Tuning Crowdsourced Human Computation

Chen Cao*, Jiayang Tu*, Zheng Liu*, Lei Chen* and H. V. Jagadish †
*The Hong Kong University of Science and Technology, Hong Kong, China
†University of Michigan, Ann Arbor, MI, USA

***Abstract*—As crowdsourcing has been dramatically investigated and utilized to address problems in the real world, it is essential and important to think about performance optimization. Analogous to computer systems with CPUs, treating each worker as a HPU (*Human Processing Unit* [1]) and studying the performance optimization on top of HPUs are interesting perspectives to resolve crowdsourcing issues. However, as we characterize HPUs in detail for this purpose, we find that there are significant differences between CPUs and HPUs, leading to the need of completely new optimization algorithms.**

In this paper, we study the specific optimization problem of obtaining results the fastest for a crowdsourced job with a fixed total budget. In crowdsourcing, jobs are usually broken down into sets of small tasks, which are assigned to workers one at a time. We consider three scenarios of increasing complexity: *Identical Round Homogeneous Tasks*, *Multiplex Round Homogeneous Tasks*, and *Multiple Round Heterogeneous Tasks*. For each scenario, we analyze the stochastic behavior of the HPU clock rate as a function of the remuneration offered. After that, we develop an optimum Budget Allocation Strategy to minimize the latency of the job completion. We validate our results through extensive simulations and experiments on Amazon Mechanical Turk.

I. Introduction

Human Computation [2] has emerged in recent years as a new and exciting computation paradigm. As a useful complement of traditional computer systems, human computation naturally allows tasks with human-intrinsic values or features, like comparing emotions of speeches, identifying objects in images and so on. The emergence of public crowdsourcing platforms like Amazon Mechanical Turk (AMT) has provided a scalable manageable workforce resource and boosted the utilization of this long-discovered human cognitive ability [3]. A wide range of data-driven applications now benefit from human computation by considering it as a new computing component. Examples include *a*) crowd-powered databases [4]–[6] and fundamental operators like filtering [7] and Max [8], [9], group-by [10], *b*) advanced data processing technologies like image tagging [11], schema matching [12] and entity resolution [13], and *c*) combinatorial problems like planning [14] and mining [15].

As crowdsourcing becomes more prevalent, there is an effort to understand and characterize it better. In this regard, it has been suggested that the system can be viewed as comprising Human Processing Units (HPUs) that are analogous to CPUs of traditional computers. In this case, a single atomic task performed by a worker is then one "instruction" of the HPU, and the time to respond is the "clock cycle". However, the HPU still has significant characteristics that differ from those of a CPU:

i) the clock time is **stochastic**; *ii*) the results are **error-prone** according to a probability; *iii*) the cost includes **monetary** expense.

Given the HPU abstractions, one can consider optimizing many aspects of HPU processing. In this paper, we aim at minimizing the total latency expectation of a computational task. For this purpose, it is a feasible starting point to optimize the clock rate of the HPUs. The clock rate of the HPU is variable, thus it is different for each instruction and each instance. In a crowdsourcing workforce market, a task is released into the marketplace with a promised reward, and then it is the freedom of the "workers" to select the tasks to work on according to their interests. Recent studies on AMT report that the task acceptance duration follows an exponential distribution [16], [17], and the task acceptance rate λ_o is mostly determined by the promised reward [18] and the type (difficulty) of the task [17]. Similarly, the processing time of a task follows an exponential distribution with another rate [19], whereas the processing rate is independent of the promised incentives [18]. As stated above, once the task has been specified, the only task-owner input that can control the completion time is the *payment*. If we only have one task to be performed by HPUs, the solution is very simple – the more we can afford to pay, the task will be completed at a faster speed as we expected.

Of course, the computational job at hand is typically performed with the aid of many HPU tasks [4]–[6]. In fact, a typical algorithm architecture repeats each task multiple times. And the requester in most crowdsourcing platforms will issue a large number of HPU tasks in parallel at the same moment, each task is possible to be repeated, and then the requester waits for all HPUs to return the results. According to this real-world circumstance, there is limited value to optimize the clock rate of a single HPU in isolation: what really matters is the latency of the entire computation, which is determined by the longest duration of the set of parallel tasks. Therefore, our HPU clock rate optimization problem is how to allocate a given fixed budget in a manner that minimizes the total latency of a set of tasks involving HPUs.

Unfortunately, such a problem has not been effectively addressed so far. In fact, most of the existing works on accelerating the clock rate of HPU simply focus on the problem of maximizing the number of completed tasks *w.r.t.* a given deadline [19]–[22]. Apparently, they are not applicable to the foregoing scenario, where all the tasks have to be completed and the overall latency is determined by the longest duration. Besides, the existing works (*e.g.* [22]) may set the

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.148

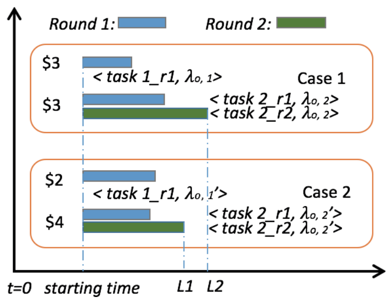


Fig. 1: Motivating Example 1 - Repetitions

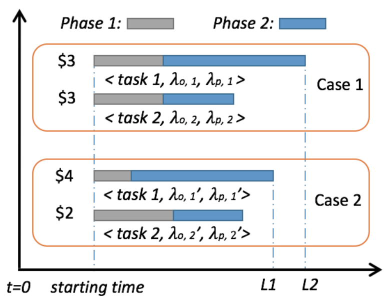


Fig. 2: Motivating Example 2 - Heterogeneous

prices of the tasks in a dynamic way (*i.e.* the prices are adapted constantly), which is both technically and morally impractical for real crowdsourcing platforms. Indeed, the tasks' prices have to be determined before they are published. Finally, HPU tasks may incur various processing latency, either because of their heterogeneous difficulties or the different number of repetitions; however, the previous works (*e.g.* [22]) simplify the HPU tasks to be identical, which severely restricts their application in practice.

To further illustrate the differences between our paper and [22] and demonstrate the challenging issues to optimize HPU clock rates, let us consider the following two motivating examples, both based on a crowd-powered database, as proposed in [4]–[6], [8], [9].

Motivating Example 1 *(Figure 1). Generally, most existing crowdsourcing platforms rely on task redundancy to determine the ground truth, this example demonstrates the latency discrimination of tasks with repetitions due to different budget allocation schemes. Consider a sorting task on 4 given items* $O = \{o_1, o_2, o_3, o_4\}$*. And as the tasks are homogeneous, we simply consider the on-hold phase (the phase from when the task is published to the time when it is accepted by a worker), thus only the task acceptance rate* λ_o *matters in this example. According to the user's requirements, the query planner, for example the "next votes" proposed in [9], decomposes the sorting task into atomic pairwise voting tasks* $T = \{\{o_1, o_2\} \times 1, \{o_3, o_4\} \times 2\}$*, which means the HPUs are expected to run the comparison tasks on such pairs for 1 and 2 repetitions (times) respectively. In this case, task 1 will run in one round while task 2 has to run in two rounds according to the grouping method stated in the latter session. Note that the repetitions in the second task are supposed to be picked by different users. As illustrated in Figure 1, the two tasks commence at the same time, but in order to finalize the entire query, the database has to wait until the end of the longest atomic task.*

There are many choices for budget allocation. Here are two possible budget allocation methods: (1) evenly allocating $3 to each of these two tasks (case 1); (2) the second one is more load-sensitive, assigning $2 for task 1 and $4 for task 2 (case 2). The results for the two cases are shown in the figure, suggesting that the second option is better. But how could we predict this? Moreover, even if this is the better of these two choices, is it the best? What is the best allocation of the two tasks?

Motivating Example 2 *(Figure 2). It is a normal phenomenon that various tasks are published and contained in the task set. For example, in an image related task set, an image annotation task obviously has a higher difficulty level in contrast with the image labeling task. Now consider a more complex scenario in which the database is required to process two types of queries simultaneously, sorting and filtering [7], where the latter can also be decomposed into pairwise voting tasks (yes or no voting). Suppose two tasks are given* $T = \{\{o_1, o_2\} \times 1, \{o_3, yes?no\} \times 1\}$*. However, unlike the previous case, different types of tasks present different difficulty levels, which leads to different processing rates. As shown in Table I, for the same price* p_i*, the processing rate* λ_p *of sort voting is lower than that of yes or no voting. For every single task, the entire latency depends on how long it is offered before it is accepted, which is influenced by the reward offered; and the latency also relies on how long it takes to process the task, which depends on the task type but not the given price. Unlike the previous example, we now have to take the processing latency into account.*

Here are two possible budget allocation methods: (1) evenly allocating $3 to each of these two tasks; (2) balancing budgets according to the task difficulties, assign the sorting task with $4 and filtering task with $2. We get latencies as shown in the figure, which shows the second one gains a better result. Once again, these are obviously not the only allocation schemes, and our interest is in finding the optimum, compounded by the difficulty of predicting the uptake rate for any reward level and of additionally folding in the task completion time into the framework.

As shown in the motivating examples above, our paper supports to find the optimal overall latency of *(1) tasks with different numbers of repetitions which are supposed to be run sequentially* and *(2) heterogeneous tasks with different processing times*. The methods proposed in [22] are inapplicable to both cases since tasks are assumed to be identical and parallel, and are regarded as completed once accepted by the workers, thus task processing latency is not considered.

These two examples also reveal that the allocation of budget to the tasks have a profound impact on the overall HPU latency. The difficulty of finding the optimal allocation

TABLE I: HPU Processing Rates for Motivating Example

task type / reward($)	sorting vote	*yes* or *no* vote
1.5	1.5	2
2	2	3
3	3	5

strategy is two-fold: 1) the latency of an atomic task is a random variable which depends on the type of the task, the allocated budget, and the current workforce market situation, therefore it is non-trivial to predict the overall latency of a set of tasks, particularly when they are of different types; 2) the search space for finding an optimal solution is large so that efficient algorithms and/or approximation tradeoffs are necessary. The promised payment has a minimum granularity ($0.01 on AMT), which renders the tuning process a discrete, rather than a continuous optimization problem.

To address these challenges, we firstly develop a stochastic model based on the HPU characteristics to predict uptake rate as a function of reward amount, and to show how to estimate the model parameters. Using these results, we can determine the expected latency for any specific budget allocation choice. Then, we formally propose the *H-Tuning Problem* to *minimize the expected latency of a set of tasks through tuning the tasks' payments w.r.t. a given budget*, and a probabilistic analysis and tuning strategies are developed under three practical scenarios: *Identical Round Homogeneous Tasks, Multiplex Round Homogeneous Tasks, and Multiple Round Heterogeneous Tasks*. In each case, we show how to solve an optimization problem with a large feasible space of budget allocation choices.

To sum up, the following contributions are made in this paper.

- A stochastic model is developed to predict uptake rate as a function of reward amount, with which the expected latency can be determined for any specific budget allocation choice.
- In Section III, the *H-Tuning Problem* is proposed to minimize the expected latency of a given set of tasks, and the optimized tuning strategies are developed under three practical scenarios: *Identical Round Homogeneous Tasks, Multiplex Round Homogeneous Tasks, and Multiple Round Heterogeneous Tasks*.
- The performance of proposed strategies is verified on a real crowdsourcing platform, and with simulation in Section IV.

II. The HPU Model

In this section, we begin with the basic crowdsourcing framework, develop the HPU model and demonstrate how to estimate HPUs' parameters. We lay the foundation for the optimization problem in next section and first introduce standard crowdsourcing concepts used throughout this paper.

- **Requester**: A requester publishes tasks, collects answers, and makes the promised reward payments. Database-wise, the *requester* is the higher-level *"executor"* as in [6] or *"task manager"* in [5]. The requester has also been called the "task-holder", "job-owner" or "builder".
- **Worker**: A worker (or crowd-worker) performs the actual human processing tasks. A worker arrives at the market in a uniformly random manner, and she immediately chooses one of the tasks to work on. The preference of task selection is based on her utility maximization principle [23]. After a period of time, the worker finalizes the task by returning the answer to the requester. Note that some research [17] reports that the worker activity on Amazon MTurk is observed to have fluctuations on both a daily and a weekly basis. However, due to the scale of data-driven microtasks, which are mainly light-weight voting, such long-term fluctuation can be ignored, provided that we use parameters that recurrent. In Section II-C we discuss the practical methodology to infer the real-time system parameters.
- **Task**: An atomic task is the most decomposed operation that a **worker** may work on. [2] suggests there are intrinsic limits of human cognitive capacity, and huge differences are observed in the demographics of crowd workers [24]. Consequently, to ensure the coherence and reliability of the human answers, a **worker** is restricted to perform a set of most basic operations like selecting from several options, ranking within a couple of objects, connecting between figures and so on. Many of these human operations can be categorized into voting, where a latent true answer needs to be obtained with some efforts (a period of time).

In the literature, tasks sometimes have been called "jobs", "HIT (Human Intelligent Tasks)" and so on. However, we reserve the word "job" for the following:

- **Job**: A job is what the requester is responsible for. A job is accomplished by invoking tasks in parallel in one or more phases, with possible additional computation performed at the requester at the beginning and end of each phase. In this paper, we will consider three different structures for tasks in phases, as we shall see below.

A. Worker Selection Model

Based on the definition of *worker* above, in a workforce market, a worker appears and starts working on a task uniformly at any time. Meanwhile, the worker's preference among the candidate tasks relies on the subjective utility measurement.

1) Worker Appearing Time: The online workers enter the crowdsourcing market in a random manner. In a short period of time, like a few hours for platforms like Amazon MTurk, the workers' arrival rate (the number of workers arriving within the unit time) can be regarded to be a constant number (according to the statistics of workers' arrival publicly released on AMT). Such property enables us to model the worker's appearing time with the following process. Denote the current workers' arriving rate with the constant number λ. For a time interval of fixed length Δt, the probability of *No worker appears* equals to $(1 - \Delta t \cdot \lambda)$. Suppose a task is submitted at time "0", and the task is accepted rightly after a worker arrives, the distribution of its acceptance can be derived as follows:

$$P(t_{acc} \leq s) = 1 - P(N(s) = 0) = 1 - (1 - \Delta t \cdot \lambda)^{\frac{s}{\Delta t}}$$

Note that t_{acc} is the time when the task is accepted and $N(s)$ denotes the number of arriving workers at time stamp s. Taking the limit of the above equation when Δt approaches zero gives the following expression:

$$P(t_{acc} \leq s) = 1 - \lim_{\Delta t \to 0} (1 - \Delta t \cdot \lambda)^{\frac{s}{\Delta t}} = 1 - e^{-\lambda s}$$

Clearly, the acceptance time of a task follows an exponential distribution on condition that the task is accepted once a worker arrives.

Recent research work [16], [17] delve into the more detailed analysis to predict appearing time of workers with more delicate consideration of time period and so on. Nevertheless, for an encapsulated computation module, a major exponential model is powerful enough for describing the latency characteristics.

2) Task Preference: In the previous discussion, we make the assumption that a task is accepted once a worker arrives. However, workers have preferences over the tasks and tend to choose the task that can maximize her benefits. In other words, a task is accepted by an appearing worker with certain probability "p". Since we have pointed out that after the submission of a task, the latency can only be adjusted through pricing strategies, therefore p is set to be a variable affected by the task's price "c" and denoted as $p(c)$. Together with the worker's arrival rate, the probability of *No worker accepts a task* is derived as: $(1 - \lambda p(c)\Delta t)$ (when $p(c) = 1$, such expression is equivalent to probability of "No worker arrives" presented in last part). Following the same procedure, the task's acceptance distribution is re-formulated as:

$$P(T < t) = 1 - e^{-\lambda_c t} = 1 - e^{-\lambda p(\mathrm{c}) t}$$

And $\lambda_o(c)$ ($\lambda_o(c) = \lambda p(\mathrm{c})$) is the joint task acceptance rate of price c. A detailed discussion of the choice model can be found in [17]. But to better estimate the latency behavior, in Section II-C, we present a real-time technique to infer parameters for tuning strategies.

B. The HPU Latency

Like in traditional CPU-based applications, when a single task is published to the HPU, there will be two phases before the answers are returned and collected: on-hold phase and processing phase. The first one is the period from the time when the task is published to the time when it is chosen by a worker; the second one is the period when the task is waiting for the response from the worker since accepted. Statistical research has been conducted on several crowdsourcing platforms to capture the traits of such latencies [16]–[18].

Definition 1 (Latency)**.** *The On-hold Latency L_o of a task (or a batch of tasks) is the clock time from when the task is published to the time when it is accepted by a worker. The Processing Latency L_p of a task (or a batch of tasks) is the clock time from when the task is accepted to the time when the answer is returned and collected by the system. The* ***Overall Latency*** *L is the sum of L_o and L_p, that is:*

$$L = L_o + L_p$$

According to the worker appearing behavior proposed previously, we can then derive that the distribution of the overall latency. Let λ_o (task acceptance rate) and λ_p (task processing rate) denote the *clock rates* in On-hold Phase and Processing Phase respectively, and the probability density functions of the latencies are as follows:

$$f_o(t) = pdf(L_o \leq t) = \lambda_o e^{-\lambda_o t}, \quad f_p(t) = pdf(L_p \leq t) = \lambda_p e^{-\lambda_p t}$$

Since the latency of On-hold Phase depends on the attractiveness of a task towards the crowds, whereas the latency of Processing Phase depends on the actual cognitive load of a task, we assume that these two phases are independent of each other, which is supported by a recent study [19]. Therefore, the probability density function for the overall latency L can be derived as following:

$$f_L(t) = pdf(L \leq t) = f_o(t) * f_p(t) = \int_0^t \lambda_o \, e^{-\lambda_o (t-u)} \lambda_p \, e^{-\lambda_p u} \, du$$
$$= \frac{\lambda_o \lambda_p}{\lambda_o - \lambda_p} (\, e^{-\lambda_p t} - \, e^{-\lambda_o t})$$

where "$*$" denotes the convolution operation of two pdfs.

1) Parallel Processing: In order to complete tasks quickly, unrelated tasks will be published simultaneously onto the crowdsourcing platforms. It is easy to understand that the latency is determined by the maximum latency of all the tasks. Given a set of k batch tasks, B_k, the cumulative distribution function (cdf) for the overall latency of parallel processing is as the following:

$$F_{para}(t) = cdf(L_{para}(B_k) \leq t) = \prod_{i=1}^{k} cdf(L(b_i) \leq t)$$

Let $f_{para}(t)$ be the probability density function (pdf) of $F_{para}(t)$. Therefore, the overall expected latency can be found as:

TABLE II: Summary of Notations

Notation	Description
t_i	an atomic task
T_N	a set of atomic tasks with size N
$L(t_i)$	latency of task t_i
K	the maximum number of possible batches
$L_o^{b_i}$	*On-hold* latency of batch b_i
$L_p^{b_i}$	*Processing* latency of batch b_i
λ	the arriving rate of the workers
λ_o	the on-hold rate or task acceptance rate
λ_p	the task processing rate
λ_o^i	the *On-hold clock rate* of batch b_i
λ_p^i	the *Processing clock rate* of batch b_i
B	the total budget
g_i	task group i, whose tasks are of i repetitions
p_i	payment for the task group i
$E(g_i)$	the expected latency for the task group i

$$E\{L\} = \int_0^{\infty} f_{para}(t) * tdt$$

Example 1. *Recall the motivating examples in the introduction, we now have the machinery in place to discuss how we obtain the latencies shown in Figure 1 and Figure 2. The expectation of the longest task for the first example is*

$$E[L] = \frac{1}{\lambda_{o,1} + \lambda_{o,2}}(1 + 2\frac{\lambda_{o,1}}{\lambda_{o,2}} + \lambda_{o,2}(1 + \frac{\lambda_{o,1}}{\lambda_{o,2}} + \frac{\lambda_{o,2}}{\lambda_{o,1}}))$$

Based on Table I, $E[case_1] = 2.93(s)$ and $E[case_2] = 2.25(s)$, where the load-sensitive strategy is better. A similar computation for the second example shows that the expected latency becomes 3.5s and 2.7s respectively.

C. The HPU Running Parameters

The workforce of crowdsourcing platform is always fluctuating in terms of demographics. However, an exponential model suffices to have a good approximation. To support a robust tuning strategy, we propose to statistically infer the parameters with the two following methodologies.

1) Parameter Inference: To infer the parameter λ_o, a "probe" program is introduced, which publishes tasks with varying prices. The workers who accept the task are simply required to make the submission as soon as possible so that the processing latency is small enough to be neglected. Due to the specific application scenario, two different inference methodologies could be adopted. Next, we demonstrate the process of how to obtain λ_o for tasks with a specific price.

Fixed Period. The probe publishes sample tasks with the same type and price. After a fixed period T_0, the number of taken tasks as N is observed.

Random Period. The probe publishes sample tasks with the same type and price at moment t_0. After N tasks have been taken (or finished), track down the length of the period T_0 starting from t_0.

For both methodologies, under maximum likelihood estimation, the parameter λ_o is given by $\hat{\lambda}_o = \frac{N}{T_0}$ for a given price. Proof of the correctness of the inference can be found in Appendix Section A. Further advanced sampling-based inference can be found in [25]. The clock rate for the processing phase λ_p is estimated in a similar manner. This time, tasks of a specific type are published and the clock rate of overall latency is estimated as: $\hat{\lambda} = \frac{N}{T_0}$. Then λ_p is estimated as: $\hat{\lambda} - \lambda_o$, where λ_o is the estimation of on-hold clock rate of this task type.

2) Linearity Hypothesis: Without loss of generality, within a certain time interval, the price c and the clock rate for the On-hold phase $\lambda_o(c)$ is observed to be linear. To provide better enhancement of the tuning strategy, we propose a Linearity Conjecture as following, which is the supporting property for strategy in Section III-B. (The concrete values of the linearity between c and $\lambda_o(c)$ does not affect the design of tuning strategy.)

Hypothesis 1 (Linearity)**.** *There exist constant values k and b, such that the rate $\lambda_o(c)$ and price c follows $\lambda_o(c) = k \cdot c + b$.*

The experiment part gives an empirical justification about this conjecture.

III. Tuning Strategies

In this section, the *H-Tuning* problem is defined in the first place. Then three tuning strategies are developed according to different scenarios.

A. Problem Definition

Definition 2 (Latency Target)**.** *The* Expectation of the Latency Target *L^* is a stochastic objective function for the tuning problem.*

The specific instantiation of L^* will be presented in each scenario.

Definition 3 (H-Tuning Problem)**.** *Given a set of atomic tasks $T = \{t_1, t_2, \ldots, t_N\}$ with size N and a discrete budget B, find an optimal budget allocation strategy so that L^* is minimized, without exceeding the budget B.*

B. Scenario I - Identical Round Homogeneous Tasks

1) Scenario Description: Scenario I is the most fundamental case. In this scenario, the system is provided with a set of identical atomic tasks (same difficulty level), and each of them is required to have the same number of running repetitions. All these atomic tasks are published simultaneously and are completed when all the repetitions are solved. A fixed budget is given at the very beginning and the system is responsible for allocating the budget to each atomic task before publishing it to the platform. The budget allocation scheme should be able to minimize the overall expected latency of all atomic tasks being solved.

2) Tuning Strategy for Scenario I: The overall latency of all tasks being solved is equivalent to the value of the maximum latency among all the tasks. Specifically, this is defined as $L^* = L(T) = \max\{L(t_i)|i = 1, 2, \ldots, N\}$. As we state earlier, the latency for each repetition is composed of two phases: the on-hold phase (Phase 1) and the processing phase (Phase 2). The latency of both phases follows an exponential distribution with parameters of λ_o and λ_p. The value of λ_o is determined by the allocated payment in a constant market condition and the value of λ_p is determined simply by the task type. Since the processing times for all homogeneous tasks are identical, the minimization of the on-hold latency leads to the minimum latency as well. Therefore, for Scenario I, the objective is indeed to find the minimization of the expected latency of the on-hold phase. In the remaining part of this section, unless otherwise specified, we use the term "expected latency" referring to the expected latency in Phase 1. Before giving the optimal solution of the budget allocation problem for Scenario I, we introduce the following Lemmas and Theorems.

Lemma 1. *Given a fixed budget of B unit payment and two identical atomic tasks t_1 and t_2, both are required to be run for one round. Allocating $\frac{B}{2}$ unit payment to both t_1 and t_2 (or if B is odd, allocating $\left\lfloor\frac{B}{2}\right\rfloor$ and $\left\lfloor\frac{B}{2}\right\rfloor + 1$ unit payment to these two atomic tasks) leads to the minimum expected latency of completing t_1 and t_2.*

Proof. Please refer to Appendix Section B1 □

Algorithm 1: Even Allocation (EA)

```
Input: budget $B$, atomic task set $T = t_1, t_2, \dots, t_N$, $m$ required
       repetition rounds
Output: allocation of payment $P = p_1, p_2, \dots, p_N$
1 if $B \leq m * N$ then
2     return the budget is not enough;
3 else
4     $\delta = \lfloor B/mN \rfloor$ and each repetition of all the atomic tasks is
      allocated with $\delta$ unit payment;
5     $\gamma = \lfloor (B\ mod\ mN)/N \rfloor$ and select $\gamma$ repetitions from each
      atomic task randomly. Increase the payment for the selected
      repetitions by one unit;
6     $\sigma = (B\ mod\ mN)\ mod\ N$ and select $\sigma$ repetitions from $\sigma$
      random atomic tasks whose payment is not increased in the
      previous step. Increase the payment of the selected repetition
      rounds by one unit;
```

Then, Lemma 2 shows that for one atomic task with multiple repetitions, allocating budget evenly to each repetition will minimize the expected latency.

Lemma 2. *For an atomic task t which needs to be run for m repetitions, and a fixed budget of B unit payment, allocating B evenly to each repetition of t leads to the minimum expected latency.*

Proof. Please refer to Appendix Section B2 □

With the above two lemmas, we have Theorem 1, which produces the budget allocation plan to minimize the expected latency.

Theorem 1. *Given two identical atomic tasks which are required to be run for the same number of repetitions and a fixed budget of B, allocating the budget evenly to each repetition of all the atomic tasks leads to the minimum expected latency.*

Proof. Please refer to Appendix Section C1. □

Theorem 1 directly leads to the optimal budget plan, whose operations are shown in Algorithm 1. As the optimal solution is obtained analytically, EA is conducted with $O(1)$ time complexity.

C. Scenario II - Multiplex Round Homogeneous Tasks

In this section, we take one more step forward: despite the identical difficulty, the tasks require different running repetitions.

1) Getting the Expected Latency: As the tasks require different number of running repetitions, the closed form of overall latency's pdf will become intractable when tasks are with large quantity. Thus, it's impossible to get the deterministic optimal solution. To address this challenge, the overall latency is processed approximately based on which the optimal budget plan is derived. Specifically, tasks are grouped if they have the same running repetitions. For example, t_1 and t_2 are atomic tasks that required to run seven times, so they are grouped together in group g = t_1, t_2. And g is supposed to repeat for seven times. In this method, the approximated overall latency is expressed by the sum of latencies of all the task groups.

Group of Single Round. Given a task group g which is composed of atomic tasks $t_1, t_2, \dots, t_n$ and required to be run for single round. According to the definition, the latency of g, which is denoted by $L(g)$, equals to $\max(L(t_1), L(t_2), \dots, L(t_n))$. Let $x_1 = \min(L(t_1), L(t_2), \dots, L(t_n))$, which means the first completion period of all the tasks within the group, and then let $x_2 = \min(\{L(t_1), L(t_2), \dots, L(t_n)\} - x_1)$ which means the time interval between the first and second completions of the atomic tasks, and then we have $x_n = \min(L(g) - \sum_{i=1}^{n-1} x_i)$ to denote the time interval between the second last and final completions of the atomic tasks. It can be derived that $L(g) = \sum_{i=1}^{i=n} x_i$. As $x_i \sim exp(\lambda_o * i)$, $L(g)$ can be regarded as the sum of n exponential variables. Therefore, $L(g) = \sum_{i=1}^{n} exp(\lambda_o * i)$.

Group of Multiple Rounds. Before we turn to the study the probabilistic model of the task group of multiple repetition rounds, the following lemma is needed to show the probabilistic property of the task which requires multiple running repetitions.

Lemma 3. *Let t denote an atomic task which needs to be run for k repetition rounds, the latency of t follows Erlang distribution of parameter k and λ_o, which is $L(t) \sim Erl\{k, \lambda_o\}$*

Proof. Please refer to Appendix Section B3. □

Now we can get the expected latency of the task group through the following deduction. Suppose we are given a task group g, which is composed of a set of tasks $\{t_1, t_2, \dots, t_n\}$ and each task is needed to be run for k repetition rounds. Let $L\{g\}$ denote the latency of the task group g, then we can have the following relationship: $L\{g\} = L\{max\{x_1, x_2, \dots, x_n\}\}$. Let $F(t)$ denote the cumulative distribution function (cdf) and $f(t)$ denote the probability density function (pdf) of the latency of the atomic tasks respectively. Let $F_g(t)$ denote the cdf and $f_g(t)$ denote the pdf of the latency of the task group respectively. The following relationship can be derived:

$$F_g(t) = F^n(t), \quad f_g(t) = n * F^{n-1}(t) * f(t)$$

With the above relationship, we get the expression of the expected latency of the task group as:

$$E\{L(g)\} = \int_0^\infty f_g(t) * tdt = \int_0^\infty n * F^{n-1}(t) * f(t) * tdt$$

According to the conclusion of $lemma$ 3, we have:

$$E\{L(g)\} = \int_0^\infty n * F_E^{n-1}(k, \lambda_o, t) * f_E(k, \lambda_o, t) * tdt$$

$F_E(k, \lambda_o, t)$ and $f_E(k, \lambda_o, t)$ denote the cdf and pdf of the Erlang distribution $Erl(k, \lambda_o)$.

2) Tuning Strategy for Scenario II: A dynamic programming based algorithm (Alg.2) is designed to solve such minimization problem. Alg.2 initializes all the prices to be 1 (line 1-2), then increases them from 1 to B' (line 5). Within each loop, the suboptimal solution is found through linear traversal of all possible pricing combinations (line 6), whose time complexity is $O(n)$. Apparently, the overall time complexity of Alg. 2 is $O(nB')$.

D. Scenario III - Multiple Round Heterogeneous Tasks

In Scenario III, the tasks are heterogeneous (in terms of difficulty) and need to be run for different numbers of repetitions. While dealing with the latency of first two scenarios, we only take the latency of Phase 1 into account. The reasons are

Algorithm 2: Repetition Algorithm (RA)

```
Input: budget $B$, task group $G = g_1, g_2, \dots, g_n$
Output: allocation of payment $P = p_1, p_2, \dots, p_n$
1  for $i = 1$ to $n$ do
2      $p_i(0) = 1$
3  $B' = B - \sum_{i=1}^{n} u_i$;
4  $E_0(0) = \sum_{i=1}^{n} E_i(P_i(0))$;
5  for $x = 1$ to $B'$ do
6      $E_0(x) = \min\{E_0(x-1),$
       $\min\{E_0(x-u_i) - [E_i(p_i) - E_i(p_i+1)] | u_i \le x\}\}$;
7      if $E_0(x-1) \le \{E_0(x-u_i) - [E_i(p_i) - E_i(p_i+1)] | u_i \le x\}$
       then
8          $\theta = \underset{i}{\operatorname{argmin}}\{E_0(x-u_i) - [E_i(p_i) - E_i(p_i+1)] | u_i \le x\}$;
9          $p_\theta(x) = p_\theta(x-1) + 1$;
10 $\forall i = 1, \dots, n, p_i = p_i(b)$
```

Algorithm 3: Heterogeneous Algorithm (HA)

```
Input: budget $B$, task group $G = g_1, g_2, \dots, g_n$
Output: allocation of payment $P = p_1, p_2, \dots, p_n$
1  for $i = 1$ to $n$ do
2      $p_i = 1$
3  $B' = B - \sum_{i=1}^{n} u_i$;
4  $CL_0 = \|OP_0 - UP\|$;
5  for $x = 1$ to $B'$ do
6      $CL_x = \min\{CL_{x-1}, \min\{CL_{x-u_i}(++p_i(x-u_i))$
7      $|u_i \le x, i = 1, \dots, n\}\}$;
8      if $CL_{x-1}$
9      $\le \{CL_{x-u_i}(++p_i(x-u_i)) | u_i \le x, i = 1, \dots, n\}$ then
10         $\theta = \underset{i}{\operatorname{argmin}}\{CL_{x-u_i}(++p_i(x-u_i)) | u_i \le x, i =$
           $1, \dots, n\}$;
11         $p_\theta(x) = p_\theta(x-1) + 1$;
12 $\forall i = 1, \dots, n, p_i = p_i(b)$
```

two-fold: the first one is that the payment does not change the latency of Phase 2, the second one is that the latency of Phase 2 is identical for all homogeneous atomic tasks. However, these properties no longer hold in Scenario III.

In this scenario, easy tasks produce smaller processing latency while harder ones will lead to longer processing latency. As a result, the previous tuning strategies do not apply well to the current problem, as the tuning result may be jeopardized by the tasks whose processing latency is significantly large. For example, if the Phase 2 latencies of some atomic tasks are extremely long that the overall latency of completing all the atomic tasks will be approximately equal to the expected latency of such atomic tasks. We call such kind of atomic tasks as "most difficult tasks". It is obvious that such type of atomic tasks generates a more critical influence to the overall latency than the others.

In order to relieve the delaying effect caused by the "most difficult tasks", we make the following adaption to the tuning strategy. In Scenario III, two objectives will be considered simultaneously: one objective is still the latency minimization of Phase 1, the other one is the latency minimization of the "most difficult task", which is equivalent to the largest expected latency of all the atomic tasks. The reason for introducing the first objective is the same as the previous scenarios, and the allocation of payment is the unique factor to change the latency of Phase 1. And the second objective serves as the penalty function to avoid the appearance of the situation where the latencies of some atomic tasks are significantly longer than those of others. One more point needs to be clarified is that we can not simply minimize the second objective, since it does not necessarily lead to the minimum latency of completing all the atomic tasks.

Formally, the objective function is defined as follows. Let $G = \{g_1, g_2, \dots, g_n\}$ denote the task group (the grouping operation is performed so that the tasks of identical type and number of repetitions fall into the same group, which is slightly different from Scenario II). Let $L^1(g_i)$ and $L^2(g_i)$ denote the latency of Phase 1 and Phase 2 of g_i respectively. Objective 1 aims at the expected latency of Phase 1 of all the atomic tasks, which is denoted by O_1 and $O_1 = E\{L^1(G)\}$. And objective 2 focuses on the sum of the expected latency of Phase 1 and Phase 2 of the most difficult atomic tasks, which is denoted by O_2 and $O_2 = max\{E\{L^1(g_i)\} + E\{L^2(g_i)\} | i = 1, \dots, n\}$.

Therefore, given the budget of B unit payment and let p_i denote the payment allocated to group g_i, the optimizing problem is defined as: $\min\{O_1, O_2\} s.t. \sum_{i=1}^{n} p_i \le B$. Here, we adopt a "compromise strategy" to solve the above two-objective optimization problem. Firstly, the "Utopia Point" (UP) is calculated, which refers to the point where both objectives are optimized independently under the given constraints. In the second place, the "Closeness" (CL) is defined as the first order distance between the objective point OP and UP. The "Closeness" is minimized under the given constrains, and the corresponding solution will serve as the optimal solution. The definition of "UP", "OP", and "CL" are formally presented as follows.

Definition 4 (Utopia Point). *Let $O_1^* = \{\min O_1 : s.t. \sum_{i=1}^{n} p_i \le B\}$ and $O_2^* = \{\min O_2 : s.t. \sum_{i=1}^{n} p_i \le B\}$. The Utopia Point is defined as $UP = (O_1^*, O_2^*)$.*

Definition 5 (Objective Point). *Let O_1 and O_2 denote the objective value of the current payment allocated to each task group. The Objective Point is the two-dimensional position determined by $\{O_1, O_2\}$.*

Definition 6 (Closeness). *The Closeness equals to the first order distance between UP and OP: $CL = \|OP - UP\|$.*

Here, the optimal budget plan is equivalent to the minimization of the following problem: $\min CL s.t. \sum_{i=1}^{n} p_i \le B$. We adopt a similar optimizing framework with Alg.2 to solve such a problem, which is shown as Alg.3. Initially, the payments are initialized to be 1 (line 1-2), and the budget is gradually increased from 1 to B' (line 5). In each loop, the suboptimal is determined through linear traversal (line 6), which results in $O(n)$ time complexity. Finally, in $O(nB')$ time complexity, the optimal solution is produced.

IV. Experiments

We extensively evaluate our model and optimization techniques, and report the results here. While the gold standard is the performance on a real platform, we can exercise greater control and thereby get a better empirical understanding of our system through simulation. We will both report the simulation results with synthetic data and results of real experiments on Amazon Mechanical Turk.

A. HPU Traits Testing with Synthetic Data

1) Experiments Settings: We conduct six sets of experiments for each scenario. The first four sets are based on linear models, which aim to verify the effectiveness of the tuning strategy under the Linear Hypothesis, and the last two sets target at testing the robustness of the tuning strategy based on nonlinear models. For the linear model based experiment, the model parameters are set as $\lambda_o = p + 1$, $\lambda_o = 10p + 1$, $\lambda_o = 0.1p + 10$, $\lambda_o = 3p + 3$. For the nonlinear models, the parameters are set as $\lambda_o = 1 + p^2$ and $\lambda_o = \log(1 + p)$. The total number of tasks is set to be 100 and same in every set of experiment. The tasks are uniformly distributed in each set of the experiment and the budget varies from 1000 to 5000. For each different budget value, we conduct all sets of experiments for each scenario and note down the latency.

Identical Round Homogeneous Tasks. All tasks call for 5 repetitions. As the difficulties of the tasks are identical, the clock rate λ_p for the processing latency is uniformly set to be 2.0. Since the optimistic solution is produced by the *even allocation* (*algorithm 1*), biased allocation strategies are adopted as the baseline comparison. Instead of allocating the budget evenly, the biased method gives more payment to one half of the tasks, while less payment is given to the remaining tasks. Specifically, half of the tasks are randomly selected as "the prior group" which take up α ($\frac{1}{2} < \alpha < 1$) of the total budget ($\alpha = \frac{1}{2}$ leads to the even allocation), and the remaining tasks get the $1 - \alpha$ of the total budget. The values of α is randomly set to be 0.67 and 0.75, which serve as two baseline comparisons with the optimal solution in Scenario I.

Multiplex Round Homogeneous Tasks. The tasks are divided equally into two groups: one group has 3 repetitions for each task, the other group has 5 repetitions for each task. λ_p is uniformly set to be 2.0 due to the identical setting of the difficulty. Two baseline methods are chosen for the comparison. The first method is called *task-even* allocation, which gives an identical price to each task, then every repetition of a task is evenly allocated. Therefore, the price of each repetition in group two is 60% of that in group one. The second one is called *rep-even* allocation, which gives an identical price to each repetition of all the tasks. So the total price for tasks in group one is 60% of that in group two.

Multiple Round Heterogeneous Tasks. The tasks are divided into two groups: tasks in the first group call for 3 repetitions, while tasks in the second group call for 5 repetitions. Then λ_p is set to be 2.0 and 3.0 to incur different processing rates in these two groups respectively. Same as Scenario II, the *rep-even* and *task-even* are chosen as the baseline methods for the comparison.

Apart from the above setting, we also conduct experiments with different settings of the budget, the number of tasks, the number of repetitions, and difficulty levels. However, there's no significant variance between these settings with the above one. Therefore, we simply demonstrate the results of the above setting for further analysis.

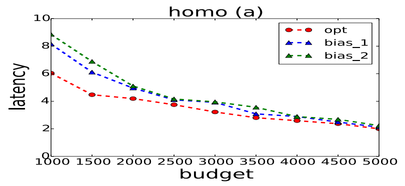
2) Results Summary: From the experiment results, the optimal solution outperforms the comparisons in terms of latency in every case. For results of Scenario I, the "*bias_1*" produces slightly better performance than "*bias_2*". This is because *bias_2* is more biased than *bias_1* since the value of α is larger. Such phenomenon further verifies our conclusion that even allocation leads to the optimal budget plan for Scenario I. Besides, we can find that although the optimal solution of Scenario I is designed based on the linear hypothesis, it still works for the nonlinear cases (*homo(e)* and *homo(f)*), which can be partially explained by the varying range of the payment: the task price varies from 1 to 9. For such relatively low prices, the non-linear relationship can be linearly approximated quite well.

We can further discover that the optimal results are relatively close to the comparisons in case (b) and (c), for all the scenarios. For case (b) ($\lambda_o = 10p + 1$), such phenomenon can be caused by the large value of the linear coefficient (p has 10 as the coefficient). When the linear coefficient is large, the on-hold clock rate is sensitive to the change of price. When the price grows, the task acceptance rate increases much faster, making the on-hold latency decrease to a low level. In this situation, the overall latency will be mostly determined by the processing phase. Similar phenomenon can be observed for case (e), (k) and (q) in each of the scenarios, where at the beginning the overall latency reduces sharply from the initial low price since the task acceptance rate is growing quadratically rather than linearly. After that, the latency will get to a stable level when the price goes higher. Note that case (c) (also with case (i) and (o)) is another extreme, where λ_o is fairly insensitive to the change of the price. In this situation, the latency is mostly determined by the initial setting of on-hold and processing phases, and the price has a limited influence to change it.

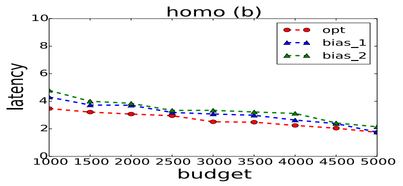
Finally, we can summarize the findings of the synthetic experiment as follows: 1) the optimal tuning strategy is robust to non-linearity. The unit price for each task is usually small, therefore the linearity hypothesis holds for normal cases. 2) The optimal tuning strategy is sensitive to the *price-λ_o* relationship: when λ_o is sensitive to the price changes, the on-hold latency drops sharply with the growing price. Then the overall latency is determined by the processing time and it is not necessary to keep increasing the price.

B. Tuning Tasks on Amazon MTurk

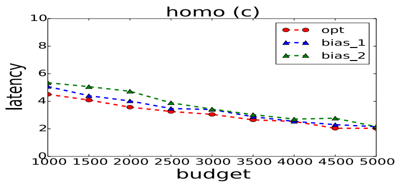
1) Experiments Settings: We create a set of image filtering tasks as the atomic tasks: we first present the workers with an image that has the exact number of the dots on it, then a set of images are presented to the workers and they are required to estimate the number of dots in each image. Based on the estimation, workers are expected to filter out the ones who have dots less than a given threshold. Under such setting, the cognitive abilities of "recognizing" and "counting" are utilized, and the task is finished by giving a binary voting



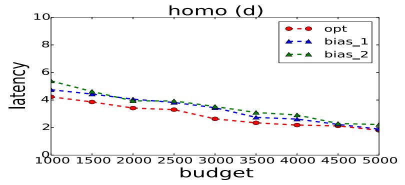
(a) Homogeneous ($\lambda_o = 1 + p$)



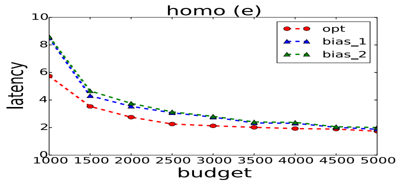
(b) Homogeneous ($\lambda_o = 10p + 1$)



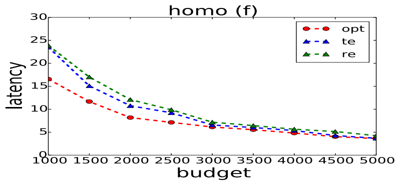
(c) Homogeneous ($\lambda_o = 0.1p + 10$)



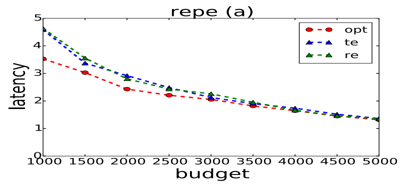
(d) Homogeneous ($\lambda_o = 3p + 3$)



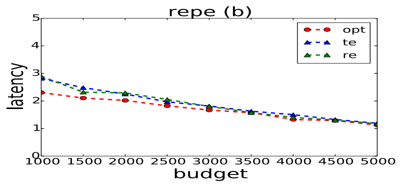
(e) Homogeneous ($\lambda_o = 1 + p^2$)



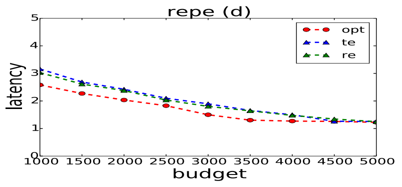
(f) Homogeneous ($\lambda_o = \log{(1 + p)}$)



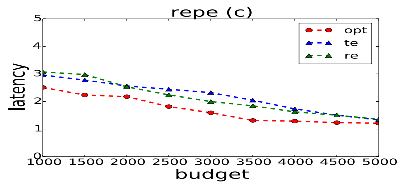
(g) Repetition ($\lambda_o = 1 + p$)



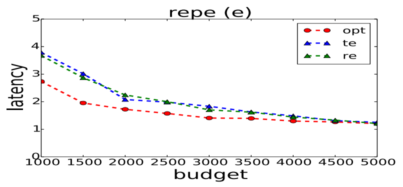
(h) Repetition ($\lambda_o = 10p + 1$)



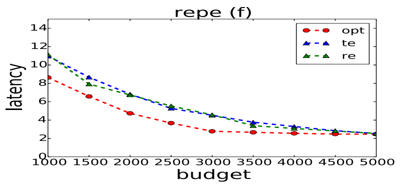
(i) Repetition ($\lambda_o = 0.1p + 10$)



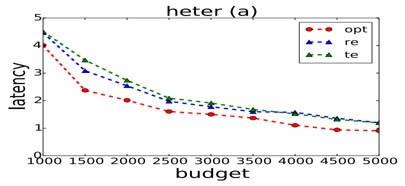
(j) Repetition ($\lambda_o = 3p + 3$)



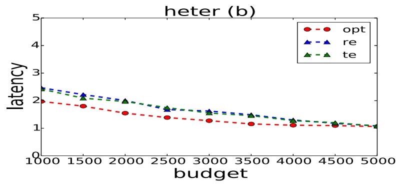
(k) Repetition ($\lambda_o = 1 + p^2$)



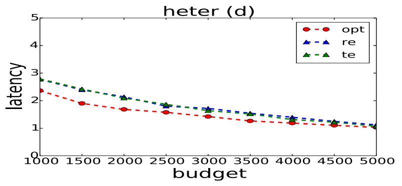
(l) Repetition ($\lambda_o = \log{(1 + p)}$)



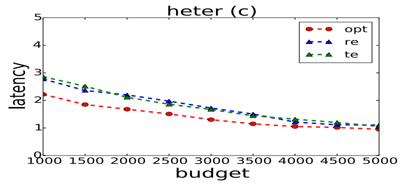
(m) Heterogeneous ($\lambda_o = 1 + p$)



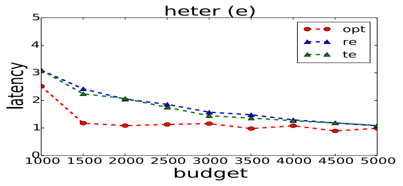
(n) Heterogeneous ($\lambda_o = 10p + 1$)



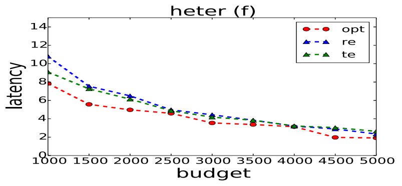
(o) Heterogeneous ($\lambda_o = 0.1p + 10$)



(p) Heterogeneous ($\lambda_o = 3p + 3$)



(q) Heterogeneous ($\lambda_o = 1 + p^2$)



(r) Heterogeneous ($\lambda_o = \log{(1 + p)}$)

Fig. 3: Experiments on Synthetic Data

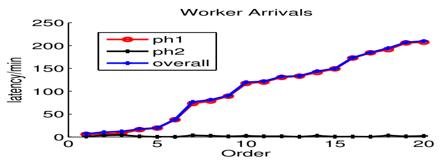


Fig. 4: Worker Arrival Moments

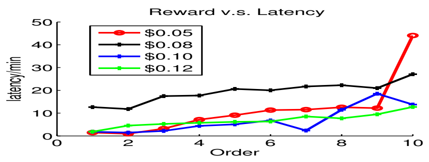


Fig. 5: Money v.s. Latency

(clicking on the checkbox). In addition, the workers receive their rewards when the provided answers are correct. We control the difficulty or type of tasks by varying the number of images given in a single task.

Our work focuses on tuning the budget allocation and real-time latency, thus we purposely design the experiment simple enough and avoid setting any worker qualifications and inter-rater agreement. In fact, in real scenarios, the atomic tasks at lowest levels are just the same as the experimental tasks: comparing items, screening out candidates and simple ranking.

2) Results Summary: Firstly, in Fig 4 we present the general behavior of the worker appearance and the latency of processing time. We issue image filtering tasks and each task is assigned with 1-unit reward ($0.05), and collect the first 20 arrivals. As shown in Fig 4, the arrival epochs of the workers exhibit linearity, indicating the suitability of the Poisson Process Model, while the latency of the second phase fluctuates in a small range. Then we examine the effect of varying the rewards: we vary the reward of a single task from 0.05 to 0.12, while for each task we require 10 repetitions. The results can be found in Fig 5, where it is obvious that the increase in rewards incurs shorter latencies. According to the methodologies introduced in Section II-C, we obtain the corresponding parameters (s^{-1}), $\lambda_{o,1} = 0.0038, \lambda_{o,2} = 0.0062, \lambda_{o,3} = 0.0121, \lambda_{o,4} = 0.0131$, which supports the Linearity Hypothesis proposed in Section II-C2.

In the end, we present the results of examining the effect of varying the task types: we vary the internal binary voting number (the number of images in one task) from 4 to 8 in order to control the task difficulty. Such a change of difficulty results in the declining worker acceptance rate (see Fig 6(a)), and the increase of the average processing time, which is shown in Fig 6(b). We then evaluate our proposed algorithms on Amazon MTurk, especially under Scenario III. Namely, 3 types of tasks are published with different repetition requirements: 10 for $t1$, 15 for $t2$ and 20 for $t3$. The total budgets are also varying from $6 to $10. For each budget, we note down the latency of all task completions. We compare our algorithms (OPT) with the heuristic where each type receives the same payment. Results can be found in Fig 6(c), where the lower latency of OPT shows the effectiveness of our algorithms. Note that for each budget, the OPT successfully avoids yielding the longest latency among the three tasks.

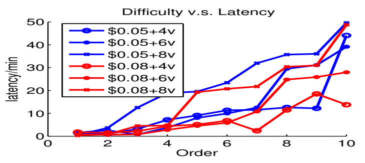
V. Related Work

Leveraging the HPU to gain a better performance has been an attractive topic ever since the emergence of crowdsourcing applications. Many recent works have studied various optimization issues associated with the HPU [26], [27]. Most of them focus on the quality issues in terms of answer confidence [28], [29], while some other efforts concentrating on the optimization of monetary cost [17].

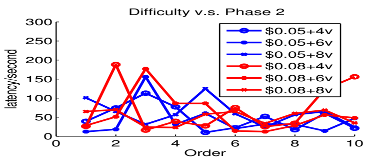
However, since tasks are usually required to be accomplished before a deadline designated by the requesters, speed or latency is always regarded as one of the most critical concerns among all the various properties. Based on the latency control, there have been several major types of existing methodologies [30]. The first approach includes certain latency models that provide feasible solutions to simplify the dynamics among the whole task pool. For example, in [31], [32], a task campaign is treated to propagate in multiple rounds of tasks and each round spends the same amount of time. Then the final completion time can be measured as the overall number of rounds. Furthermore, [31], [32] reduce the latency with the favor of answer deduction method. On the other hand, [17], [19] build probability models to predict the worker arrival rate and infer the expected latency. Except for latency modeling, [17], [22] observe a tradeoff between payment and completion time, and they discover that incentive adaptation can be utilized to optimize the completion time in order to meet the deadline.

Most of the current literature aims at reducing the number of issued queries to the crowd [2], [6], [7], [9], [12], [14], [15], which in return shortens the answering period. In crowdsourcing platforms, workers are usually fluctuant in the task pool and not necessarily bound to finish the assigned sequence of tasks. To simulate this real world circumstance, it is essential to consider the HPU as the smallest hardware unit based on a low-level clock-rate model instead of the high-level number of queries. Unfortunately, the stochastic human behavior makes this model rather intractable. Several applications try to optimize the HPU's performance on a real-time basis in order to finish tasks before a preset deadline [19]–[21]. But their approaches are highly dependent on the applications and thus hard to adapt to a general framework; in addition, the "deadline" semantic does not support the batch processing scenario where a general HPU usually meets.

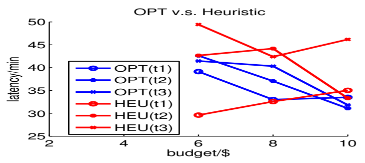
There is another practical methodology developed by recruiting a set of prepaid workers so that they can wait online and process the task immediately when they are published. In the work of [33], the authors propose such a prepaid model to instantiate a real-time respond crowdsourcing interface, and a Retailer Model is adopted to describe the behaviors of prepaid workers [34]. Following the Retailer Model, one analytic effort on optimally organizing the microtasks can be



(a) Difficulty v.s. Phase 1



(b) Difficulty v.s. Phase 2



(c) OPT v.s. Heuristic

Fig. 6: Experiments Results II

found in [35]. Note that the prepaid implementation differs greatly from our work. The tasks for prepaid implementation entails high instantaneity, where the tasks are expected to be submitted in several seconds, contributing to a relatively higher payment. However, the HPU tuning assumes a system-level perspective, where the latency of the task set varies over a larger range according to the specific requirement of the database users. Besides, the *Queuing Theory* based model of prepaid implementation cannot be tailored into the HPU scenarios easily.

This work is most similar to [22], where the problem of minimizing crowdsourcing latency is formulated into two optimization issues: *1. minimizing the completion cost of all the tasks given a deterministic deadline of every task*, and *2. minimizing the latency under a budget constraint*. Our work is fundamentally different from [22] in terms of the way the tasks' prices are set. In [22], the tasks' prices are determined dynamically, *i.e.* the system constantly adapts the prices. However, such a dynamic manner is both technically challenging and morally impractical in real crowdsourcing platforms. First of all, for the current crowdsourcing platforms, like Amazon MTurk, the tasks' prices have to be determined before they are published and remained fixed until they are solved. Even if the tasks' prices could be changed, such a dynamic strategy will incur significant computational and I/O burden. Besides, once a worker picks a task and starts solving it, the task's price should remain fixed until she submits her answer and gets paid. By initially setting the tasks' prices in an optimal and static way, our work is free of all the aforementioned limitations.

VI. Conclusion

In this paper, we address the problem of tuning the modularized human computation, so that the latency in real clock time could be minimized. The difficulty of such problem arises in the stochastic behavior of the latency of the HPU. To address this challenge, we theoretically and practically propose that appearance of the crowd "workers" follows a Poisson Process, whose parameter differs at different budget levels and types of atomic tasks. Then we formally propose the *H-Tuning Problem* to optimize the expected latency of the longest tasks. Moreover, under three most general scenarios on crowd-powered applications, advanced strategies are designed to cope with the *H-Tuning Problem*. Finally, a series of experiments conducted on both simulated data and real commercial platform reveals the effectiveness of the proposed models and strategies. To conclude, the crowdsourced human computation is now equipped with primitive tuning ability in terms of running time.

Acknowledgment

The work is supported partially by Hong Kong RGC Project N HKUST637/13, NSFC Guang Dong Grant No. U1301253, NSFC 61232018, NSFC Grant No. 61328202, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, Microsoft Research Asia Gift Grant, and NSF under Grant IIS-1250880.

References

[1] E. Adar, "Why i hate Mechanical Turk research (and workshops)," in *CHI 2011*.
[2] *Human Computation*.
[3] D. A. Grier, "The math tables project of the work projects administration: The reluctant start of the computing era," *IEEE Ann. Hist. Comput.*
[4] A. Parameswaran and N. Polyzotis, "Answering queries using humans, algorithms and databases," in *CIDR 2011*.
[5] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: answering queries with crowdsourcing," in *SIGMOD 2011*.
[6] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *VLDB 2011*.
[7] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: algorithms for filtering data with humans," in *SIGMOD 2012*.
[8] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments."
[9] S. Guo, A. G. Parameswaran, and H. Garcia-Molina, "So who won?: dynamic max discovery with the crowd," in *SIGMOD Conference 2012*.
[10] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *ICDT 2013*.
[11] X. S. Yang, R. Cheng, L. Mo, B. Kao, and D. W. Cheung, "On incentive-based tagging," in *ICDE 2013*.
[12] J. C. Zhang, L. Chen, H. V. Jagadish, and C. C. CAO, "Reducing uncertainty of schema matching via crowdsourcing," *VLDB 2013*.
[13] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *SIGMOD 2013*.
[14] I. Lotosh, T. Milo, and S. Novgorodov, "Crowdplanr: Planning made easy with crowd," in *ICDE 2013*.
[15] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart, "Crowd mining," in *SIGMOD 2013*.
[16] J. Wang, S. Faridani, and P. Ipeirotis, "Estimating the completion time of crowdsourced tasks using survival analysis models," *CSDM 2011*.
[17] S. Faridani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time." in *Hcomp 2011*.
[18] W. Mason and D. J. Watts, "Financial incentives and the "performance of crowds"," ser. HCOMP 2009.
[19] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones," in *MobiSys 2010*.
[20] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh, "Vizwiz: nearly real-time answers to visual questions," in *UIST 2010*.
[21] G. Pickard, W. Pan, I. Rahwan, M. Cebrian, R. Crane, A. Madan, and A. Pentland, "Time-critical social mobilization," *Science*, 2011.

[22] Y. Gao and A. Parameswaran, "Finish them!: Pricing algorithms for human computation," in *VLDB 2014*.

[23] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 173–184.

[24] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers?: shifting demographics in mechanical turk," in *CHI 2010*.

[25] I. Basawa and B. Rao, *Statistical inference for stochastic processes*, ser. Probability and mathematical statistics. Academic Press, 1980.

[26] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "Cdas: a crowdsourcing data analytics system," *VLDB 2012*.

[27] C. C. CAO, J. She, Y. Tong, and L. Chen, "Whom to ask? jury selection for decision making tasks on micro-blog services," *VLDB 2012*.

[28] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*.

[29] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan, "Asking the right questions in crowd data sourcing," in *ICDE 2012*.

[30] G. Li, J. Wang, Y. Zheng, and M. Franklin, "Crowdsourced data management: A survey," 2016.

[31] A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy, "Crowd-powered find algorithms," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 964–975.

[32] V. Verroios, P. Lofgren, and H. Garcia-Molina, "tdp: An optimal-latency budget allocation strategy for crowdsourced maximum operations," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1047–1062.

[33] M. S. Bernstein, J. Brandt, R. C. Miller, and D. R. Karger, "Crowds in two seconds: Enabling realtime crowd-powered interfaces," in *UIST 2011*.

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

[35] P. Minder, S. Seuken, A. Bernstein, and M. Zollinger, "Crowdmanager - combinatorial allocation and pricing of crowdsourcing tasks with time constraints," in *ACM-EC 2012*.

Appendix

A. Inference Of Parameter λ

Fixed Period. The likelihood function of parameter λ is $L = \lambda_N \exp[-\lambda \sum_{k=1}^{N} T_k] \exp[-\lambda(T_0 - t_n)] = \lambda_N \exp[-\lambda T_0]$. To maximize the likelihood, the ML estimation of λ is derived as $\hat{\lambda} = N/T_0$, which is unbiased according to Rao-Blackwell Theorem.

Random Period. Suppose each worker appears at the epochs $0 < t_1 < t_2 \ldots < t_N$, we could obtain the likelihood function: $L = \lambda^N e^{-\lambda T_0}$. Thus to maximize the likelihood, the ML estimation of λ is given by $\hat{\lambda} = \frac{N}{T_0}$ (same as the one in Fixed Period). To remove the bias, the parameter can be further updated: $\tilde{\lambda} = ((N-1)N)\hat{\lambda}$.

B. Proof of Lemma

1) Proof of Lemma 1: As illustrated in the previous section, Phase 1 of both t_1 and t_2 follow exponential distributions, whose parameters are denoted as $\lambda^o_{t_1}$ and $\lambda^o_{t_2}$. And according to Hypothesis I in Section II-C, when allocating t_1 with x unit payment and t_2 with $B - x$ unit payment, $\lambda^o_{t_1} = kx$ and $\lambda^o_{t_2} = k(B - x)$ (k is the constant coefficient). With the two parameters defined above, we can derive the following relationship: $E(\{t_1, t_2\}) = E\{\max\{L^o(t_1), L^o(t_2)\}\} = \frac{\lambda^o_{t_1}+\lambda^o_{t_2}}{\lambda^o_{t_1}\lambda^o_{t_2}} - \frac{1}{\lambda^o_{t_1}+\lambda^o_{t_2}}$. As $\lambda^o_{t_1} = kx$ and $\lambda^o_{t_2} = k(B-x)$, $E_(t_1, t_2)$ turns out to be a convex function and reaches its minimum point when $x = \frac{B}{2}$ (or $\lfloor B/2 \rfloor$ if B is odd). Hence, Lemma 1 is proved.

2) Proof of Lemma 2: Let $\{p_1, \ldots, p_m\}$ denote the payment allocated to each repetition of an atomic task t, and $\{\lambda^o_1, \ldots, \lambda^o_m\}$ denote the exponential parameter of each repetition. It is obvious to see that $\sum_{i=1}^{k} p_i = B$, and based on Hypothesis 1, $\lambda^o_i = k * p_i$. With the above information, we can derive the expected latency of t as follows. $E\{L(t)\} = \sum_{i=1}^{m} 1/\lambda^o_i = \sum_{i=1}^{m} 1/kp_i$. Since $E\{L(t)\} = \sum_{i=1}^{m} 1/kp_i \leq B^2/km$, and the equality is established iff. $\forall i \in \{1, \ldots, m\}$, $x_i = B/m$. Therefore, we come to the conclusion that allocating the budget evenly to each repetition of the atomic task leads to the minimum expected latency.

3) Proof of Lemma 3: Let r_i denote the ith repetition of task x. According to Lemma 1, $\forall r_i \in \{r_1, \ldots, r_n\}$, $L(r_i)$ follows an exponential distribution of the same parameter λ. So $L(x) = L(\sum_{i=1}^{k} r_i)$. This meets the requirement of Erlang distribution and makes $L(x) \sim Erl(k, \lambda)$.

C. Proof of Theorem

1) Proof of Theorem 1: This theorem is proved by mathematical induction. Firstly, Lemma 1 shows that the theorem holds when both atomic tasks are run for exactly once. Then, we prove that the theorem still holds when the number of *repetitions* (reps) is increased to $n + 1$ on condition that it holds when the number of reps equals to n. Suppose we have two identical tasks t_1 and t_2, which require n number of reps. Now we allocate each *repetition* (rep) with x unit payment. Based on our presumption, this budget allocation leads to the minimum expected latency. Let H denote the completion of task t_1, and the completion of both tasks as $\max\{H, H\}$. Now we increase the number of reps of both tasks to $n + 1$, and the budget to $2(n + 1)x$. Suppose we have a better budget allocation which outperforms the even allocation. It is trivial to see that one task will be allocated with more payment and the other atomic task will be allocated with less payment. At the same time, the payment for each rep of the same atomic task remains identical. Let I denote the completion of the first n reps of t_1 and i denote the completion of the last rep of t_1. So the completion of t_1 is denoted as $\{I + i\}$. Let J denote the completion of the first n reps of t_2 and j denote the completion of the last rep of t_2. The completion of t_2 can be denoted as $J + j$. Similarly, when allocating the budget evenly to each rep of both tasks, we use H to denote the first n reps of the atomic task and h to denote the last rep of the atomic task, and the completion of t_1 (or t_2) is denoted as $\{H + h\}$. Then, the completion of both tasks with the assumed optimal budget allocation is denoted by $\max\{\{I + i\}, \{J + j\}\}$ and the completion of both tasks with the evenly allocated budget is denoted as $\max\{\{H + h\}, \{H + h\}\}$. Here, we will have the flowing relationship: $E\{\max\{\{I + i\}, \{J + j\}\} = (E\{I + i\} + E\{J + j\}) - E\{\min\{\{I + i\}, \{J + j\}\}\}$, and $E\{\max\{\{H + h\}, \{H + h\}\}\} = (E\{H+h\} + E\{H+h\}) - E\{\min\{\{H + h\}, \{H + h\}\}\}$. As $E\{I + i\} + E\{J + j\} = n/(k(x - \varepsilon)) + n/(k(x + \varepsilon)) \leq 2n/kx = E\{H + h\} + E\{H + h\}$, then we have $E\{\max\{\{I + i\}, \{J + j\}\}\} \geq E\{\max\{\{H + h\}, \{H + h\}\}\}$. This shows that the theorem still holds when the number of reps increases to $n + 1$, which proves the theorem to be true.