On task assignment for real-time reliable crowdsourcing

Ioannis Boutsis, Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business, Athens, Greece
{mpoutsis, vana}@aueb.gr

***Abstract*—With the rapid growth of mobile smartphone users, several commercial mobile companies have exploited crowdsourcing as an effective approach to collect and analyze data, to improve their services. In a crowdsourcing system, "human workers" are enlisted to perform small tasks, that are difficult to be automated, in return for some monetary compensation. This paper presents our crowdsourcing system that seeks to address the challenge of determining the most efficient allocation of tasks to the human crowd. The goal of our algorithm is to efficiently determine the most appropriate set of workers to assign to each incoming task, so that the real-time demands are met and high quality results are returned. We empirically evaluate our approach and show that our system effectively meets the requested demands, has low overhead and can improve the number of tasks processed under the defined constraints over 71% compared to traditional approaches.**

***Keywords*—distributed systems; crowdsourcing; real-time**

I. Introduction

Crowdsourcing has emerged as an attractive paradigm in recent years that can leverage the collective intelligence of a large crowd of human workers quickly and inexpensively, to accomplish computational tasks. Several prominent companies, recognizing the opportunities presented in such mass collaboration systems developed their own crowdsourcing marketplaces, including Amazon's Mechanical Turk (AMT) [1], CrowdFlower [2], microWorkers [3], etc. Typical crowdsourcing platforms constitute marketplaces for tasks, where *Requesters* define tasks, and a diverse pool of humans workers, referred to as *Crowdworkers*, with different expertise level, execute them in exchange for a monetary or other reward.

Crowdsourcing can provide great benefit in cases where tasks cannot be fully automated, or to achieve a better understanding of a situation. For example, as shown in [4], crowdsourcing can play critical role in emergency situations, both during and after the event, collecting information from emergency scenes and visualizing the data to generate community emergency maps. Crowdsourcing allows capable crowds to participate in various tasks, from simply providing pictures and status reports, to "validating" pieces of information, or more complicated editing and management tasks. Such automated tools can offer a common disaster view and help organizations ascertain the current situation, complementing the information available through fixed, static infrastructures. Other examples include automating stakeholder analysis[5], improving the search engines results[6], detecting service-level network events[7], and identifying transportation events of interest [8].

Crowdsourcing systems introduce multiple challenges. First, current systems attract a fluid, diverse pool of human workers, (*i.e.,* spammers), which often submit arbitrary answers regardless of the task to collect the respective fee [9], [10]. Spammers act as opportunistic workers which submit random answers, often without looking at the tasks, or by creating automated bots pretending to be human workers to complete the tasks. At another extreme, spammers can be malicious, working individually or in groups, focusing on the same groups of tasks, to spread rumors, create fake responses or influence the markets. One common approach to deal with this problem is to introduce redundancy [11], [12], *i.e.,* ask multiple workers to execute the same task and use a technique such as majority voting to determine the correct answer. Statistically, more workers working on a task give a higher-quality result since the impact of wrong answers is reduced by the correct ones [13].

A second challenge is that, often, the execution of the crowdsourcing tasks may be time-critical. For example, real-time tasks such as emergency response, are becoming more commonplace; these have important real-time response requirements within which the results should be received. Real-time execution is a challenging problem in these settings, due to the highly dynamic and transient crowd, as well as resource constraints and fluctuations in resource constrained environments, such as mobile settings where the workers can reside. Thus, to achieve real-time response, the workers have to be explored both for the quality of their results, and their ability to deliver the results within the required time constraints.

The final challenge comes from the application's budget constraint. The monetary compensation in the majority of today's crowdsourcing systems is given in an automated way and only a few requesters of the tasks provide feedback for the individual workers. However, based on their background (*i.e.*, expertise, reliability, performance, location), some workers are more suitable for some tasks than others, so we need to ensure that the tasks will be executed by the most suitable workers. This can lead to a reduction of the requester's monetary cost, since a small amount of suitable workers provides results with higher confidence, compared to a group that potentially consists of both suitable and unsuitable workers but with a lot of noise, due to the unsuitable workers, that would inevitably lead to a larger group (and respective payments) to increase the confidence for the result.

Traditional crowdsourcing architectures, such as AMT, do not provide enough flexibility as they depend on the willingness of the users to process the tasks. These are further limited

as: (i) they are not designed to optimize the task assignment process and (ii) they do not provide support for meeting real-time response demands. A few recent works have looked at the problem of task assignment in crowdsourcing environments. These papers focus on the accuracy of the results [14], [15], minimizing the error, subject to a budget constraint [16], [9] or minimizing the amount of task assignments to achieve a target reliability [17], [18]. However, they do not consider the time needed to retrieve the results. On the other hand, works that take into account response times [19], assume a generic distribution for the individual workers execution time, and do not consider other constraints. Finally, the problem differs from the traditional task assignment problem in distributed systems as: (i) crowdsourcing systems assign a variable workload of tasks to a set of humans with different skills and characteristics, that might be unknown, (ii) the workers cannot be expected to be persistent or willing to execute subsequent tasks, and, (iii) the human factor makes the execution time unpredictable.

In this paper we present CRITICAl (Crowdsourcing under RelIability and TIme ConstrAints), our system that aims to exploit the wisdom of the crowd to assign tasks to human crowd workers under both reliability and real-time constraints. Thus, our objective is to solve the task assignment problem with respect to performance, reliability, and cost. CRITICAl combines crowdsourcing with task assignment techniques to provide high-quality results and satisfy real-time response requirements of the tasks. We summarize our contributions:

- We propose CRITICAl, our system that determines the most appropriate groups of human workers to assign a set of tasks, so that high-quality results are returned while meeting task real-time constraints. Our system computes the reliability of a group of workers and estimates the probability that the group of workers will execute the task within the deadline in order to perform dynamic assignment of the crowdsourcing tasks into groups of workers based on the characteristics of the tasks and the worker profiles.
- We show that the task assignment problem is NP-hard and then provide a polynomial online algorithm that effectively reduces the search space by several orders of magnitude. Our approach systematically investigates the search space for Pareto optimal solutions, with respect to the objectives.
- We carry out a case study on CrowdFlower [2], a popular crowdsourcing system, to extract the behavior of the human workers for real world tasks, and use these findings in our experimental evaluation.
- We perform extensive experiments to evaluate our approach and show that CRITICAl effectively meets the requested demands, has low overhead and can achieve over 71% improvement on the number of tasks processed under the defined constraints, compared to traditional approaches such as AMT.

II. System Model

Worker Model. The crowdsourcing system comprises a set of human crowd workers, denoted as $w_i \in W$, that register to CRITICAl to receive tasks, defined by the task Requesters. Each worker w_i is associated with a set of attributes, denoted as $\langle id_i,\ lat_i,\ long_i,\ rel_i,\ completed\_tasks_i[], preferences_i[]\rangle$, where id_i represents the worker's unique identifier, the user's geographical coordinates are represented in $(lat_i, long_i)$, and statistics about each completed task are maintained in $completed\_tasks_i[]$. We maintain the current reliability for w_i, denoted as rel_i, that represents the probability that the worker will provide a correct answer for a task. This metric is dynamically computed, and updated based on the answers provided by w_i (explained in Section IV.A). To assist in the selection of tasks, some crowdsourcing systems allow workers to specify a list of $preferences_i[]$ in the execution of tasks, based on the workers' interests and capabilities, such as desired task categories and a desired monetary compensation for the tasks. These assist the system in selecting workers for tasks, for example, it might be necessary to assign skilled workers to tasks to produce high confidence results.

Task Model. Each $task_j \in T$ is associated with a set of attributes: $\langle id_j, lat_j, long_j, description_j, deadline_j, reward_j, category_j, amount_j\rangle$, where id_j represents the unique identifier for $task_j$, $(lat_j, long_j)$ represent the geographical coordinates of the location where $task_j$ refers to, and $description_j$, is the text description of the task along with the possible choices. For instance a $description_j$ might be: *"Is there traffic congestion in your geographical location? (Yes/No)"*, etc., $reward_j$ refers to the monetary reward received by the worker when he/she completes $task_j$, $category_j$ denotes the category that the task belongs to, and $amount_j$ represents the amount of workers requested to process the task.

Each $task_j$ is characterized by a time constraint $deadline_j$, specified by the Requester, within which the task needs to complete. This is an absolute value. If a task does not complete within a deadline, then we assume the worker failed to meet the task requirement. When a $task_j$ is assigned to the system, we denote as ttd_{ij} the time-to-deadline, thus the time interval from the assignment of $task_j$ to w_i until the $deadline_j$ expires. As the task executes, the ttd_{ij} is updated dynamically based on the time elapsed. Also, we denote $exec_{ij}$ as the total execution time from the assignment of the $task_j$ to w_i until its completion. Each worker w_i provides a $response_{ij}$ for each $task_j$ it executes. We consider as valid responses the ones received within the task's $deadline_j$. From the list of the worker responses we can identify the correct answer for the task. In particular, in CRITICAl we consider tasks that can be binary or multiple selection tasks, with a list of possible answers as a finite set. We extract the final answer for a task using the Majority voting technique where we determine the correct answer as a product of the Majority voting of each $response_{ij}$ received before the $deadline_j$ expires and denoted as $response_j$. However, metrics other than Majority voting could also be incorporated (discussed in Section IV-A).

Discussion. Similar to the commercial crowdsourcing systems, we denote the $amount_j$ for the workers that process a task. Intuitively, the more the workers that process a task, the higher the reliability of the outcome[13], and thus, the $amount_j$ is only constrained by the payment that these workers receive. As we show in the experiments, selecting small groups of reliable workers can perform better, since assigning additional reliable workers slightly improves the task's aggregated reliability, but sabotages subsequent tasks.

A. Bipartite Graph

We use a bipartite graph to represent all the possible assignments between the workers and the tasks, similar to [18], [20]. In a bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ the vertices can be divided into two disjoint sets $\mathcal{U}$ and $\mathcal{V}$ such that each edge $e_{ij} = (u_i, v_j) \in \mathcal{E}$ connects a vertex $u_i \in \mathcal{U}$ with one $v_j \in \mathcal{V}$. Each vertex in $\mathcal{U}$ represents a worker w_i, who is available to execute tasks. Vertices in $\mathcal{V}$ represent all tasks $task_j$ to be processed by workers. The respective edges $e_{ij} = (w_i, task_j) \in \mathcal{E}$, represent a possible assignment between a task and a worker. When a $task_j$ is matched to several workers, then the graph contains multiple edges for $task_j$; each edge corresponds to a different worker. We denote as $group_j = \{u_g | \forall e_{gj} \in \mathcal{E}\}$, all the workers which are assigned for $task_j$, defined from the set of the edges.

Graph Construction. In the crowdsourcing systems for the location-dependent tasks that we consider, the environment can be very dynamic with frequent worker insertions and departures and dynamic injection of tasks, causing changes to the respective graph. Thus, the bipartite graph needs to be constructed and maintained in real-time. The CRITICAl system needs to keep track only those tasks that involve its geographical region and the workers that currently remain in that region, since we target on geo-located tasks, so that tasks will be processed by workers that are spatially close to the location defined by the task. Hence, each worker is registered to the CRITICAl server in the geographical area in which the worker belongs based on his geographical location, and each task is registered to the respective server based on the task's lat_j and $long_j$. To provide a scalable solution we perform a spatial decomposition of the geographic area into a number of non-overlapping regions, similar to [21], where each region is assigned to a CRITICAl server who is responsible for the region; this allows our task assignment algorithm to perform in a scalable manner.

III. Problem Definition

Assume $\mathcal{U}$ the set of workers and $\mathcal{V}$ the set of tasks to be allocated to the workers. Our goal is to determine the appropriate group of workers $group_j$ in the bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ so that each set of workers assigned to process $task_j$, will fulfill two objectives : (i) maximize the probability of providing the correct $response_j$, based on the group's majority vote denoted as $P(vote_{gj}), \forall u_g \in group_j$, (ii) maximize the probability that the $group_j$ of workers will execute the $task_j$ within its deadline $P(exec_{gj} < ttd_{gj})$, referred as $P(time_{gj}) \forall u_g \in group_j$, and ensure that the probability of meeting this constraint is higher than a predefined threshold, denoted as B. This is formulated as follows:

$$\begin{aligned} &\text{maximize } \big(P(vote_{gj}),\ P(time_{gj})\big), \forall u_g \in group_j \\ &\text{subject to } \sum_{i=1}^{|U|} x_{ij} \leq 1, \forall j = 1, ..., |V|, \qquad \text{where } x_{ij} \in \{0, 1\} \\ &\qquad\qquad P(time_{gj}) > B \end{aligned}$$

where x_{ij} is 1 denotes that edge $(w_i, task_j)$ is an instantiated edge for the matching. Finding a feasible solution that minimizes both objective functions simultaneously is a challenging problem. In the following section we show that the problem is NP-hard and it can be reduced from the Knapsack problem. Hence, we seek to find Pareto optimal solutions [22], [23], thus a selection of workers which cannot be improved in any of the objectives without degrading the other objective. Solving this problem at runtime raises several additional challenges that need to be addressed: (1) the amount of possible groups of workers can be numerous, thus computing all possible combinations can be computationally infeasible to provide a real-time solution, (2) the objective $P(time_{gj})$ needs to be computed at runtime since it depends on the available slack time until the $deadline_j$ of a $task_j$ expires, and (3) computing the objective $P(vote_{gj})$ is a computationally costly procedure.

IV. Task Assignment Problem

In this section we present our techniques for task assignment. We start by computing the reliability of an individual worker and a group of workers in section IV-A, and we then estimate the probability that the group of workers will execute the task within the deadline in section IV-B. Finally, in section IV-C we show the NP-hardness of the problem and propose a solution that defines the most appropriate group of workers for each task, based on the defined constraints.

A. Worker Reliability

1) Estimating Individual Worker Reliability: As discussed earlier, in a crowdsourcing system it is difficult to estimate the reliability of the users a priori, as the set of workers is not persistent, and the workers available to participate at a given time instance is not known in advance. Furthermore, some users may be new participants in the system. Several techniques have been proposed to extract user reliability in environments like crowdsourcing[14], [15].

We use an iterative approach, inspired by the streaming Expectation Maximization (EM) algorithm proposed in [24], to estimate the reliability rel_i of each worker w_i. Our technique has the benefit of determining the users unobserved reliabilities adaptively, until it converges to their true values. In addition, it is able to operate in a streaming environment, like crowdsourcing systems, by adjusting the reliabilities, whenever a user executes a task. Suppose an unobserved categorical variable X_j, for task j. The true answer of X_j is denoted as $x_j \in Q(X_j)$, where $Q(X_j)$ is the set of possible answers for X_j. For each $task_j$ we assume a uniform distribution $P(X_j)$ over the possible set of answers. The answer of w_i is denoted as $y_{i,j}$ if queried for X_j, and $Y_{i,j}$ is the associated variable. Each w_i has an unknown probability rel'_i to provide a wrong answer x, for X_j. We assume that a participant provides an answer from the defined list of answers ($Q(Y_{i,j}) = Q(X_j)$) and that each wrong answer is equiprobable. Thus:

$$P(Y_{i,j} = x_j | X_j = x_j) = 1 - rel'_i, \forall i, j \tag{1}$$

$$P(Y_{i,j} = x | X_j = x_j) = \frac{rel'_i}{|Q(X_j)| - 1}, \forall i, j, x \in Q(X_j) \setminus \{x_j\} \tag{2}$$

If the probabilities that a user lies, $\{rel'_i\}_i$, are known, inferring a distribution is a straightforward task. These parameters, however, are difficult to be estimated. The EM algorithm is a common technique to solve such problems. It alternates between computing an expectation of the parameters' values,

considering the observations and the current estimates, and updates the parameters by maximizing this expectation. Following [25], [24], [26], we use an online EM algorithm that updates the reliability values using a stochastic approximation step. For completeness we summarize the main points below.

In our approach, each participant's reliability is updated using a specific γ_{d_i}, which is defined such that $lim_{D\to\infty}\sum_{d_i=1}^{D}\gamma_{d_i} = \infty$ and $lim_{D\to\infty}\sum_{d_i=1}^{D}\gamma_{d_i}^2 < \infty$ and d_i is the number of times this participant has been queried, to perform the stochastic approximation update of the reliability estimates. Our algorithm retrieves the answers from the selected workers and updates their reliabilities rel_i for each w_i as follows. We estimate for each answer $x \in Q(x)$ of the task, the function $\alpha(x)$, that represents the posterior probability $\alpha(x) \equiv P(X_j = x|A_j, p_1, p_2, ...)$,where A_j are the associated answers for $task_j$ as: $\alpha(x) = \frac{\hat{\alpha}(y_{i,j})}{\sum_{x\in Q(X_j)}\hat{\alpha}(x)}$ where $\hat{\alpha}(x) = P(X_j = x)\prod_{i\in W}P(Y_{i,j}|X_{i,j} = x)$. Then, we estimate the reliability of each w_i, based on the answer as:

$$rel_i' = (1-\gamma_{d_i})rel_i' + \gamma_{d_i}\big(1 - \frac{\alpha(y_{i,j})}{\sum_{x\in Q(X_j)}\alpha(x)}\big) \quad (3)$$

Hence, the weight metric $rel_i = 1 - rel_i'$ represents the user reliability, where a w_i provides correct $response_{ij}$ with a probability rel_i and wrong $response_{ij}$ with probability rel_i'.

2) Group Reliability: We now explain how we compute the reliability for a group of workers $group_j$, given the reliability of an individual w_i as rel_i. One common approach in the Crowdsourcing systems[27], [28] is to allocate each task to several workers and accept as $response_j$ the one provided from the majority of workers, denoted as Majority Voting [9]. The intuition behind Majority Voting is based on the assumption that the majority of the workers can be trusted[13]. Thus, we follow that assumption, to compute the probability of the selected $group_j$ to provide a correct $response_j$ for $task_j$.

Suppose a variable r_{ij}, for each each $response_{ij}$ that w_i provides for $task_j$, where $r_{ij} = 1$ when w_i provides a correct $response_{ij}$ and 0 otherwise. Thus, r_{ij} is 1 with probability rel_i and 0 with probability rel_i'. We denote as $\mathcal{C}_j = \{[r_{1j}, r_{2j}, ..r_{group_{ij}}], \forall w_i \in group_j, where \sum_{i=1}^{|group_j|} r_{ij} >= |group_j/2|\}$. Thus, the set $\mathcal{C}_j$ contains all variations of answers from the set of all $2^{|group_j|}$ possible combinations, where at least half of the workers provide a correct answer.

Clearly, this approach is computationally costly. One way to extract this set efficiently at runtime is to perform Iterative deepening depth-first search (IDDFS)[29] to extract all possible combinations of the workers to receive a positive answer, assuming that if a worker is not included at a node, he provides a negative answer. IDDFS is equivalent to breadth-first search, but it uses less memory. It visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is breadth-first. Thus, we can extract the combinations of workers providing positive answers, while pruning can exclude all nodes whose amount of workers is less than $|group_j/2|$. Finally, we compute the probability $P(vote_{gj})$ by summing the probabilities of the individual items l in set $\mathcal{C}_j$, where the probability of each item is computed by the product of the probability of the correct or wrong answer r_{ij} that worker i provides for that item.

$$P(vote_{gj}) = \sum_{l\in L}\big(\prod rel_i, \forall w_i \in l \&\& r_{ij} = 1 * \prod rel_i', \forall w_i \in l \&\& r_{ij} = 0\big) \quad (4)$$

Thus, maximizing this probability means that the workers that will be selected will have high accuracy in the particular task category, so they can be considered as experts, and we should select these first rather than considering low-accuracy workers.

Discussion. Although in our model the correct group $response_j$ is determined by a Majority Vote approach, different evaluation metrics might be employed [30]. For instance, another evaluation metric might determine the $response_j$ as the answer selected from those workers with the highest individual reliability, through a technique such as Weighted Majority Voting. Consider the example where two reliable workers answer "a" and three unreliable workers answer "b". Such a Weighted Majority Voting metric could be defined by dividing the summation of the workers reliabilities that provided a specific answer to the amount of the workers, and selecting the highest one. For such a metric we alter the set of desired combinations as: $\mathcal{C}_j = \{[r_{1j}, r_{2j}, ..r_{group_{ij}}], \forall w_i \in group_j$, where $\big(\frac{\sum_{rel_i}}{|group_j|}, \forall w_i$ s.t. $r_{ij} = 1\big) > \big(\frac{\sum_{rel_i}}{|group_j|}, \forall w_i$ s.t. $r_{ij} = 0\big)\}$, and compute the probability using equation (4).

Also note, that, although the possibilities on the set $\mathcal{C}_j$ refer to binary answers, since in multiple choice answers a majority vote might contain less than half of the answers, incorporating multiple choice answers is a direct extension to our approach, by inserting the appropriate possibilities.

B. Real-time Execution

Estimating the execution time of a task by a worker is a challenging problem since it depends on the human factor, which makes it difficult to provide an accurate prediction beforehand. However, Ipeirotis in [31] has observed that the execution time in these systems follow a Power Law Distribution [32], forcing the execution times of the tasks to cluster around a typical value. We exploit that observation and use the CDF of the Power Law distribution for each individual worker, based on its previous execution times, to estimate the probability of a task being completed before its deadline.

A quantity x obeys power law when it is drawn from a probability distribution $p(k) \propto k^{-\alpha}$, where α is a constant parameter, known as the scaling parameter. We denote $P(k)$ the CDF of the power-law distributed variable, that is defined as $P(k) = Pr(K \geq k)$, where K is the observed value. Thus, $P(k) = \int_k^\infty p(k')dk' = (\frac{k}{k_{min}})^{-\alpha+1}$, where $k_{min} > 0$ must be a lower bound on the power-law behavior. The α value is computed as: $\alpha = 1 + n[\sum_{i=1}^{n} ln\frac{k_i}{k_{min}-1/2}]^{-1}$. In our approach the k_i values represent the execution times $exec_{ih}$ of w_i, for each task h that he has completed. The lower bound k_{min} is set as the worker's lowest measured execution time for a task. Thus, we can compute the probability $P(exec_{ij} < ttd_{ij})$, referred as $P(time_{ij})$ for each w_i to meet the $deadline_j$ of $task_j$ as: $P(time_{ij}) = 1 - P(ttd_{ij})$. Finally, estimating the probability of all the workers in the $group_j$ to finish

the execution before the $deadline_j$, can be computed as the product of the individual workers probabilities:

$$P(time_{gj}) = \prod_{w_i \in group_j} (P(time_{ij})) \quad (5)$$

Discussion. In our approach we estimate each user's reliability and execution times based on his profile. Since users might perform differently under tasks with varying difficulty or for different types of tasks, we can exploit the $category_j$ field of the user's completed tasks to preserve different user statistics for each category and estimate both reliability and execution times for the respective type of task.

C. Multi-Objective Optimization Algorithm

Naively, the task assignment problem can be solved using techniques such as branch and bound, to determine all feasible solutions and then select the optimum one. Clearly, these solutions are computationally infeasible to compute at runtime for the size of the graphs that we consider, since they would need to evaluate all possibilities of correct answers from the $\binom{n}{amount_j}$ possible groups of workers and this evaluation would need to be estimated at runtime, leading to a complexity of $\mathcal{O}(\binom{n}{amount_j}^2)$ to select $amount_j$ workers.

In the remainder we propose an approach which searches for Pareto optimal solutions[22] that considers both (i) maximizing the reliability of a group of workers and (ii) meeting real-time constraints. We present the notion of dominance between two solutions. Given two solutions, a solution is *dominated* by another one if its performance is identical or worse than the other one for both metrics (reliability and real-time) and strictly worse for at least one of the metrics. Essentially, a strategy that is not dominated by any other strategy is *Pareto optimal*; this means, it is impossible to find a better solution for both metrics without paying more in cost.

In our system, we aim at identifying feasible Pareto optimal solutions for the objective functions, by determining an assignment of the tasks to a group of workers with respect to the objectives. Existing approaches, that exploit Pareto optimality in problems that can be expressed as bipartite graphs, focus on matching the individual nodes from the vertices set, and do not consider the problem of the optimal matching for a group of vertices with another vertex [23]. We first observe that the problem is NP-hard and then we propose an effective and fast algorithm for the task assignment problem that produces high-quality results in a computational efficient manner.

NP-hardness. Let us consider a simplified version of our problem, where we aim at optimizing the one objective $P(vote_{gj})$, subject to the constraint $P(time_{gj}) > B$. Determining the optimal allocation, where each selection has a utility with a respective cost and the total bound for the cost cannot be exceeded is an NP-hard problem, as it can be reduced from the well-known Knapsack problem.

Assuming an instance of the Knapsack problem we can create an instance of our problem. Given a set of objects with utility z_i and cost c_i, and a bound C, the goal is to find a set of x objects that maximizes $\sum_{i=1}^{x} z_i$, while $\sum_{i=1}^{x} c_i \leq C$. We create an instance of our problem where for each object we have a worker w_i with reliability rel_i (as a utility) and a respective probability $P(time_{ij})$. Compared to the original problem, our objective function $P(vote_{gj})$ is more complex than the summation of values of the selected items. Although, we prove in Lemma 4.1 that $P(vote_{gj})$ is monotonic given a fixed size of workers, we do not know its monotonicity with respect to the size of the group. These properties make the objective function a generally non-linear one, which reflects more hardness than the general Knapsack Problem. Our defined constraint $P(time_{gj}) = \prod_{w_i \in group_j}(P(time_{ij})) > B$ is equivalent to $-\log \prod_{w_i \in group_j}(P(time_{ij})) < -\log(B)$, which is equivalent to $\sum_{w_i \in group_j}(-\log(P(time_{ij}))) < -\log(B)$ and so the cost for each w_i is $-\log(P(time_{ij}))$ and the bound $C = -\log(B)$. Hence, the Knapsack problem can be reduced to our problem and so our problem is NP-hard.

CRITICAl Algorithm. In order to solve the NP-hard problem of task assignment we develop a polynomial CRITICAl algorithm. Our algorithm is based on an iteration for every unassigned $u_i \in \mathcal{U}$, where in each iteration we alter the group selection $group_j$ to ensure that it fulfills the real time constraint and tries to concurrently maximize both objectives.

Our algorithm is executed in one-pass by (i) setting a boundary on one of the two objectives, (ii) determining the solution that maximizes that objective and (iii) traversing among the *feasible* solutions that are able to maximize the other objective. In the following we set a bound on the $P(time_{gj})$ objective denoted as B, although we could easily bound the other objective as well. This bound is utilized to make sure that the probability of meeting the real time constraints will be higher than B. Thus, we state that our one-pass algorithm provides a solution that fulfills the boundary for the objective function $P(time_{gj})$ (if such a solution exists) and aims to maximize $P(vote_{gj})$, for the bound that we have set.

Step 1: First we extract the list of the available workers $\mathcal{L}$ from the bipartite graph as: $\mathcal{L} = \{u_i \in \mathcal{U}$, where $e_{ij} \notin \mathcal{E} \forall task_j \in \mathcal{V}\}$ (thus there is no edge among these workers with any task) and we sort the list $\mathcal{L}$, based on the individual probabilities of the workers to meet the real-time constraints and their reliabilities, as: Sort($\mathcal{L}$) by $(P(time_{ij}), rel_i)$. Then, for the $task_j$ that we need to allocate workers, we create an empty set $group_j = \{\emptyset\}$ that represents the group of workers that will process $task_j$. This initialization step enables us to instantiate the needed structures.

Step 2: At this step we iterate through the workers in sorted list $\mathcal{L}$ and we add every w_i in $group_j$ until the $amount_j$ requirement has been fulfilled. Thus, this step ensures that we will retrieve a feasible solution, if it exists, that maximizes $P(time_{gj})$. This happens, as, due to the sorting in Step 1, we have inserted the $amount_j$ workers with the highest probability $P(time_{ij})$, and due to the definition of that probability there is no other worker allocation with $amount_j$ workers that provides a highest $P(time_{gj})$.

Step 3: We continue iterating through the workers list $\mathcal{L}$. For each w_i we instantiate a new group of workers $group'_j$, which extends our existing selected group with the current w_i. Then, we examine if the current worker is more reliable than one of the selected workers in $group_j$, while remaining among the set of feasible solutions, defined by the B bound. We investigate if we can swap that worker with any of the workers w_g assigned to the group $group_j$ and estimate if

this swap can increase the objective function that refers to the reliability. We also state that due to Lemma 4.1 (shown below), we do not compute the objective $P(vote_{gj})$ which is computationally costly, but we only evaluate the distance of each individual w_g in the group with the current worker w_i to determine the best swap decision. After determining the "best" swap, *i.e.* the swap that increases the group reliability more than the other possible swaps, while remaining in the feasible region, we choose to make the swap. Thus, we remove w_g from the set and we add w_i. Also note, that this swap produces a local optimum in every iteration as we show in Lemma 4.2. If w_i does not increase the objective probabilities, it means that he is either an unreliable worker or will probably miss the deadline, compared to our selected workers, and we do not consider him for $task_j$. We also state that we could stop the iterations when a specific threshold has been reached for both probabilities, to reduce the computation time. The pseudocode of this step is summarized in Algorithm 1.

Algorithm 1 Evaluate $w_i \in \mathcal{L}$ for $group_j$

$tmp\_group = null; tmp\_dist = 0;$
for $(w_g \in group_j)$ **do**
  $group'_j = (group_j \cup \{w_i\}) \setminus \{w_g\}$
  if $(rel_i - rel_g > tmp\_dist \;\&\&\; P(time'_{gj}) > B)$ **then**
    $tmp\_group = group'_j; tmp\_dist = rel_i - rel_g;$
if $(tmp\_group! = null)$ **then** $group_j = tmp\_group;$

Lemma 4.1: The complexity of the objective function $P(vote_{gj})$ grows exponentially with the $amount_j$ of workers, since we need to compute all the possibilities that provide positive majority answers out of the 2^{amount_j} possibilities. However, the function is monotonic, since selecting a worker k with individual reliability $rel_k \in g$, instead of a worker k' with individual reliability $rel'_k \in g'$, where $rel_k \geq rel'_k$ and $g \setminus \{k\} = g' \setminus \{k'\}$, results in $P(vote_{gj}) \geq P(vote_{g'j})$. This happens since the derivative of $P(vote_{gj})$ with respect to one of the workers reliability rel_k, provides a positive number as explained below. The derivative $P(vote_{gj})'$ is equal to the summation of the derivatives: $(\prod rel_i, \forall w_i \in l \&\& r_{ij} = 1 * \prod rel'_i, \forall w_i \in l \&\& r_{ij} = 0, \forall l \in \mathcal{L})'$. Starting from the case where everybody answers correctly: $(rel_1 * rel_2 * ..rel_k * ..rel_{|}\mathcal{L}|)'$, each of the cases, that w_k answers correctly, eliminates the negative product of the derivative of the case of answering wrongly $(rel_1 * rel_2 * ..(1 - rel_k) * ..rel_{|}\mathcal{L}|)'$. However, when we reach the case that w_k answers correctly and the correct answers are $|group_j/2|$, we cannot consider the same case with w_k answering wrongly, since we would exceed the constraints. Thus, the remaining products are positive numbers and so $P(vote_{gj})$ is monotonic and increasing.

Lemma 4.2: In every iteration the selected worker w_i has a probability to meet the real-time constraint $P(time_{ij})$, that is worse or equal to the probabilities of the workers in $group_j$, due to the sorting. However, we swap the w_i with another worker with the smallest reliability from $group_j$ that provides a solution in the feasible region. Thus, taking Lemma 4.1 into consideration, we ensure that for the set of workers $group_j \cup \{w_i\}$, there is no other feasible solution with $amount_j$ workers and highest $P(vote_{gj})$ and thus, it is a local optimum.

Dynamic Selection of the Workers. Crowdsourcing environments are inherent dynamic, affecting the structure of the bipartite graph. Thus, if new workers become available during the task assignment phase, we do not take these workers into account, since it would increase the complexity of defining a solution. On the other hand, if during a task execution workers disconnect from the system, or similarly fail to provide an answer within the time deadlines, these workers are considered as providing a wrong answer for the task.

Worst-Case Complexity of our algorithm. Assume n workers. First we sort the workers that costs $\mathcal{O}(nlogn)$ and we traverse through the workers list. In case we have not fulfilled the $amount_j$ requirement we add the current worker to the worker group that costs $\mathcal{O}(1)$, thus it costs $\mathcal{O}(amount_j)$. Then for each of the $n - amount_j$ workers, we iterate through the $amount_j$ workers in the $group_j$ to determine if a swap can provide an improvement, which costs $\mathcal{O}(amount_j * (n - amount_j))$, since the evaluation of the improvement is $\mathcal{O}(1)$. Thus, the iterations through all the workers cost $\mathcal{O}(amount_j * (1 + (n - amount_j)))$ and the worst case complexity of our algorithm is $\mathcal{O}(nlogn + amount_j * (1 + (n - amount_j)))$=$\mathcal{O}(n * amount_j)$. Keep in mind, though, that typically we expect the users to provide a small $amount_j$ of requested workers, compared to the total available workers.

V. Experimental Evaluation

Case Study. Crowdsourcing systems such as the AMT, do not allow us to control the task assignment. Thus, we carried out a study on CrowdFlower, that exploits several crowdsourcing platforms (including AMT), to extract the behaviors of the workers for different types of tasks. We created and assigned approximately 4K tasks and exported response times for the different tasks and workers accuracy (both historical accuracy provided in the Crowdflower results and the actual accuracy in the specific tasks). The crowdsourcing tasks that we injected contained Tweets from LiveDrive; this is a Twitter account where people in Dublin report traffic events. The objective was to evaluate whether each specific Tweet refers to traffic events such as "M50 South is heavy from J16 Cherrywood to Bray North Exit.", or not, such as "@livedrive I never received my t-shirt, what happened?". Each task contained five tweets and the study reported that users could answer our tasks in 69 seconds on average. However, in some cases the completion time could take up to 3 hours, due to the lack of an assignment policy, which is unacceptable for real-time response.

We used the observations and results from these tasks to define the values of the parameters in our experiments. We extracted the "trust" field from the results, to determine the users reliability. The distribution of the workers' reliability is shown in figure 7 and we will discuss it later. For the purposes of our experiments each worker that receives tasks from the system, processes them within a time interval (min_time, max_time) that is randomly decided based on the worker's profile. These times can range from an absolute minimum of 9 seconds to an absolute maximum of 90 seconds. We decided to set tight deadlines for the tasks ranging between 20-60 seconds, based on the completion times of the tasks. Hence, since the execution time of the workers could be higher than the task deadlines, except from deadline misses, we used it to model a worker that abandons a task or leaves the system. All experiments in the CRITICAl system were executed on an Intel Core2 Quad Q9300 processor with 4GB of RAM.

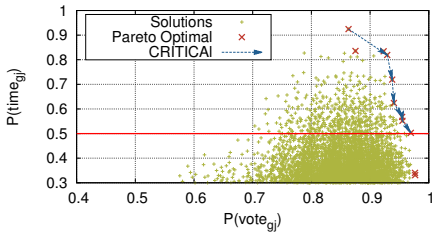

Fig. 1. Overview of CRITICAl.

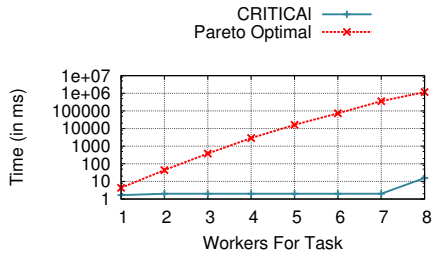

Fig. 2. Running Time - Varying Amount of Selected Workers.

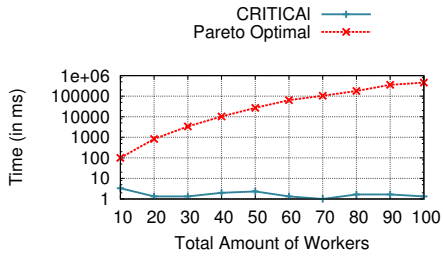

Fig. 3. Running Time - Varying Amount of Available Workers Set.

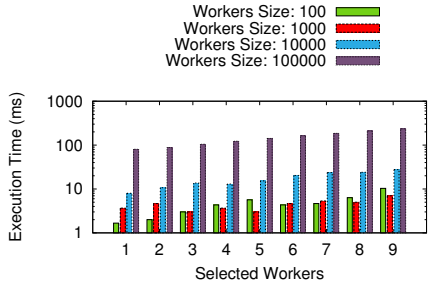

Fig. 4. Running Time for Selecting Workers.

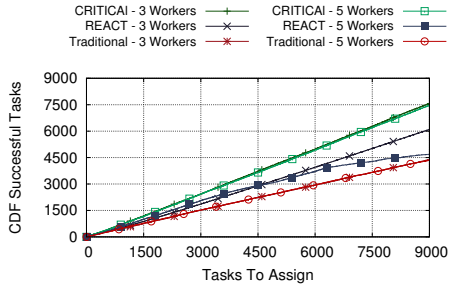

Fig. 5. CDF of Successfully Processed Tasks.

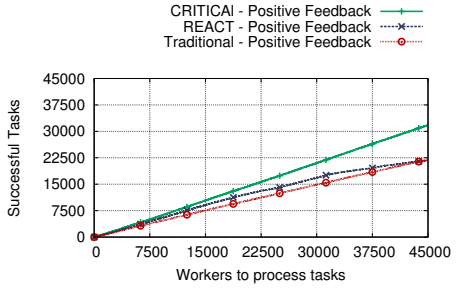

Fig. 6. CDF of Successfully Processed Tasks from Individual Workers.

Since there is no technique, that we know of, that deals with the dual problem of real-time and reliable task assignment in crowdsourcing systems, we compare our approach with: (i)the Traditional approach, where the tasks are assigned to workers in a random manner, and (ii)the REACT Algorithm, developed in our previous work in [33], that aims to determine the optimal selection among workers and tasks to meet real-time criteria only. The REACT algorithm, for a predefined amount of cycles, selects different edges for the matching arbitrarily and accepts the new state if it improves the objective function. REACT differs from CRITICAl in two important ways: (i) it was developed to match each task with one worker, rather than matching each task with $amount_j$ workers which is the focus of this work, and (ii) focuses on real-time constraints. For the purposes of the comparison we extend REACT so as to match multiple workers for each task. The experimental evaluation focuses on the following parameters: (i) **Quality of Results**, (ii) **Efficiency**, (iii) **Average Execution Time**, (iv) **Comparison with existing techniques**, (v) **Fair Selection of Workers**, (vi) **Comparison with Individual Objectives** and (vii) **Scalability Comparison**.

Quality of Results. Figure 1 illustrates the working of our approach. Assuming there are 50 available workers and we aim to select 4 workers for a task. We plot the set of "Solutions" that presents all the possible groups of 4 workers (approximately 230K) and we also plot the ones that were evaluated as "Pareto optimal" solutions (shown with a red color). Suppose that the real-time bound B is set to 0.5 (while in the following experiments we set B as 0.85^{amount_j}), and thus only the solutions above the red line are considered as feasible solutions. We plot the selection of the solutions from CRITICAl, as the algorithm iterates through the available workers. As can be observed, CRITICAl first selects the solution with the highest group probability that the deadline will be met, and traverses through the Pareto optimal solutions to improve the group reliability, while remaining in the feasible region, imposed by the real-time constraint B. CRITICAl selects 6 out of the 8 Pareto optimal solutions that exist in the feasible region and accepts the last one which reflects the maximum reliability for the selected bound B.

Efficiency. We also present the efficiency of our approach in terms of execution time compared to the optimal solution which is the first Pareto optimal solution that meets the real-time constraint. To determine that solution we evaluate all possible solutions, but we use pruning techniques to skip evaluating solutions that will not lead to an optimal or feasible solution. All of the following are averages of 5 runs.

Figure 2 presents the average time needed to select the group of workers when varying the number of workers from 1 to 8, from a set of 30 workers. The selection of a small set for the available workers and amount of selected ones, is due to the running time for determining Pareto optimal solutions. As can be seen, the running time of CRITICAl is negligible, with a worst case of 15ms for the assignment of 8 workers, while the optimal solution needed 1166 seconds. We also state that the respective euclidean distance of CRITICAl from the closest feasible Pareto optimal solution, (in the range $[0,\sqrt{2}]$) was zero in the majority of the experiments and did not exceed 0.018. In figure 3 we conduct a similar experiment, but we vary the set of available workers from 10 to 100 and select 4 workers. Again, CRITICAl does not exceed 3.3ms to provide an answer, while the optimal solution for the set of 100 workers took 455 seconds, while the euclidean distance was negligible.

Average Execution Time. In figure 4 we present the running time of CRITICAl to determine a solution for various worker sets and amount of selected workers, when the available workers could be high. As can be observed, the amount of the selected workers plays a small role to the execution time, compared to the size of the workers set. Nevertheless, considering CRITICAl servers with small available worker sets, our algorithm can provide rapid results (it takes less than 11ms for selecting 9 workers from 1000 workers). Respectively, even for a set of 10000 workers we need less than 27ms to determine groups of 9 workers, while in a worker set of 100000 workers we may need up to 235ms to find the group of 9 most suitable workers. Keep in mind that the tasks typically need some minutes to be performed, and so spending some milliseconds

Fig. 7. Accuracy per Worker.

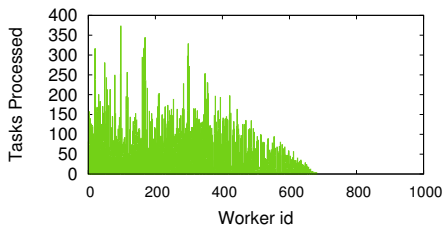

Fig. 8. Processed Tasks per Worker.

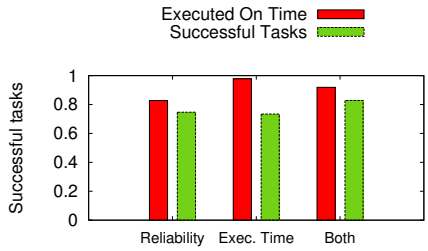

Fig. 9. Comparison with Individual Objectives.

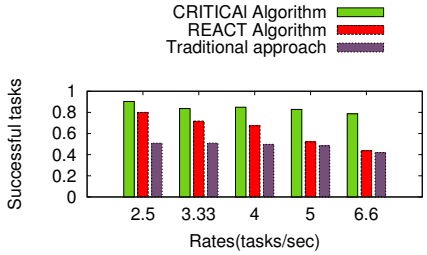

Fig. 10. Comparison on Varying Rates.

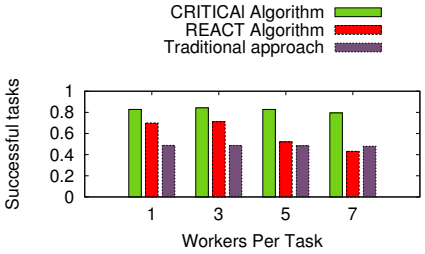

Fig. 11. Comparison on Varying Workers Amount.

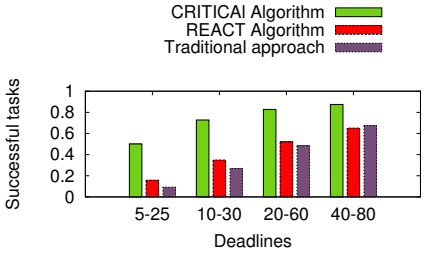

Fig. 12. Comparison on Varying Task Deadlines.

to determine suitable workers will lead to the reduction of the total execution time (due to the selection of faster workers).

Comparison with existing techniques. We also conduct real-time experiments that emulate the behavior of the workers on different tasks, that arrive on predefined rates. In all experiments we use a worker set of 1000 workers with a reliability distribution, defined by figure 7. Figure 5 illustrates the amount of tasks that have been processed successfully, meaning that the majority of the workers in the group have answered correctly and before the deadline. We present the behavior of the approaches where each task is processed either by a group of 3 or 5 workers. As can be observed, when a task is processed by 3 workers, CRITICAl outperforms the other approaches by successfully processing 84.2% of the tasks; this is due to some infeasible solutions (*e.g.* tasks with extremely small deadlines or inadequate available workers). Also note that for the first two tasks each worker receives a probability of 1.0 to process the task on time, so as to train the system, that causes some "non-optimal" assignments. The REACT algorithm with 3 workers per group has a better performance than the Traditional approach that processes approximately 48.8% of the tasks successfully, with a percentage of 72.1%. Hence, although the amount of cycles for REACT is maximized according to the tasks rate, such an approach is not enough to provide a good solution for a group of workers, in contrast to its performance for a single worker per task assignment; this is because an arbitrarily selection might insert a good worker in the group, but it requires too many iterations to provide good solutions, as the $amount_j$ increases. When selecting 5 workers, CRITICAl has a slightly reduced performance with 82.8% of the tasks successfully performed, which is due to the limited amount of workers since: (i) increasing the amount of workers increases the probability of missing the deadline individually and (ii) selecting more workers per task reduces the amount of "good" available workers for the following tasks. Although our approach is slightly affected by these factors, the REACT algorithm is highly affected with only 52% of the tasks successfully processed. The Traditional approach performs similarly with 48.3% of the tasks successfully processed due to the random selection.

Successfully Processed Tasks. Figure 6 shows the amount of tasks that were processed successfully by each individual worker. An important observation is that although only 70.6% of the tasks were successfully performed by the individual workers, in our approach, the selection of groups with both high reliability and execution time, has led to 82.8% successful tasks. Thus, although the selection might include some unreliable or slow workers, we ensure that the group as a whole will provide good results. The REACT algorithm managed to process 48.3% of the tasks successfully for the individual workers, meaning that the selection of the workers for the groups increased the efficiency slightly, due to the insufficient amount of cycles. Finally, the Traditional approach performed only 48.7% accuracy for the individual workers, similarly to the group's selection, proving that, a random selection of a large group does not ensure that the task will be successfully processed. Although the REACT and the Traditional approach have almost the same amount of successful tasks from the individual workers, the group selection of REACT leads to a higher amount of successful tasks as a group (figure 5).

Fair Selection of Workers. Figure 7 presents the corresponding accuracy of the individual workers with CRITICAl. As the figure shows, the workers typically answer with a probability that is close to their reliability percentage (as shown by the blue line). Nevertheless, the workers #640-#680, provide different percentages, since they have only received a few tasks, as we show in the following figure and thus their reliability has not converged to their actual reliability. Moreover, as illustrated in the figure, the workers after #680 have not been selected at all for executing a task. This is because CRITICAl prefers to select reliable workers (experts) when the rate of tasks enables such a selection. However, different rates or deadlines, differentiate the amount of selected workers as the system becomes more loaded. In figure 8 we show the corresponding amount of tasks that each worker processed. Note that, the amount of tasks does not depend on the reliability of the workers (although the slope is similar), since some of the workers are faster and they are preferred when a task has a tight deadline. Moreover, these workers become available faster and they can process more tasks.

Comparison with Individual Objectives. Figure 9 illustrates the benefit maximizing both objectives, although this

leads to a higher computational complexity than maximizing one of the objectives. First, we execute the same experiment as before but we select the five most reliable available workers for each task. As the figure shows only 82.8% of the tasks managed to meet the deadline and 74.2% were completed successfully, since reliable workers might respond slowly and the amount of highly reliable available workers decreases as we assign the workers to tasks. On the other hand, when we select the five workers with the highest probability to meet the deadline for each task, we manage to retrieve 97.8% of the tasks on time. However, without considering their reliability we select a lot of unsuitable workers and thus only 73.4% of the tasks were completed successfully. CRITICAl bridges the gap among these objectives and manages to process 82.8% of the tasks before the deadline.

Scalability Comparison. In this experiment we try to "stress" the approaches, in different rates and group size, to better observe their behavior. We use the same scenario as before but we vary the rate from 2.5 tasks per second up to 6.6 tasks per second. We would like to point out that we set our scalability rates (6.6 HITs/sec) even higher than the AMT rates, since according to [34] there were less than 18K HITs on AMT on the 25/7/12, while our rate (6.6 HITs/sec) equals to more than 23K HITs per hour and just for one CRITICAl server. As figure 10 shows CRITICAl provides the highest quality on the amount of successfully processed tasks, due to its fast and effective selection, with more than 78.7% successful tasks for the rate of 6.6tasks/second. Note that the percentage is slightly reduced as the rate increases, since fewer "good" workers are available in the workers set. On the other hand, the REACT is highly affected and drops from 80% at the rate of 2.5tasks/second to 43.7% when the rate becomes 6.6tasks/second. This is because there are fewer "good" workers and the algorithm needs too many cycles to identify them, while the cycles need to be reduced so that all the incoming tasks will be assigned. The traditional approach, that assigns the tasks randomly, only becomes affected when the rate becomes 6.6tasks/second where the (random) selection of slower workers leads to queuing.

Figure 11 shows how the approaches scale when the group of workers to be selected increases. As can be observed, CRITICAl is only affected for a group of 7 workers that leads to a 79.5% of the tasks being successfully processed. This derives from the fact that the "good" available workers decrease when we assign more workers on each task. The REACT algorithm is affected by the same issue, however, its percentage of successful processed tasks drops to 43.1% for a group of 7 workers, since the maximum amount of cycles that can be achieved is not sufficient when the group size increases. Once again, the Traditional approach is slightly affected, although its quality is basically unacceptable for a commercial system. We also note that, we do not further extend the group size, since (i) typically tasks are executed from less than 10 workers and (ii) selecting additional workers for each group would not increase the amount of successful tasks, as shown in the figure, since we have already selected the workers that maximize both objectives. Thus, inserting more workers for a task will not improve the result, as shown in [9], but it will increase the cost for paying the workers.

Finally, in figure 12 we present how the approaches behave with more strict and lax task deadlines, compared to the deadline we had set for the previous experiments that ranged among [20,60]. As can be observed when we use tighter deadlines ([5,25] and [10,30]), where tasks may receive deadlines which are 200% and 400% stricter than before, all approaches reduce their efficiency. However, CRITICAl is the least affected approach and the reason for the unsuccessful tasks in these deadlines is the lack of available workers that are able to execute tasks in such a short deadline. However, even with the tightest deadline, our approach manages to process successfully more than half of the tasks, while the REACT approach processed successfully 15.8% of the tasks and the traditional approach only 9.3%. On the other hand, with a lax deadline (40-80), where most of the workers can process the tasks on time, all of the approaches increase their efficiency. As expected, CRITICAl had the least increase, since it was already close to one, however, it manages to reach 87.5%.

VI. Related Work

The popularity of crowdsourcing systems has led to the development of several commercial crowdsourcing platforms, including AMT [1], CrowdFlower [2] and microWorkers [3]. Additionally, applications have began to utilize crowdsourcing systems *e.g.,* Waze[8] is a community-driven navigation application for mobile phones, where users voluntarily share location-based data, to provide real-time traffic updates.

Authors in [14] propose a fact-finding approach to determine user reliability, to allow applications to process streaming data efficiently. They develop a recursive Expectation Maximization approach that adaptively processes the newly updated data. However, CRITICAl estimates the workers reliability and determines the real-time capability of the workers as a group, to process a task, at runtime. Venetis *et al* [35] utilize Max Algorithms to determine the best answer in crowdsourcing environments. They consider the trade-off between the quality, the monetary cost and the execution time of the steps for their algorithm, rather than considering the execution time of the tasks. They also assume that workers should provide series of answers for a task, while we determine the most suitable group of workers that provides answers with high confidence without the need to retrieve series of answers to evaluate them.

Karger *et al* [18] provide a task allocation scheme based on random graphs that minimizes the number of task assignments subject to an overall reliability constraint. However, we have shown that our approach outperforms adaptive random assignments. In [16], they present a two-phase voting-based approach to maximize the accuracy of the results by dynamically selecting multiple users to process them, while we select workers based on their reliability to reduce the cost of the multiple assignments. In [17] they propose a task assignment model, similar to a simplified version of the online adwords problem, where the worker's skill remains unknown. In [27] they present a budget allocation algorithm for tasks with different costs and they aim to minimize the total error of the answers, with respect to a budget limit. However, they consider the reliability of the users executing the tasks unknown. Authors in [9] propose an approach to allocate workers that minimize the error to receive a correct answer subject to a constrained budget. In [20] they also model the assignment problem as a bipartite graph and propose techniques based on budget

feasible mechanisms. Nevertheless, none of these approaches considers the real-time constraints imposed from the tasks.

Authors in [19] present a route recommendation system, that utilizes crowdsourcing. Their worker selection process, considers the response time and the familiarity of the worker. However, the response times do not depend on the workers' history and the selection considers each worker individually. Khazankin *et al* [15] propose a task assignment model that depends on the workers' "quality" and suggest that task assignment should depend on the workers' availability, to overcome deadline issues. CRITICAl improves that, since it considers the worker profiles and uses real time measurements for the estimation of deadline misses. In our previous work [33], we also study the task allocation to workers, in terms of satisfying real-time requirements. However, that work focused in assigning tasks to individual workers and as we prove in the experiments, the synergies developed in groups of workers need a different approach to assign the tasks efficiently.

Alt *et al* [36] implement a prototype for providing location-based tasks to the mobile crowd. They exploit the GPS location to enable workers select nearby tasks, while in CRITICAl this is achieved implicitly since the workers would be assigned to tasks that reside in the same area. They also claim that the task solutions need to be submitted within a time period, but they do not propose such a solution, as we do in CRITICAl. In [28], they aim to assign tasks to the closest spatially workers and propose three approaches for the problem. Nevertheless, they do not consider the characteristics of the workers but they assume that the majority of the workers can be trusted.

VII. Conclusions

In this paper we develop a crowdsourcing system called CRITICAl and solve a task assignment problem, that efficiently determines the most appropriate group of workers to assign for each incoming task, so as to satisfy application real-time demands and to return high quality results under budget constraints. Our detailed experimental results show that our system effectively meets application requested demands, has low overhead, and is highly efficient as it improves the amount of tasks processed under the defined constraints over 71% compared to traditional approaches.

Acknowledgment

This research has been financed by the European Union through the FP7 INSIGHT project and the ERC IDEAS NGHCS project.

References

[1] "Amazon mechanical turk," http://www.mturk.com/.

[2] "Crowdflower," http://crowdflower.com/.

[3] "Microworkers," http://microworkers.com/.

[4] H. Gao, G. Barbier, and R. Goolsby, "Harnessing the crowdsourcing power of social media for disaster relief," *IEEE Intelligent Systems*, vol. 26, no. 3, pp. 10–14, 2011.

[5] S. L. Lim, D. Quercia, and A. Finkelstein, "Stakesource: harnessing the power of crowdsourcing and social networks in stakeholder analysis," in *ICSE*, Cape Town, South Africa, May 2010, pp. 239–242.

[6] P. Felber, P. Kropf, L. Leonini, T. Luu, M. Rajman, and E. Rivière, "Collaborative ranking and profiling: Exploiting the wisdom of crowds in tailored web search," in *DAIS*, Amsterdam, Netherlands, June 2010.

[7] D. R. Choffnes, F. E. Bustamante, and Z. Ge, "Crowdsourcing service-level network event monitoring," in *SIGCOMM*, New Delhi, India, August 2010, pp. 387–398.

[8] "Waze," http://www.waze.com/.

[9] C. C. Cao, J. She, Y. Tong, and L. Chen, "Whom to ask?: jury selection for decision making tasks on micro-blog services," *PVLDB*, vol. 5, no. 11, pp. 1495–1506, 2012.

[10] S. Guo, A. Parameswaran, and H. Garcia-Molina, "So who won?: dynamic max discovery with the crowd," in *SIGMOD*, Scottsdale, AZ, March 2012, pp. 385–396.

[11] L. Mo, R. Cheng, B. Kao, X. S. Yang, C. Ren, S. Lei, D. W. Cheung, and E. Lo, "Optimizing plurality for human intelligence tasks," in *CIKM*, Burlingame, CA, October 2013, pp. 1929–1938.

[12] J. Wang, P. G. Ipeirotis, and F. Provost, "Managing crowdsourcing workers," in *The Winter Conference on Business Intelligence*, Mar 2011.

[13] J. Surowiecki, *The wisdom of crowds*. Random House Digital, 2005.

[14] D. Wang, T. Abdelzaher, L. Kaplan, and C. Aggarwal, "Recursive fact-finding: A streaming approach to truth estimation in crowdsourcing applications," in *ICDCS*, Philadelphia, USA, July 2013, pp. 530–539.

[15] R. Khazankin, H. Psaier, D. Schall, and S. Dustdar, "Qos-based task scheduling in crowdsourcing environments," in *ICSOC*, Paphos, Cyprus, December 2011, pp. 297–311.

[16] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "Cdas: a crowdsourcing data analytics system," *PVLDB*, vol. 5, no. 10, 2012.

[17] C.-J. Ho and J. W. Vaughan, "Online task assignment in crowdsourcing markets," in *AAAI*, Toronto, Canada, July 2012.

[18] D. R. Karger, S. Oh, and D. Shah, "Iterative learning for reliable crowdsourcing systems," in *NIPS*, Granada, Spain, December 2011.

[19] H. Su, "Crowdplanner: A crowd-based route recommendation system," in *ICDE*, Chicago, IL, USA, March 2014.

[20] G. Goel, A. Nikzad, and A. Singla, "Matching workers expertise with tasks: Incentives in heterogeneous crowdsourcing markets," in *NIPS*, Lake Tahoe, NV, USA, December 2013.

[21] S. Subramaniam, V. Kalogeraki, and T. Palpanas, "Distributed real-time detection and tracking of homogeneous regions in sensor networks," in *RTSS*, Rio de Janeiro, Brazil, Dec 2006, pp. 401–411.

[22] K. Deb *et al.*, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons Chichester, 2001, vol. 2012.

[23] R. Fleischer and Y. Wang, "Dynamic pareto optimal matching," in *ISISE*, Shanghai, China, December 2008, pp. 797–802.

[24] O. Cappé and E. Moulines, "On-line expectation–maximization algorithm for latent data models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, no. 3, pp. 593–613, 2009.

[25] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, and D. Kinane, "Heterogeneous stream processing and crowdsourcing for urban traffic management," in *EDBT*, Athens, Greece, March 2014, pp. 712–723.

[26] E. Ozkan, C. Fritsche, and F. Gustafsson, "Online em algorithm for joint state and mixture measurement noise estimation," in *FUSION*, Singapore, July 2012, pp. 1935–1940.

[27] L. Tran-Thanh, M. Venanzi, A. Rogers, and N. R. Jennings, "Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks," in *AAMAS*, Saint Paul, MN, USA, May 2013, pp. 901–908.

[28] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *SIGSPATIAL*, Redondo Beach, CA, November 2012, pp. 189–198.

[29] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial intelligence*, vol. 27, no. 1, pp. 97–109, 1985.

[30] S. Xie, W. Fan, and S. Y. Philip, "An iterative and re-weighting framework for rejection and uncertainty resolution in crowdsourcing." in *SDM*, Anaheim, CA, USA, April 2012, pp. 1107–1118.

[31] P. G. Ipeirotis, "Analyzing the amazon mechanical turk marketplace," *XRDS*, vol. 17, no. 2, pp. 16–21, Dec. 2010.

[32] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, no. 4, Nov. 2009.

[33] I. Boutsis and V. Kalogeraki, "Crowdsourcing under real-time constraints," in *IPDPS*, Boston, MA, USA, May 2013, pp. 753–764.

[34] "Mechanical turk tracker," http://mturk-tracker.com/arrivals/.

[35] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments," in *WWW*, Lyon, France, April 2012, pp. 989–998.

[36] F. Alt, A. S. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz, "Location-based crowdsourcing: extending crowdsourcing to the real world," in *NordiCHI*, Reykjavik, Iceland, Oct 2010, pp. 13–22.