IV. Analytical Evaluation

In this section, we start with a basic system and scheduling strategy. We derive an analytical solution that helps to understand the general underlying principles and effects.

A. System Description

We start the evaluation of scheduling strategies by considering a basic system similar to the one depicted in Figure 1b. For the analysis of the model, we assume single-request Poisson arrivals. Poisson arrivals correspond very well to the empirical data as discussed in Section III-B1, although the real process is a batch arrival process.

When the scheduler picks up tasks from the queue, it combines them to a batch, i.e., a single HIT in case of MTurk, and forwards them to the crowdsourcing platform. In the following, we consider a simple scheduler that decides to pick up a batch whenever one of two conditions is met: 1) k requests are waiting in the queue or 2) the oldest request in the queue has already waited for τ seconds. The event C_k implies that a batch is forwarded because the maximum size k is reached, whereas the complementary event $C_\tau = \overline{C_k}$ implies that the batch is forwarded because the interval τ is over. The random variable (RV) N is defined as the number of arrivals within an interval of length τ. The processing time of the scheduler itself is negligible.

After the batch reaches the crowdsourcing platform it has to wait for acceptance and is processed afterwards. The distributions used for modeling the system are the waiting time distribution $W(t)$, the distribution of the acceptance delay $G(t)$, the discrete batch size distribution $b(i)$ and the processing time distribution $P_i(t)$. The processing time of the crowdworker $P_i(t)$ is a special case, since it is modeled using

a linear model and therefore, depends on the batch size i. After the request is processed, it leaves the system.

Recurring functions in the following sections are the unit-step function $\theta(x)$ and the Dirac-function $\delta(x)$, which is the derivative of $\theta(x)$. The function $U_{[a,b]}(t)$ denotes the distribution functions of a uniformly distributed RV $X \sim U(a,b)$ and $Be_{a,b}(t)$ the distribution function of a beta distributed RV $Y \sim Be(a,b)$. Finally the operator $*$ refers to the discrete or continuous convolution operation.

B. Case Probabilities

To find an analytical model, we need to distinguish between two different cases of batch forwarding: 1) Forwarding because the maximum size k is reached (event C_k) and 2) forwarding because the interval τ is over (event C_τ)

Since we assume a Poisson arrival process, the number of arrivals N in a fixed timespan follows a Poisson distribution. The number of arrivals in an aggregation interval is shifted by one, since there is always one request which starts the interval. Therefore $P(C_\tau)$ is the probability of less than $k-1$ arrivals after an interval is started:

$$P(C_\tau) = \sum_{i=0}^{k-2} P(N=i) = \sum_{i=0}^{k-2} \frac{(\lambda\tau)^i}{i!} e^{-\lambda\tau} \tag{1}$$

$$P(C_k) = 1 - P(C_\tau) = 1 - \sum_{i=0}^{k-2} \frac{(\lambda\tau)^i}{i!} e^{-\lambda\tau} \tag{2}$$

Equations 1 and 2 quantify the probability that batches are forwarded because the maximum waiting time τ was reached or a maximum size k. In the following, these findings are used to calculate the batch size distribution for the forwarded batches.

C. Batch Size

Intuitively, the batch size ranges between 1 and k because at most k requests can wait in the queue before the scheduler decides to forward a batch. To find the exact batch size distribution $b(i)$, we distinguish between two cases C_k and C_τ having probabilities $P(C_k)$ and $P(C_\tau)$. The batch size distribution $b(i)$ can be calculated, by first finding the conditional batch size distributions $b_{|C_k}(i)$ and $b_{|C_\tau}(i)$ and combining them using a mixture distribution. The batch size in case C_k is fixed at k, therefore $b_{|C_k}(i) = \delta(i-k)$. For case C_τ, the batch sizes follow a conditional Poisson distribution as in Equation 3, because we have Poisson arrivals and know there must be less than k for C_τ to happen.

$$b_{|C_\tau}(i) = P(N=i|i<k) = \begin{cases} \frac{P(N=i)}{\sum_{j=0}^{k-1} P(N=j)} & i \leq k \\ 0 & i > k \end{cases} \tag{3}$$

Using a mixture distribution to combine both cases, we get the batch size distribution as

$$b(i) = P(C_k)\cdot\delta(i-k) + (1-P(C_k))\cdot P(N=n|n<k). \tag{4}$$

D. Waiting Time

To find an analytical solution for the waiting time W of requests, the same procedure as for the batch size distribution is used. First, we distinguish between the two cases C_k and C_τ and then recombine the conditional distributions $W_{|C_k}(t)$ and $W_{|C_\tau}(t)$ using a mixture distribution.

A formula for $W_{|C_\tau}(t)$ is given by

$$W_{|C_\tau}(t) = \underbrace{\lambda^{-1}\cdot\theta(t-\tau)}_{\text{first arrival}} + \underbrace{(1-\lambda^{-1})\cdot U_{[0,\tau]}(t)}_{\text{arrival during } \tau}. \tag{5}$$

We distinguish between the first request, which starts the aggregation interval and consecutive arrivals. The first arrival sees an empty system with probability $x(0)$ [16], which for Poisson arrivals is a backward recurrence time λ^{-1}. Because the first arrival starts the interval, it has to wait for time τ in the queue. For the consecutive arrivals, each point in time is equally likely due to the memorylessness of the arrival process. Therefore the waiting time is a uniform distribution over the interval $[0,\tau]$ [22].

Next, we need to find $W_{|C_k}$. Even though a batch is forwarded with the k-th arrival, we still consider a full interval τ with n arrivals, where $n \geq k$. T_i denotes the random variable for the time of the i-th arrival. Since there are at least k arrivals in the interval, we can get the number of events using a conditional Poisson distribution as in Equation 6:

$$P(N=n|n\geq k) = \begin{cases} \frac{P(N=n)}{1-\sum_{i=0}^{k-1} P(N=i)} & \text{if } n \geq k \\ 0 & \text{if } n < k \end{cases} \tag{6}$$

Conditioning over the number of events N in the interval, we can use a beta distribution scaled to $[0,\tau]$ to calculate T_i, as T_i is the i-th order statistic of n events uniformly [22] distributed over $[0,\tau]$ [23].

$$P(T_i < t|N=n) = \underbrace{Be_{i,n-i+1}}_{\substack{i\text{-th order} \\ \text{statistic of} \\ n \text{ events}}} \underbrace{(t/\tau)}_{\substack{\text{scaled} \\ \text{to } [0,\tau]}} \tag{7}$$

Since the batch does not have to wait for a full interval τ, but is already accepted with the arrival of the k-th request at time T_k, the waiting time $W_{|C_k,i}$ of the i-th request is the time difference between the i-th to the k-th order statistic, which again is a beta distribution [24]. The k-th request triggers the forwarding and therefore does not wait at all.

$$W_{|C_k,i}(t|N=n) = \begin{cases} \theta(0) & i = k \\ Beta_{k-i,n-k+i+1}(t/\tau) & i = 1\ldots k-1 \end{cases} \tag{8}$$

Using the law of total probability for all realizations of N yields

$$W_{|C_k,i}(t) = \sum_{n=k}^{\infty} P(N=n|n\geq k)\cdot W_{|C_k,i}(t|N=n) \tag{9}$$

Since the batches are fixed size at size k, each of the k positions within a batch is equally likely. Therefore, we get $W_{|C_k}$ by averaging the $W_{|C_k,i}$

$$W_{|C_k}(t) = \frac{1}{k} \cdot \Big(\sum_{i=1}^{k-1} W_{|C_k,i}(t) + \theta(0) \Big) \tag{10}$$

Now we can combine the cases C_k and C_τ using a mixture distribution yielding:

$$\begin{aligned} W(t) &= P(C_\tau) \cdot W_{|C_\tau} + P(C_k) \cdot W_{|C_k} \\ &= P(C_\tau) \cdot \big(\lambda^{-1} \cdot \theta(t-\tau) + (1-\lambda^{-1}) \cdot U_{[0,\tau]}(t)\big) + \\ &\quad P(C_k) \cdot \big(\frac{1}{k} \cdot \Big(\sum_{i=1}^{k-1} W_{|C_k,i}(t) + \theta(0) \Big)\big) \end{aligned} \tag{11}$$

E. Completion Time

Finally, we consider the completion time R for a request to pass through the system. If we had an univariate distribution for the processing time P, we could simply use $R = W + P + G$, where G is the acceptance delay at the crowdsourcing platform. The data analysis has shown, that a model, linear in the size of the job, is a sensible choice for the processing time. When using a linear model, we can not simply add the RVs because of interdependencies between the batch size, waiting time and processing time.

Therefore, we define W_i' as the waiting time of a request, given the batch size is i and calculate the total time by conditioning over i:

$$R(t) = \sum_{i=1}^{k} P(B = i) \cdot (P_i(t) * W_i'(t) * G(t)). \tag{12}$$

To calculate W_i', we again distinguish between the cases C_k and C_τ. If the batch has size k, the waiting time is the same as calculated for C_k in Section IV-D. If the batch is smaller than k, the first request in the batch has to wait for τ and all other requests have to wait a uniformly distributed time because of the properties of the Poisson arrivals discussed in [22]. W_i' is therefore given by:

$$W_i'(t) = \begin{cases} \frac{1}{i} \cdot \theta(t-\tau) + \frac{i-1}{i} \cdot U_{[0,\tau]}(t) & i = 1 \dots k-1 \\ W_k(t) & i = k \end{cases} \tag{13}$$

F. Guaranteeing completion time SLAs

Since our goal is to give completion time SLAs by appropriate choices of k and τ, we are mainly interested in the 99% quantile of the completion time. Additionally, we look at the mean batch size, i.e., the size of the HITs scheduled on MTurk, to estimate the costs. The size of the batches can be directly transformed into costs, as the workers are payed a fixed amount per HIT in our case. Consequently, larger HITs result in less cost per task. We use an arrival rate of $\lambda = 0.006\frac{1}{s}$ for the following nummerical example. This rate it not related to the actual parameters observed from the ScaleHub data, as the example is just intended to help obtaining a general understanding of the system behavior.

Figure 5a shows the relationship between k, τ and the mean batch size. We see that the curves originate at $k = 5$ and disperse for higher values of k. Higher values of τ lead to higher mean batch sizes. For small values of τ and high values of k the batch size is almost exclusively determined by τ. This can be seen for $\tau = 1500$s, where changes in k, e.g., from 20 to 30, do not affect the batch size. For high values of τ, the batch size is almost exclusively determined by k, as can be seen for $\tau = 3500$s. This means that k and τ synergize and should be chosen carefully and with respect to each other.

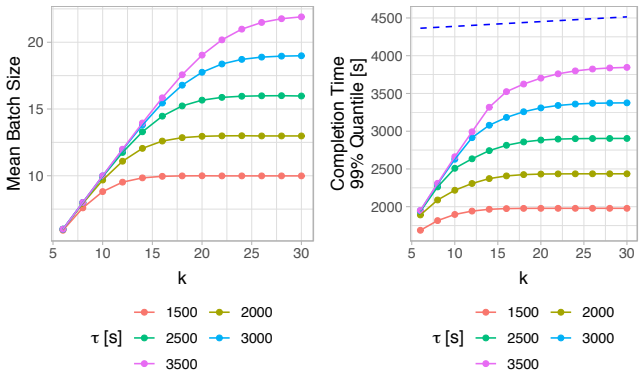
Figure 5b shows a similar diagram for the 99% quantile of the completion time. For high values of k, the curves converge to the sum of the processing time and maximum waiting time τ. We can find a theoretical upper bound for the 99% quantile which is the sum of τ, the 99% quantile of the acceptance time and the 99% quantile of the processing time for the largest batch size. For $\tau = 3500$s, this theoretical upper bound is depicted by the dashed line. For practical purposes, this bound appears to be quite loose. The figure indicates that τ is very effective for controlling the 99% quantile, by providing an upper bound on the waiting time, which is the main component of the completion time.

If we want to guarantee SLAs, we are mainly interested in the 99% quantile and the batch size (which corresponds to the cost). In this case, suitable values of k and τ can be derive from our mode. In our concrete example, we can use the values from Figure 5c. Please note that due to limited space in the graphic not all combinations are shown and not all points are labeled. The plot shows the trade-off between the batch size on the x-axis and the 99% completion time quantile on the y-axis. Values for τ are encoded using colors, whereas the values for k are written in the plot. To guarantee that 99% of the requests finish within 2500s, we could for example choose $k = 8$ and $\tau = 3000$s, or $k = 10$ and $\tau = 2500$s or we could choose $k = 16$ and $\tau = 2000$s. In general, choosing a larger batch size might be cheaper, but too large batches might not be well received by the workers at the crowdsourcing platform.

V. Simulative Evaluation

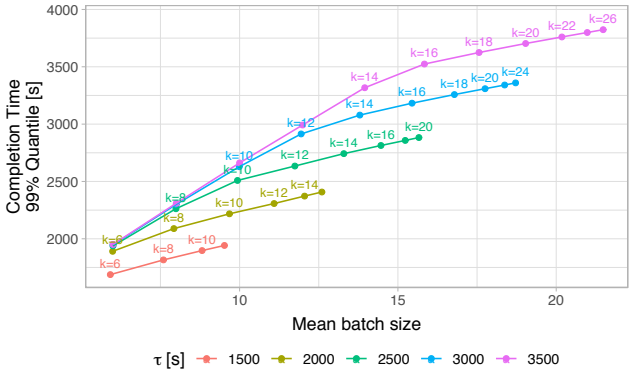
Next we evaluate the systems with batch-arrivals and failed requests. Due to the limitations of our analytical solution, we use a Java-based discrete-event simulation here. To generate random variates and for statistical functions we rely on the Apache Commons Math Library (version 3.6.1) [25]. The simulation was validated by simulating the scenario used in the analytical analysis and comparing the simulated and analytical results. Due the limited space, a detailed comparison is not shown here.

As shown before, the batch sizes of the arriving requests can be approximated by a negative binomial distribution with parameters $r = 5.9$ and $p = 0.34$. We further assume a Poisson arrival process with $E[A] = 200$s, which corresponds to the empirical data from the ScaleHub platform. The results



(a) Mean batch sizes

(b) Completion time 99% quantile



(c) Parameter trade-offs

Fig. 5: Analytical results for a system without failed requests and Poisson arrivals

of crowdsourced tasks can be insufficient in quality or plain incorrect. As a counter measure, platforms like ScaleHub often schedule multiple copies of the same task and compare the results submitted by independent workers. Our simulation model takes those failed requests into account and enables us to investigate the effects on the completion time and cost. In contrast to the analytical model, requests now have a success probability of s. With probability $1 - s$ the request fails, e.g., because the result does not meet the desired quality, and returns to the queue. To handle failed requests, the capabilities of the scheduler are extended. Each batch is now scheduled m-fold, meaning m identical copies are sent to the crowdsourcing platform. The success probability s of a single instance is analyzed in Section III-B3. For m instances, we assume the request to be a success if two out of m instances agree. A majority decision is not needed because we assume incorrect results to differ from each other. The success probability s_m of m instances is given by

$$s_m = 1 - \underbrace{(1-s)^m}_{\text{all incorrect}} - \underbrace{m \cdot (1-s)^{m-1} \cdot s}_{\text{one correct}} \quad \text{for } m > 1. \quad (14)$$

In the following, we also consider the cost to successfully process a task. In general, only successfully completed HITs are paid, i.e., if a crowd-worker fails to meet the quality requirements, the worker does not receive any payment. In the case of ScaleHub, the quality control of a task is accomplished by asking multiple workers to transcribe the same piece of a document. If a worker diverges with his input from the majority, this task is considered as invalid. A ScaleHub HIT consists of several of these tasks, and is considered invalid, if a fixed share of the tasks is invalid. The data from ScaleHub shows that even if single tasks in a HIT might be invalid, the share of invalid and thus unpaid HITs is neglectable small. Therefore, we simply approximate the cost per successfully process a task, as the cost per HIT divided by the number of tasks per HIT. In the case of ScaleHub the cost per HIT type are constant and the following example, we assume of 1 USD per HIT.

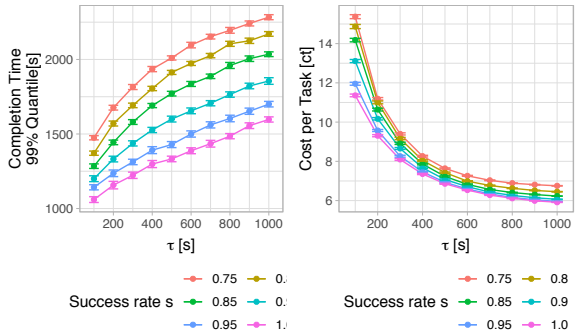
In the presence of failed requests, the 99% quantile of the completion time may rises significantly because failed requests have to take additional runs through the system. To illustrate the effects of different success probabilities, the system is simulated for 100 runs of length 100,000s. The first 20,000s simulation time are not considered in the evaluation to account for the transient phase. The success probabilities range from 0.75 to 1 in steps of 0.05, which are realistic values for reasonably complicated tasks. Further, we always schedule $m = 2$ instances, similar to the current ScaleHub implementation and use a fixed value of $k = 35$.

Figure 6a shows the results of the simulation. The 99% quantile of the completion time is plotted on the y-axis. The x-axis shows the different values of the waiting time limit τ. Success rates are indicated by different colors. All curves are monotonically rising, meaning for higher values of τ the quantile also rises. This is not supprising as τ is an upper bound for the waiting time of the tasks. Additionally, the quatile also increases as the success probability decreases. This is caused by the raising number of failed tasks that need to be processed again. However, we still observer that τ is an efficient way to control the quantil of the completion time. Especially for lower success rates causing more tasks to be processed mutilple times, a low upper bound for the waiting time is usefull.

However, decreasing τ also has severe drawbacks. Figure 6b shows the effects that the changes in τ have on the cost. We observe that a change τ from $800s$ to $200s$ almost doubles the costs. For large request volumes, lowering the value of τ can therefore become quite expensive. Here, more sophisticated scheduling mechanisms that prioritize already failed request can be a possible solution.

VI. Conclusion

In the paper, we analyzed how scheduling mechanisms can help crowdsourcing-based service providers to meet completion time SLAs. To this end, we derived a system model based on the insights of a commercial platform and considered a simple strategy that schedules tasks either if k tasks are queued or if the oldest task is already waiting for τ seconds. The system model and the scheduling mechanism



(a) 99% completion time quantile

(b) Cost per task

Fig. 6: Controlling completion time quantile using τ

laid the foundation for an analytical and a simulative evaluation, while the model parameters were obtained by the analysis of a large-scale real-world dataset. The analytical model showed that for a fixed completion time SLA several combinations of k and τ are feasible, but especially τ is efficient for controlling the completion time. However, further crowdsourcing specific aspects have to be taken into account, here, e.g., the maximum size of work a crowd-worker is willing to accept. The analytical model enables crowdsourcing-based service providers to get an estimation of reasonable scheduler parameters while still being able to select the parameter combination best for their specific use-case. A simulation model further extended the analytical model to feature batch arrivals and failed tasks. The results showed that in the presence of failed tasks, τ can still be effectively used to control the completion time, however, the costs for processing the tasks can increase significantly.

An interesting metric that the model does not capture, but that is also not the scope of this work, is the throughput. Since the system is modeled using delay-resources, the throughput of the system is unbounded. But for a reasonably-sized crowdsourcing platform and work, which does not need specialized skills, the number of workers is large enough that delay resources are a reasonable approximation.

This paper provided first insights on how schedules can be used to match completion time SLAs. As a next step more sophisticated scheduling mechanisms can be analyzed to provide better trade-offs between the completion time and the costs. Further, the use of multiple crowdsourcing platforms with different properties, i.e., differing success rates and costs, can provide further interesting insights.

Acknowledgement

This work is supported by the Forschungsgemeinschaft (DFG) under Grants HO4770/2–2 and TR257/38–2 and by the EU's European Regional Development Fund (ERDF) and the federal state of Brandenburg under project KITSUNE. The authors alone are responsible for the content.

References

[1] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, 2006.

[2] E. Estellés-Arolas and F. González-Ladrón-De-Guevara, "Towards an integrated crowdsourcing definition," *Journal of Information science*, vol. 38, no. 2, 2012.

[3] M. Hirth, T. Hoßfeld, and P. Tran-Gia, "Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms," *Mathematical and Computer Modelling*, vol. 57, no. 11-12, 2013.

[4] R. Buettner, "A systematic literature review of crowdsourcing research from a human resource management perspective," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, 2015.

[5] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013.

[6] M. Hirth, J. Jacques, P. Rodgers, O. Scekic, and M. Wybrow, "Crowdsourcing technology to support academic research," in *Evaluation in the Crowd. Crowdsourcing and Human-Centered Experiments*. Springer, 2017.

[7] J. Pedersen, G.-J. de Vreede, D. Kocsis, A. Tripathi, A. Tarrell, A. Weerakoon, N. Tahmasbi, J. Xiong, W. Deng, and O. Oh, "Conceptual foundations of crowdsourcing: A review of is research," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013.

[8] J. Goo, R. Kishore, H. R. Rao, and K. Nam, "The role of service level agreements in relational management of information technology outsourcing: an empirical study," *MIS quarterly*, 2009.

[9] D. Nevo and J. Kotlarsky, "Primary vendor capabilities in a mediated outsourcing model: Can it service providers leverage crowdsourcing?" *Decision Support Systems*, vol. 65, 2014.

[10] S. R. Chakravarthy and S. Ozkar, "Crowdsourcing and stochastic modeling," *Business and Management Research*, vol. 5, no. 2, 2016.

[11] S. Faradani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time." *Human computation*, vol. 11, 2011.

[12] M. S. Bernstein, D. R. Karger, R. C. Miller, and J. Brandt, "Analytic methods for optimizing realtime crowdsourcing," *arXiv preprint arXiv:1204.2995*, 2012.

[13] C. Schwartz, K. Borchert, M. Hirth, and P. Tran-Gia, "Modeling crowdsourcing platforms to enable workforce dimensioning," in *Telecommunication Networks and Applications Conference (ITNAC), 2015 International*. IEEE, 2015.

[14] P. G. Ipeirotis, "Analyzing the amazon mechanical turk marketplace," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 2, 2010.

[15] M. Hirth, T. Hoßfeld, and P. Tran-Gia, "Anatomy of a crowdsourcing platform-using the example of microworkers. com," in *2011 Fifth international conference on innovative mobile and internet services in ubiquitous computing*. IEEE, 2011.

[16] S. Gebert, T. Zinner, S. Lange, C. Schwartz, and P. Tran-Gia, "Performance modeling of softwarized network functions using discrete-time analysis," in *Teletraffic Congress (ITC 28), 2016 28th International*, vol. 1, 2016.

[17] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, 1996.

[18] "Fitdistrplus package for R," https://cran.r-project.org/web/packages/fitdistrplus/, accessed: 2019-09-21.

[19] "Homepage of the membership magazine of the american statistical association," http://magazine.amstat.org/blog/2010/09/01/statrevolution/, accessed: 2019-00-21.

[20] W. Hoeffding, "A non-parametric test of independence," *The annals of mathematical statistics*, 1948.

[21] R. Nelson, *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer Science & Business Media, 2013.

[22] L. Kleinrock, *Queueing Systems: Volume I: Theory*. Wiley-Interscience, 1975.

[23] A. K. Gupta and S. Nadarajah, *Handbook of beta distribution and its applications*. CRC press, 2004.

[24] H. A. David and H. N. Nagaraja, *Order Statistics*. Wiley-Blackwell, 2003.

[25] "Apache Commons Math library for Java," http://commons.apache.org/proper/commons-math/, accessed: 2019-03-21.