Crowdsourcing under Real-Time Constraints

Ioannis Boutsis
Department of Informatics
Athens University of Economics and Business
Athens, Greece
mpoutsis@aueb.gr

Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business
Athens, Greece
vana@aueb.gr

***Abstract*—In the recent years we are experiencing the rapid growth of crowdsourcing systems, in which "human workers" are enlisted to perform tasks more effectively than computers, and get compensated for the work they provide. The common belief is that the wisdom of the "human crowd" can greatly complement many computer tasks which are assigned to machines. A significant challenge facing these systems is determining the most efficient allocation of tasks to workers to achieve successful completion of the tasks under real-time constraints. This paper presents REACT, a crowdsourcing system that seeks to address this challenge and proposes algorithms that aim to stimulate user participation and handle dynamic task assignment and execution in the crowdsourcing system. The goal is to determine the most appropriate workers to assign incoming tasks, in such a way so that the real-time demands are met and high quality results are returned. We empirically evaluate our approach and show that REACT meets the requested real-time demands, achieves good accuracy, is efficient, and improves the amount of successful tasks that meet their deadlines up to 61% compared to traditional approaches like AMT.**

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

I. Introduction

In the recent years we have witnessed the proliferation of crowdsourcing marketplaces that are increasingly being employed to solve computational problems that require human intelligence. Crowdsourcing refers to a distributed problem-solving process that involves outsourcing tasks to a network of people, known as the crowd [1]. The basic practice in these systems is to leverage the strengths and diversity of the human crowd to perform tasks more effectively than computers, for a small payment. Crowdsourcing systems provide access to a diverse, on-demand, scalable workforce of human workers capable of performing specific, small tasks in return for a payment. Several prominent companies have already recognized the opportunities presented in such mass collaboration systems and have developed a business model [2]. Thus, a number of commercial crowdsourcing systems have been developed, including Amazon's Mechanical Turk(AMT) [3], CrowdFlower [4], CloudCrowd [5], microWorkers [6] etc. Recently, systems such as Gigwalk[7] and FieldAgent[8], aim to incorporate crowdsourcing in *time-sensitive* tasks such as traffic monitoring, location-aware surveys, points-of-interest suggestions, real-time entertainment recommendations, price checks, etc.

One significant shortcoming facing these systems, is that, despite the reward given to the users, crowdsourcing attracts a diverse pool of workers from different locations around the world that earn money by mainly executing random tasks at variable scheduling speeds. The reason is two-fold: First, while a few systems have the capability to request feedback before providing compensation, the vast majority of the crowdsourcing systems gives compensation via an automated payment method without checking the quality of the results. As a consequence, low-quality results or the existence of malicious workers can severely degrade the quality of the outcome, leading in poor system performance. Second, existing crowdsourcing systems have no provision for *real-time response* (*i.e.,* their goal instead is to optimize throughput rather than be responsive). We consider this a fundamental challenge for the wider adoption of the systems. However, real-time execution is a challenging problem due to highly dynamic crowd, as well as resource constraints and fluctuations in network quality and channel capacity, especially for crowdsourcing tasks over resource constrained environments such as mobile environments. This makes it extremely difficult to estimate execution times to provide real-time support.

The task assignment problem is closely related to several problems that have been studied in the literature. An offline version of the problem can be solved using linear programming or by the Hungarian algorithm [9]. However, these approaches have high computational overhead which makes them inappropriate for use in dynamic systems. The online version of the problem can be considered as an online bipartite matching problem, similar to the online adwords problem [10], or the related display ad allocation problem [11]. Our problem is closely related to the task assignment problem in distributed systems, where static and dynamic techniques have been proposed mainly for load balancing, and for which several works exist in the literature (including earlier work from our group) [12], [13], [14], [15]. However, crowdsourcing systems differ from traditional task assignment techniques in three important ways: (1) They do not aim to balance the workload to meet machine performance

metrics. They rather aim to assign a highly variable workload to human workers that depict a wide variety of skill levels, which is often unknown in advance. (2) We cannot assume that workers are persistent and that are willing to work on subsequent tasks, since, even the most reliable workers may have short connectivity cycles and may have particular skills for specific tasks. (3) The human factor makes the execution time of the tasks uncertain, which makes it difficult to be estimated in advance. Thus, crowdsourcing systems require online solutions in order to define the best matching among tasks and workers that would be able to solve the tasks under time constraints. The algorithm must operate in batches, during which an appropriate assignment of tasks to the most relevant workers both in terms of skills and time constraints is performed, for all tasks submitted during the batch.

In this paper we present **REACT** (REAl-time schEduling for Crowd-based Tasks), the middleware that we have developed that aims to exploit the wisdom of the crowd for scheduling tasks to workers under time constraints. REACT combines crowdsourcing with scheduling techniques to improve the quality of the results and satisfy user real-time response requirements. Our work targets applications with tight deadlines, such as CrowdSearch [16], and is the first paper that we know of that deals with this problem. In order to address these two challenges we develop a task scheduling approach where the incoming tasks are matched to the most appropriate human workers, using a Bipartite matching that is executed dynamically at runtime. Our scheduling approach can be used to find the most appropriate worker for each task based on multiple attributes such as location, reputation, profile, etc. In addition, we use a probabilistic scheme in order to estimate whether the worker will be able to complete the task before the deadline or the task should be reassigned to another worker that will be able to fulfil that demand.

We summarize our contributions below:

- We present REACT, our middleware that uses estimates of worker profiles (end-to-end completion times, feedback accuracy, etc.) to achieve a dynamic assignment of the incoming tasks to the most appropriate workers available, based on worker profiles, skills and real-time computational capabilities.
- We model the dynamic task assignment problem as an online Weighted Bipartite Graph Matching (WBGM) problem and provide an online algorithm that aims to maximize the probability of selecting the most appropriate workers dynamically, while meeting the real-time constraints of the tasks.
- Our approach uses a probabilistic model to estimate the execution times of the tasks and compute the probability of meeting their real-time demands.
- We carried out a case study on CrowdFlower to understand the behavior of the human workers in terms of response times, accuracy etc, on real-time location based tasks such as the traffic monitoring application.
- We performed extensive experiments on PlanetLab [17] to validate our approach. Our experimental results illustrate that our approach is practical, effectively schedules crowdsourcing tasks by selecting the most appropriate workers and it manages a reduction of upto 45% on the execution time of the tasks, while it meets the real-time demands of 61% more tasks compared to the traditional approaches like AMT.

II. Background

We first provide a brief introduction to crowdsourcing. Crowdsourcing systems constitute a marketplace for tasks, where humans (called the *Requesters*) define tasks, and *Crowdworkers*, consisting of experts, small businesses and human users, execute them in exchange for a small monetary reward. Crowdsourcing is more effective for tasks that cannot be easily automated, or tasks that humans can do much more effectively than computers. For example, humans can easily tag a photograph with a description based on its content, or when events such as concerts occur in a city, humans can provide up-to-date local traffic conditions around the event areas. Crowdsourcing could also be combined with mechanisms that are traditionally computer-based, in order to further improve their efficiency. In a typical crowsourcing system, human requesters are asked to provide a rating from Excellent to Bad (e.g., Bad=1, Poor=2, Fair=3, Good=4, Excellent=5) to grade the quality of the workers. Several commercial crowdsourcing platforms today have developed a business model around crowdsourcing such as AMT [3], CrowdFlower [4], CloudCrowd [5], microWorkers [6] etc.

Traditional crowdsourcing architectures, such as Amazon's Mechanical Turk, are not designed for optimizing the task assignment process. First, typical crowdsourcing systems do not provide a task assignment policy; they let the workers make their own decisions about which tasks to execute. The majority of the current crowdsourcing platforms announce received tasks at their portal and then the workers choose among the assorted mass of tasks the ones that they would like to process. Since the assignment is initiated by the worker, it hardly allows the system to have an influence upon assignments such as determining the suitability of involved workers. Thus, the task selection is highly motivated only by worker preferences and not by their skills, location, efficiency or reputation. The tasks submitted to the platform also vary significantly in their characteristics. As multiple workers could be suitable for a task, efficient crowdsourcing solutions should assign the task to the most suitable worker that would be willing to execute the task.

Another challenge facing crowdsourcing systems is that they do not provide support for meeting real-time response demands, which is fundamental for the real-time applications that we consider. Current systems enable the requesters to define expiration times for the tasks in the form of soft deadlines; these deadlines represent the available time that

each worker receives in order to complete the task. If the deadline expires while being executed, the task returns to the tasks repository as unassigned. However, although requesters are able to define an expiration time for a task, there is no mechanism in place to make sure that the tasks will be executed before they expire.

Crowdsourcing tasks can be executed by workers through their personal devices such as desktop machines, mobile devices or cellphones. The techniques we propose are used to support heterogeneous workers in the crowdsourcing platform, but we consider that they are all entitled to receive the same reward although the resource cost is higher for the mobile users, e.g., due to battery consumption, possible 3G costs, etc.

Finally, the monetary payment in these systems is relatively low. Based on [18], 90% of the tasks pay less than 0.10$. Thus, we conclude that one worker has to complete a lot of tasks in order to receive a decent payment. On the other hand, requesters are able to have their tasks completed and receive answers at a relatively low cost.

III. System Architecture and Model

A. Crowdsourcing Architecture

The REACT system, as shown in Figure 1, comprises a number of users, acting either as task *Requesters*, by allocating tasks to the system, or/and as *Crowd workers* by receiving tasks to process, and a number of servers, responsible to receive tasks from the Requester nodes and matching them to the most appropriate Workers. We assume a spatial decomposition of the geographic area into a number of non-overlapping regions, similar to that of [19], where each region is assigned to a REACT server that is responsible for the matching in the region. The organization of the area into regions can be done with respect to the size of the geographic area or the number of tasks in the region, possibly defining several tiers at different levels of granularity, ranging from small local areas at the lowest tier, to the entire network area at the highest tier; this allows the system to collect task information from all the users in a scalable manner. The region size can be defined based on the application characteristics. Our experimental evaluation has shown that an average of 500-1000 workers per region depicts good performance, since the selection of the workers to assign 1000 tasks takes almost 10 seconds.

The REACT Server is implemented with the following four main components that work in concert. These are : i) a Profiling Component, ii) a Task Management Component, iii) a Scheduling Component and iv) a Dynamic Assignment Component.

The **Profiling Component** is responsible to keep track of the workers' information and statistics. It maintains for every worker (identified by $worker\_id$), information about his $geographical\_location$ and current $availability\_status$. Furthermore, it keeps statistics about every task that has

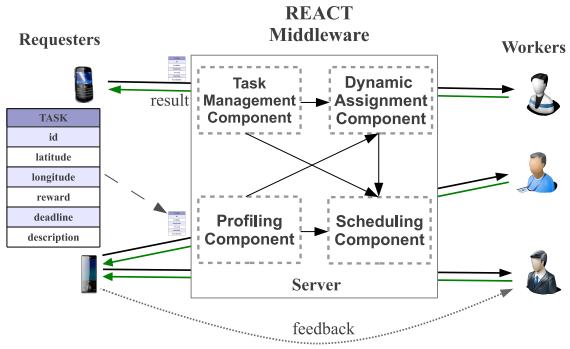

Figure 1. REACT architecture.

been assigned to the worker, such as $completion\_time$, $feedback\_accuracy$ (defined by the total amount of positive and negative feedbacks that requesters have provided upon the completion of their task by the current worker), $task\_category$, etc. This makes it easy to compute the accuracy of each worker for the different types of tasks, preference on specific task categories, etc. Our model follows closely the AMT model, where parameters such as skills and interests are not considered for the workers.

Every task that a Requester loads to our platform is received by the **Task Management Component**, which is responsible to provide information about all the available tasks in the REACT platform. Examples of such information is the task's $remaining\_time$ until it expires, if the task is currently assigned to a worker and what is the id of that worker, the $time\_elapsed$ since it was assigned, etc.

The **Scheduling Component** is responsible to match the unassigned tasks (which are provided by the Task Management component) to the Available Workers (that are maintained at the Profiling Component). This component creates a graph as will be discussed in the following sections, in order to perform a matching based on the tasks' needs and the workers' capabilities. Subsequently, the Scheduling Component assigns the tasks to the workers.

The **Dynamic Assignment Component** is a component that estimates for each assigned task the possibility of meeting its real-time demands, based only on the worker's profile. For each worker, we estimate the probability of meeting the task real-time demands, using a Cumulative Distribution Function (CDF) based on the worker's previous performance data. The purpose of this component is to remove an assigned task from the specific worker, when it seems infeasible to meet the task's real-time demands, so as to enable the Scheduling Component to find a better match.

B. Profiling

Our system comprises a set of workers denoted as $worker_i \in W$, that register to our system in order to receive task assignments. Tasks are being inserted to the system, by the Requesters. Each $task_j \in T$ is associated in the system with a number of information as $< id_j, latitude_j,$

$longitude_j$, $deadline_j$, $reward_j$, $description_j >$, that we use in the location based application in our experiments. Thus, every task receives a unique id so as to be tracked, the coordinates of the location that the task involves, the time interval in which the task should be completed, the corresponding reward and a task description. The task description, provided by the requester, describes what information needs to be provided by a $worker_i$. Examples of task descriptions are: *"Is road A highly congested?"*, *"How much time do I need to go from location B to location C?"*, etc. Each $worker_i$ is compensated by a monetary $reward_j$, defined by the requester, when he/she completes $task_j$.

In our model each $task_j$ is associated with a soft real-time $deadline_j$, that represents the time interval within which the $task_j$ should be completed in the system. We assume a soft real-time system, where our goal is to maximize the number of deadlines met and missing a deadline is not catastrophic for the system. Moreover the metric $TimeToDeadline_{ij}$ represents the time interval from the timepoint of the assignment of $task_j$ to $worker_i$ until the $deadline_j$ expires. We also denote as t_{ij} the time that has elapsed since the $task_j$ was assigned to $worker_i$ and $ExecTime_{ij}$ as the total time from the assignment of the $task_j$ to $worker_i$ until its completion.

C. Weighted Bipartite Graph

Crowdsourcing assigns tasks to an unknown, large and dynamic group of workers rather than a fixed, stable number of nodes, such as in traditional distributed environments, e.g. clusters. Our REACT middleware is responsible to select the worker who will process each task, so that the real-time demands of the task are met.

We note that, in our approach, a worker is assigned with one task at a time. This way we have an up-to-date view of workers and resources. On the contrary, assigning a series of tasks to a worker would be more complex since additional constraints (synergies and correlations among tasks) should be taken into consideration. Nevertheless, solving the problem for a single task has merit independently.

Our task assignment model is represented by a weighted bipartite graph. A bipartite graph $G = (U, V, E)$ is a graph whose vertices can be divided into two disjoint sets U and V such that each edge $(u_i, v_j) \in E$ connects a vertex $u_i \in U$ with one $v_j \in V$. If each edge in graph G has an associated weight w_{ij} , the graph G is called a weighted bipartite graph. We use the weighted bipartite graph to represent all the possible assignments between the workers and the tasks.

In our approach, each vertex in U represents a $worker_i$ who is available to execute tasks. Vertices in V represent all tasks $task_j$ to be processed by workers. The respective edges $(worker_i, task_j) \in E$, 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 corresponding to a different worker.

Each edge is associated with a weight, computed via function $F(worker_i, task_j)$. Thus, we define $w_{ij} = F(worker_i, task_j)$ for every edge among a worker and a task. This function may depend on several factors such as the worker's previous executing times compared to the deadline, the accuracy of the worker's executed tasks, the worker's location, etc. We present the definition of the weight function that we used in our experiments in section 4.

Each $task_j$ is associated with a geographical region so the task set changes only when new tasks arrive or executing tasks finish processing. Our Dynamic Assignment Component is able to deal with changes in the worker set as well, by reassigning the tasks when workers abandon the system and new workers can receive unassigned tasks.

Maximum Weighted Bipartite Matching. In the bipartite graph $G = (U, V, E)$, a matching M of graph G is a subset of E such that no two edges in M share a common vertex. If the graph G is a weighted bipartite graph, the maximum weighted bipartite matching is a matching whose sum of the weights of the edges is maximum. Hence, in our model the maximum weighted bipartite matching can be formulated as follows:

$$
\begin{aligned}
Maximize \quad & \sum_{(worker_i, task_j) \in E} w_{ij} * x_{ij} \\
subject\ to \quad & \sum_{i=1}^{|U|} x_{ij} \leq 1, \forall j = 1, ..., |V| \\
& \sum_{j=1}^{|V|} x_{ij} \leq 1, \forall i = 1, ..., |U| \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

where x_{ij} is 1 denotes that edge $(worker_i, task_j)$ is an edge in the maximum weighted bipartite matching. and with w_{ij} representing the weight for the respective edge among $worker_i$ and $task_j$.

Our goal is to solve the above maximization problem, that is, to select the edges, that would provide the highest $\sum_{(worker_i, task_j) \in E} w_{ij} * x_{ij}$ subject to the constraints, in order to schedule the execution of the tasks to the appropriate worker nodes. The maximization function is computed as the sum of the weights w_{ij} that exist in the matching solution. The constraints are formulated so that each worker would not be connected with more than one task, and each task would not be connected with more than one worker. This process is done by the Scheduling Component, as shown in Figure 2.

Task Rewards. One important aspect in the crowdsourcing systems for the selection of the tasks by the workers is the $reward_j$ of $task_j$. Typically, users select either tasks with a low monetary compensation that need a little time to be processed or tasks with a higher reward that take a lot of time. Although we do not use pricing in our implementation, our approach is flexible enough and could be extended so

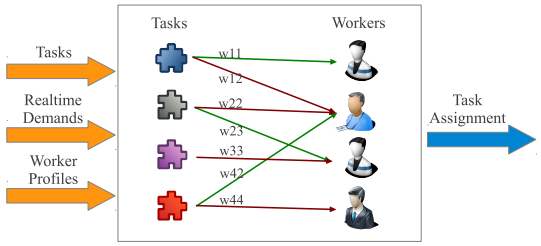

Figure 2. Weighted Bipartite Graph Matching process.

that pricing issues are considered. In order to achieve that we would only have to add a task reward range on the user's profile (that could be changed based on the user's current needs and mood), which could be exploited when we create the edges among tasks and workers. Thus, if the $reward_j$ of $task_j$ does not meet the reward range demands of the $worker_i$ the respective $(worker_i, task_j)$ edge would not be instantiated, $(worker_i, task_j) \notin E$.

IV. Our Approach

In this section we present and discuss our techniques for task assignment approach.

A. Task Assignment

Our technique aims to define the most appropriate workers for each task. To accomplish that we propose a dynamic assignment algorithm, whose purpose is to match each $task_j$ that has been received in the recent time interval and is reported by the Task Management Component as unassigned, to the most appropriate $worker_i$. This problem is solved by our online Weighted Bipartite Graph Matching algorithm. Our solution works in batches, which are initiated periodically, or if the number of unassigned tasks has exceeded a boundary.

Graph Construction. If we had a static environment with specific workers and long-running tasks, we could assume that the graph can be given to the system. However, in the crowdsourcing systems we consider the environment can be very dynamic, where the workers and the tasks depart and arrive frequently and the respective graph changes accordingly. Thus, in our approach the weighted bipartite graph is constructed and maintained in real-time from the Scheduling Component of each REACT Server, for the Server's region. This approach, reduces the size of the matching problem, without affecting the output, since the server needs to keep track only for the tasks that involve its region and the workers that currently belong to its region. Each worker is registered to the server related to the area where he belongs, based on his $geographical\_location$, provided to the system, and each task is registered to the respective server based on the task's $latitude_j$ and $longitude_j$.

Whenever a worker is available, the corresponding vertex is added and vice versa. The same procedure is repeated for each $task_j$ that is available to be assigned in the server's region. Moreover, when a task arrives we should also create the corresponding edges with the workers. Each $task_j$ could be connected to multiple $worker_i$. In our system the choice of whether to instantiate an edge among a task and a worker or to prune the current edge, meaning that the edge will not be inserted at all on the set of edges, is based on the estimation of the probability that $worker_i$ will process $task_j$ before $deadline_j$ expires. Hence, we define the probability $\Pr(ExecTime_{ij} < TimeToDeadline_{ij}) \in [0, 1]$ that represents that case and which can be easily computed (as will be shown later). We only create edges when this probability is above a specific application-defined lower bound, which essentially represents when we stop instantiating edges. Thus, when the $\Pr(ExecTime_{ij} < TimeToDeadline_{ij})$ is greater than the threshold we instantiate the edge and we assign the corresponding weight, defined by $F(worker_i, task_j)$, otherwise we prune the edge. Note that for the first z assignments of a new worker, we instantiate the edges with all available tasks and we assign the maximum value of $F(worker_i, task_j)$ to train him and build the worker profiles (accuracy, execution times) based on the executed tasks.

Weight function. In order to define the weights for the edges among tasks and workers there are several factors that need to be considered. The weight should be decided based on the application's goals. Thus, the weight function may include real time variables (such as the probability that the Deadline will not be exceeded by the worker), the worker's location, etc. In our approach, since each server is responsible for a specific region, it is expected that the selected worker would be in the region that the task denotes. However, we could use their geographical distance on the weight in order to get the nearest worker for the specific task, similar to [20], as $w_{ij} = distance(geographical\_location_i, [latitude_j, longitude_j])$. This is useful for location-based applications such as congestion detection, since a worker who is physically located on the requested location would provide accurate results. Although there are several different weight functions that can be employed, in the experimental evaluation, we use the weight function that is based on the worker's "quality", similar to [21]:

$$F(worker_i, task_j) = w_{ij} = \frac{\sum PositiveTasks_{ij}}{\sum FinishedTasks_{ij}} \quad (1)$$

The factor $\frac{\sum PositiveTasks_{ij}}{\sum FinishedTasks_{ij}} \in [0, 1]$ represents the percentage of the positive feedbacks that $worker_i$ has received for his completed tasks that belong in the same category with $task_j$, out of his total finished tasks in that category. This information can be extracted from the worker's profile. The requester of the task gives a positive or negative feedback for $worker_i$ that completed the task to denote the quality of the result and this is recorded by the system. Hence, the weight metric w_{ij} represents the user accuracy for the specific task

category. This choice was made since the users with high accuracy in a particular task category are considered as experts, and thus we should select these first rather than considering low-accuracy workers. Moreover, we suggest that low weighted edges could be pruned to reduce the graph's size since they would imply a task assignment with worker of a low quality, which means that the worker is either malicious or inappropriate for this type of task. Thus, we choose a weight for the edges of the graph with the goal of finding high-quality workers for the task and we only create edges when the goal of the real-time demands should be fulfilled. Nevertheless, we have seen [18] that there are tasks that are being avoided even by high-accuracy workers in the AMT due to small rewards. In such cases, low accuracy workers could be selected to execute these tasks.

REACT Weighted Bipartite Graph Matching (WBGM) Algorithm. The algorithm for selecting the set of edges among tasks and workers in order to assign the tasks has to be both effective and fast. Algorithms that produce optimal results for the WBGM problem are computational costly due to high complexity. Thus, we propose an algorithm that produces both high-quality results and also in a short time.

Our technique is based on states, where the selection of a new state is either accepted or rejected compared to a fitness function $g(x)$ that we wish to maximize. Each state represents a selection of edges for the matching problem. Thus, in every iteration we alter that selection and decide to accept the new selection, if this new state improves the result of the problem.

Let x be the current search state, where $x \in \{0,1\}^{|E|}$ describes the set of matchings $(worker_i, task_j)$. x_{ij} = 1 denotes a matching edge and x_{ij} = 0 denotes that it has not been selected. We iterate the following algorithm for a predefined number of cycles c. The number of cycles can be either small when our main purpose is the speed of the algorithm, or large when we aim on maximizing the output of the matching. This parameter plays a significant role both to the optimality of the solution and to the execution time. In our evaluation the cycles parameter has a predefined value, however an adaptive cycles parameter based on the graph's order of magnitude could be selected. In each iteration, we choose $(worker_i, task_j) \in E$ at random and flip x_{ij}. Thus, in the current state x, we develop a new search state $x' = x'_{00}, ..., x'_{|U||V|}$ where $x'_{ij} = 1 - x_{ij}$, and $x'_{gh} = x_{gh}$, if the edges $(task_g, worker_h) \neq (task_i, worker_j)$. After developing a new state x', we use a fitness function g to decide whether to choose the new search state x' as the next state or not. We use a fitness function g, where $g(x) = 0$ for the state x where two edges in the matching share a common vertex, and $g(x) = \sum_{worker_i \in U, task_j \in V} x_{ij} * w_{ij}$ for the state x, that does not have two edges sharing the same vertex. This fitness function g also guarantees the matching is reasonable in each iteration. In the case that $g(x') \geq g(x)$, we accept x'. Else, if $g(x') = 0$, then it means that we tried to match two vertices and another edge was already matched with one of the vertices, or two different edges were already matched with each of the two vertices. Thus, if the new edge has a higher weight than the old one(s) we remove the old edge(s) and accept the new state. Else, we remove the new edge. Finally in case where $g(x') < g(x)$ we accept x' with the probability $e^{\frac{g(x')-g(x)}{K}}$, where K is a constant, otherwise, we return to x. The steps of our algorithm are summarized in Algorithm 1.

Algorithm 1 REACT Weighted Bipartite Graph Matching

```
while (Loops < c) do
  Choose (worker_i, task_j) ∈ E at random and flip x_ij
  if (g(x') ≥ g(x)) then
    accept x'
  else if (g(x') = 0) then
    Define the weight w_kl for every already matched
    edge (worker_k, task_l)' ∈ E for which k = i or
    l = j
    if (w_kl < w_ij for each w_kl) then
      Remove (worker_k, task_l) and accept x'
  else if (g(x') < g(x)) then
    sample α from (0, 1)
    if (α ≤ e^((g(x')-g(x))/K)) then
      accept x'
```

Time vs. Optimal result trade-off. Although our approach does not provide optimal results, it is able to define a solution quickly. Our choice to use a technique to define the best workers for each task quickly, is based both on the size of the graph and on the real-time application that we consider. Thus, the delay of defining the optimal workers might result in the expiration of the tasks' deadlines. Moreover, we expect that the size of the graph in our system will be large, with numerous workers and tasks and optimal solutions for large graphs in WBGM need time that is unacceptable for our real-time application. The benefit of choosing such an approach is illustrated in the Experimental Evaluation Section.

Worst-Case Complexity of the REACT WBGM. Our approach runs for c cycles. Choosing a random edge and flipping it, only costs $O(1)$. In every cycle the algorithm computes the new $g(x')$ that also costs $O(1)$, by adding or subtracting the edge's weight or setting $g(x') = 0$ if one of the two vertices is marked as matched. It costs only $O(1)$ to verify if the new state is better than the previous one and accept it. In the case where $g(x') = 0$ we need to find the matched edges that contain $worker_i$ or $task_j$ and define their weights to make a decision about the new state. In order to do that we need to consider all edges which takes $O(E)$. Finally, the cost for the case $g(x') < g(x)$, where we select the state with a predefined probability is $O(1)$. Thus,

the worst-case complexity of our algorithm is $O(cE)$.

B. Estimating the execution time of the tasks

For each executing $task_j$ at a $worker_i$ node we need to be able to estimate if the $worker_i$ will be able to successfully accomplish the task given the real-time requirements of the task. The basic idea is, that, because these systems depend on the human factor, we may not be able to have an accurate prediction for the execution time of the task beforehand. For example, a worker might want to do other things in parallel that extend the expected execution time or he/she might even abandon the task completely without informing the crowdsourcing system. However, it has been observed that these systems follow a Power Law Distribution [22], based on the analysis of Ipeirotis in [18]. That means that the execution times of the tasks should cluster around a typical value. We make use of that observation in order to estimate the probability of a task being completed before its deadline, using the CDF for the Power Law Distribution for the current worker, which is based on its previous execution times, provided by the Profiling Component.

Hence, for each timepoint t_{ij} we need to compute the probability $Pr(t_{ij} < ExecTime_{ij} < TimeToDeadline_{ij})$, using the CDF of the Power Law Distribution. If this probability gets lower than a threshold we should reassign the task to another worker. That scheme takes advantage of the fact that the probabilities for these distributions decrease rapidly after they exceed the typical values. Thus, the remaining time can be enough for another worker to accomplish the task before the $deadline_j$ expires.

Mathematically, a quantity x obeys a power law if it is drawn from a probability distribution $p(k) \propto k^{-\alpha}$, where α is a constant parameter of the distribution known as the exponent or scaling parameter. In order to define the probability that we discussed above we need to use the CDF of the power-law distributed variable, which we denote $P(k)$ and 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 value of α 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. Thus, we use the information stored at the worker's i profile to obtain the execution times $ExecTime_{ih}$, for each task h that he has completed, in order to compute the probability $Pr(TimeToDeadline_{ij})$ and thus the probability that the $ExecTime_{ij}$ on $worker_i$ will meet the Deadline of $task_j$. The lower bound k_{min} is set as the worker's lowest measured execution time for a task. Similarly, we compute the probability $Pr(t_{ij})$ of the current execution time t_{ij} of $task_j$ on $worker_i$, to be higher than the typical execution times. Finally, we need to compute the intersection of the possibility that the execution time would be less than the deadline and higher than the current timepoint. This probability is computed as:

$$Pr(t_{ij} < ExecTime_{ij} < TimeToDeadline_{ij}) = 1 - (Pr(TimeToDeadline_{ij}) + (1 - Pr(t_{ij}))) \quad (2)$$

Hence, we estimate the probability of a worker to finish the execution of a task in the time interval, before the deadline and after the current timepoint. If this probability gets lower than a predefined threshold we should reassign the task to another worker. For instance, in our experimental evaluation we have set the threshold as 10%, meaning that if the probability of a worker to complete a task before the deadline is less than 10%, then the task should be reassigned. This happens since we expect that the worker has either abandoned or delayed the task by doing something else in parallel, when this probability highly diminishes. Thus, if we estimate that the task would not be completed in time by the current worker it is essential to have it reassigned.

The probability that we used earlier, when constructing the graph, $Pr(ExecTime_{ij} < TimeToDeadline_{ij})$, to decide whether to instantiate the edge, can be computed as:

$$Pr(ExecTime_{ij} < TimeToDeadline_{ij}) = 1 - Pr(TimeToDeadline_{ij}) \quad (3)$$

V. Experimental Evaluation

A. Experimental Setup

We have implemented our techniques over the REACT middleware and tested it on the PlanetLab [17] testbed. Our system was implemented in Java6u26 with approximately 2K lines of code.

The experimental evaluation focuses on the following parameters: (i) **WBGM efficiency**, (ii) **Quality of Results**, (iii) **Deadline misses**, (iv) **Average Execution Time**, (v) **Scalability**.

B. Evaluation of the WBGM Matching

In this set of experiments we evaluate the performance of our matching solution. We compare our own REACT matching method with two other WBGM algorithms: (i) the Metropolis Algorithm [23] and (ii) a Greedy algorithm.

Metropolis Algorithm: The basic idea of this algorithm is to apply a Markov chain Monte Carlo method on the maximum weighted bipartite graph matching problem. Their solution, like our own, is based on the acceptance of a new state and on the decision of whether the new state is better. However, we consider the states differently and a major difference among our algorithm and the Metropolis is that they do not consider the case for $g(x') = 0$ at all. Moreover, our goal is more than providing a matching algorithm but also to meet real time constraints in an online system.

Greedy Algorithm: We also compare our algorithm with a Greedy one, that produces high quality results since it

selects the greatest valued edge for every unassigned task out of the available workers. Thus, the basic idea of the Greedy matching is to select the edge $(worker_i, task_j)$ for any unassigned $task_j \in V$ with the highest weight w_{ij}, that is subject to the constraints defined for the WBGM. The complexity of such an approach is $O(VE)$ since for every task it needs to iterate through the edges and check its weight with all of the available workers, to find the highest one.

Comparison Among the Algorithms: In the first set of experiments we compare our REACT matching algorithm against the Greedy and the Metropolis one. Firstly, we initiate 1000 workers and we match them with a number of tasks that range from 1 to 1000, to prove the benefits of our algorithm. We use a full graph where all the tasks are connected with edges with every worker, which is the worst case scenario for the WBGM algorithms. We present the outcome in Figures 3 and 4.

In Figure 3 we illustrate the execution time for the assignment of the tasks. As can be observed, the Greedy matching takes a lot of time to execute when the size of the graph increases due to its complexity. Thus, for 1000 tasks it takes 99.7 seconds while the REACT and Metropolis algorithm with 1000 cycles execute for 12 seconds and with 3000 cycles they both need 45 seconds.

Figure 4 illustrates the output for the function that we aim to maximize. For the current experiment, we use weights on the edges in the range between $[0, 1]$ and thus the output of our function for the optimal matching cannot exceed the maximum number of tasks, due to the 1-to-1 selection. As can be seen, the Greedy succeeds an almost optimal behavior because we use a full graph and thus while there are a lot of workers that have not been assigned with tasks, it can find workers with weights on their joint edge that are close to 1.0. However, the respective time for these results is unacceptable for a real-time system, where we expect that the size of the graph can be even larger. It is obvious in this experiment, as the number of tasks increases, the REACT algorithm cannot achieve the same performance. This happens since the number of cycles that remains the same, eventually becomes insufficient for the amount of tasks and workers, and thus some of the tasks might not be able to be assigned with a worker during the iterations. Hence, both the Metropolis and the REACT algorithm need to have the cycles adjusted carefully to maximize the benefit of the trade-off among the time needed and the respective output. However, it is important to observe that although Metropolis and REACT algorithms needed almost the same time to execute, for the same cycle parameter, the REACT algorithm results on a higher output even with a third of the cycles.

Discussion: In order to use the REACT and Metropolis Matching algorithms efficiently, it is essential that a minimum number of unassigned tasks needs to be gathered before executing the algorithm. Otherwise, the algorithm is going to run through cycles without improving the matching output and wasting resources. On the contrary, the Greedy one can be either triggered for each unassigned task or wait for a number of tasks. However, the REACT and Metropolis algorithms are far less time expensive and the REACT algorithm produces results with increased weight than the Metropolis one at all times.

C. Evaluation of the REACT Approach

In the second set of experiments we demonstrate the benefit of our REACT algorithm, in a crowdsourcing system.

Case Study. Although, we tried to obtain real workloads from existing crowdsourcing platforms such as AMT, to use them in our experiments, in the current setting the systems do not allow us to control the task assignment.

Thus, we carried out a study on CrowdFlower, that exploits the AMT platform, in order to understand their behavior and determine the deadlines, response times and accuracy to use in our system. The study was conducted with the following use case. We injected crowdsourcing tasks where the users were asked to estimate the traffic in specific geographic regions, based on the Google maps traffic layers. During our study we noticed the following: although the first couple of results arrived in only a few seconds, the remaining tasks could take up to 6 hours to complete, which is unacceptable for real-time systems. We used the observations and results from this case study in order to define the values of the parameters in the remaining experiment. We extracted the trust field from the results, to determine the workers accuracy, and we computed that 70% of the workers had a feedback that was more than 50%. Moreover, 50% of the responses were received in less than 20 seconds, although our proposed time was 20 seconds, so we concluded that this was a sufficient time for someone to complete our task. However, the remaining of the users might delay to process or to select the task, since they may not be familiar with such a task, and thus the completion time could take up to hours. Hence, we decided to set the deadline for such a task among 60-120 seconds.

Experimental Setup. For the purposes of our experiments we create a number of workers that receive tasks from the system and process them among a time interval that is randomly decided based on their profile and ranges from a minimum to a maximum time. Although each worker receives a unique minimum and maximum time these times are constrained among 1-20 seconds, as this was the proposed time that we had set in our tasks in Crowdflower. However, a worker might choose to delay or abandon the task randomly with a probability of 50% and thus the executing time may reach up to 130 seconds. Moreover, based on the case study, each worker has a unique feedback $\in [0, 1]$ assigned with a distribution where the 70% of the workers receive a feedback that is above 0.50. Each task also receives a deadline from 60-120 seconds, since its production, that is in fact a tight

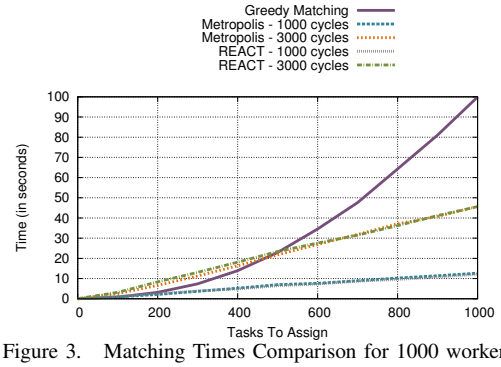

Figure 3. Matching Times Comparison for 1000 workers.

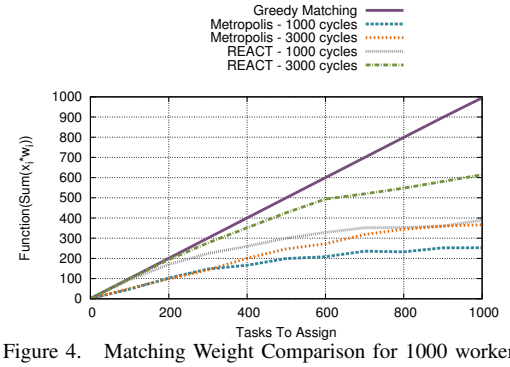

Figure 4. Matching Weight Comparison for 1000 workers.

deadline for such systems, to be able to satisfy applications that need quick responses such as traffic information.

We compare our approach with the case where the WBGM is being done with the Greedy matching and with a Traditional approach. When we use the Greedy matching we also use the online probabilistic model to reassign the tasks, as in the REACT algorithm. However, in the traditional approach we try to simulate the traditional non real-time crowdsourcing systems, such as the AMT. Hence, we use uniform matching for the assignment and the probabilistic model that we developed is not being used. We have defined the threshold of the probabilistic model to 10%. Thus if the probability that the task will be processed before the deadline from the currently assigned worker gets lower than 0.1 the task is being sent to the Scheduling Component to be reassigned. In the REACT matching algorithm we have set the number of cycles to 1000.

In order to conduct the experiment we consider only one region server with 750 online workers. The system receives tasks in a rate of 9.375 tasks/second that need to be assigned to the workers and all of the matching algorithms are triggered when the unassigned tasks (both the newly arrived and the abandoned by the workers) are more than 10. The rate of the incoming tasks was selected to be higher than the rate of the AMT [24], for each of our servers. We should also note that in all the experiments, the reassignment of the tasks based on the probabilistic model needs at least 3 completed tasks in the worker's profile to be initiated. Hence, the first 3 tasks in every worker are not going to be reassigned so as to train the system about his performance.

Figure 5 presents the amount of tasks that were finished before the deadline to the total amount of received tasks. As can be observed the REACT algorithm approach has the best behavior over the 3 techniques. As the figure shows, it manages to process 6091 out of the 8371 tasks with a total execution time that was lower than the deadline. Moreover, the majority of the missed deadlines is observed before the needed tasks for the system training have been completed, since most of the deadlines where lost at the beginning of the experiment and thus they expired before the probabilistic model was initiated. We can also observe that when we used the Greedy algorithm, the curve is rising for the first 4200 tasks and then it starts missing a lot of deadlines. This happens due to the fact that the Greedy algorithm, as shown in the previous set of experiments, takes a lot of time to execute when the size of the graph increases. Thus, in the current rate and while the tasks keep on coming the matching takes too long, causing a lot of queueing for the tasks that need to be processed. Hence, when the tasks are eventually assigned to a worker they have already expired. Finally, the traditional approach managed to process only 4264 tasks before their expiration and it is obvious that the difference with our approach would be increased constantly afterwards, as the REACT algorithm would miss only a few deadlines.

In Figure 6 we can observe the amount of positive feedbacks for the tasks. The feedback is decided when a task is finished and it is positive only if the task finished before the deadline, with a probability that is defined from the worker's unique feedback percentage. As the figure illustrates, our technique that uses the worker's profile to decide for the ones to execute the task managed to have 4941 positive feedbacks, while the traditional approach had only 3066. Thus, we infer that the highest percentage of positive feedback compared to the tasks that were under deadline is due to the WBGM that we introduced. This happens due to the fact that selecting "good" workers even with a non optimal matching, results on a higher quality output and it proves that we can achieve the goal of excluding low-quality workers from the system on real-time. Providing a highest cycle size to the REACT approach would result on even higher improvement, although it would be more time expensive. The curve of the approach for the Greedy matching is similar to Figure 5, since the tasks that lost their deadlines receive a negative feedback as well.

Finally, we present the average execution times per worker and the average of the total execution times for each technique in Figures 7, 8. In Figure 7 we report the execution times of the workers that executed the tasks. As can be seen, our approach has a short execution time per worker because it can quickly estimate whether the worker will delay the execution of the task and the reassignment selects workers with faster execution times. This graph includes only the final worker that finished the task, if the worker was previously assigned to another worker. Thus, we infer

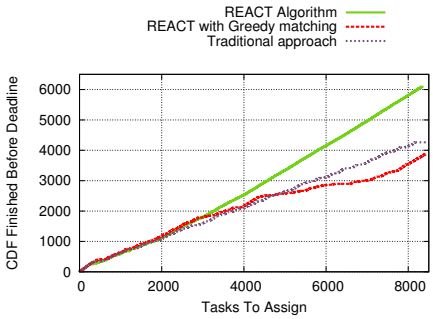

Figure 5. CDF of the Tasks executed before the Deadline.

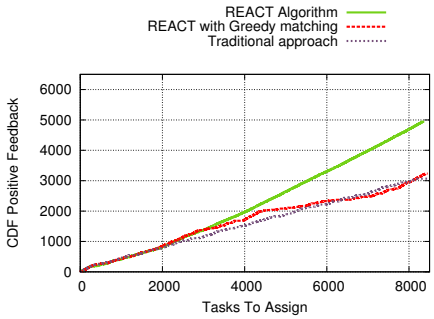

Figure 6. CDF of the Tasks with a Positive Feedback.

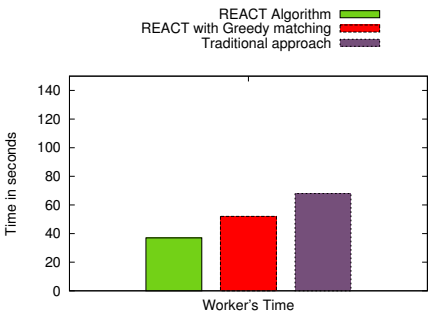

Figure 7. Workers Average Execution Times.

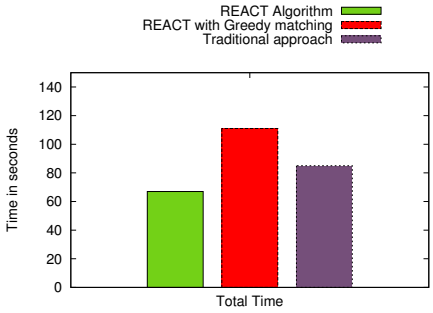

Figure 8. Average Total Execution Times.

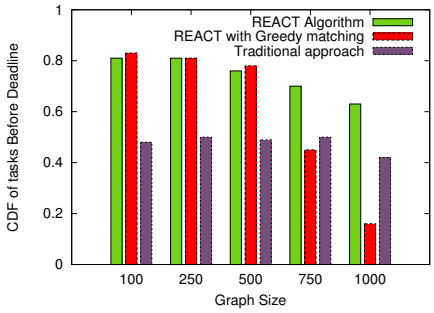

Figure 9. Scalability - Tasks executed before the Deadline.

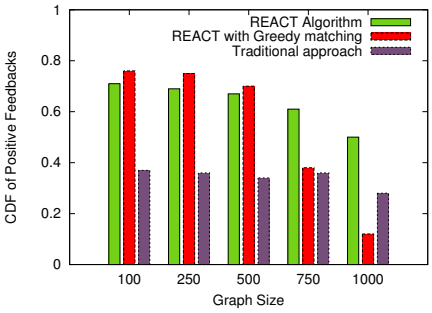

Figure 10. Scalability - Tasks with a Positive Feedback.

that using our model for reassigning the task improved the execution times for the case where the workers decided to delay the execution. The Greedy approach needed more time than REACT, due to the queueing and the reason is that although the Greedy uses the probabilistic model as well, when the queueing forces the tasks to miss their deadlines, they are executed in the time that the worker decides. This happens since there is no worker that will have a better probability to finish the task before deadline when it has already expired and thus there will be no reassignment in that case. The traditional approach, as expected, has the worst average execution times since it does not react when the user delays a task.

Figure 8 illustrates the average total execution time, including the time needed for potential task assignments. What is interesting to observe here is that although our technique reassigns the tasks to other workers, opposed to the traditional technique, it manages to process them faster than the traditional technique. This happens because our model can decide quickly whether the worker will execute the task before the deadline. In addition, it proves that the overhead of the probabilistic model and for the WBGM did not affect the system's performance. Moreover, it can be observed that queueing forced the Greedy approach to result high average execution times.

D. Scalability

In the third set of experiments we conduct an experiment with the same scenario as in the second set, however we use different size of graphs (in terms of workers and tasks) and a different rate of incoming tasks so as to observe the behavior of our approach.

In this set we are trying to "stress" the approaches of our system to better observe their behavior. Thus, we use the same scenario as before with a different graph size and rates. We use a graph size of 100, 250, 500, 750 and 1000 workers and the tasks are received with a rate of 1.5, 3.125, 6.25, 9.375 and 12.5 tasks per second respectively.

We would like to point out that we set our scalability rates (12.5 HITs/sec) even higher than the AMT rates. According to [24] there were fewer than 18K HITs on AMT on the 25/7/12. With our rate (12.5 HITs/sec) there are 45K HITs per hour at every server.

Figure 9 illustrates the percentage of tasks that were processed before the deadline for all the approaches on the different graph sizes. The REACT approach seems to be a little influenced as the graph size increases, however it was expected that forcing the approaches to work on higher rates and sizes with the same deadlines would decrease their performance. On the contrary, the traditional approach that only needs to define an available worker for each task seems to be influenced only by the size of 1000 workers. We can also observe that the Greedy algorithm had a better performance for the size of 100 than REACT, but as the size increases and queueing occurs it starts to miss too many deadlines and it is very interesting that for the size of 1000 workers it only managed to have 16% of the tasks completed before the deadline.

In figure 10 we can observe the percentage of the positive feedbacks that the tasks received as the graph size increases.

The graph seems to be proportional to figure 9 for all approaches. The REACT approach decreases its feedback accuracy as the size increases. This, can also be denoted to the fact that the cycle size was 1000 for all the experiments. Hence, while the amount of the edges increases along with the graph, the output would not be similarly maximized if the number of cycles remains the same. The traditional approach was not influenced again from the increased graph size. Finally, the Greedy matching approach reduced its feedback percentage along with the missed deadlines. Although it seems that the Greedy approach has a better performance than REACT when the graph size is low, as was concluded from the first set of experiments as well, the overhead for coping with large graph sizes is high. On the contrary REACT was also influenced by the increased size but not to such an extent.

We have tried to run scalability experiments on larger size graphs and higher rates. However, in these conditions we observed that the task assignment process cannot be sustained by the system. Although our approach still performs better than the others, the system gets overloaded and as a result the assignment of the tasks to the workers takes time. That fact leads to inevitable missed deadlines and queueing for all the approaches. One possible solution for that problem is to split the regions so that each of the servers would contain sufficient workers and tasks without being overloaded.

VI. Related Work

Crowdsourcing systems have recently become extremely popular and a number of commercial crowdsourcing systems have emerged, including AMT [3], CrowdFlower [4], CloudCrowd [5] and microWorkers [6]. Additionally, applications have began to utilize crowdsourcing systems *e.g.,* for labelling images such as the work of Sorokin et al. in [25] where they show that using AMT for image annotation is a quick way to annotate large image databases.

A number of works have investigated the assignment of crowdsourcing tasks to worker nodes. Authors in [26] propose a task assignment model for crowdsourcing markets where the worker's skill is unknown. Their approach is similar to a simplified version of the well-studied online adwords problem. Although their approach manages to discover workers who are appropriate for the tasks, they do not consider any QoS or real-time constraints for the tasks.

Alt *et al* in [20] consider the problem of providing location-based tasks to the mobile crowd and implement a prototype as a use case. Although they claim that tasks in such systems can be time-critical and solutions need to be submitted within a predefined time period, they do not propose a solution for this problem, as we do with our approach. Moreover, their approach works in the same manner with the traditional crowdsourcing systems where the workers decide for their tasks. Their prototype is implemented on mobile phones and the workers can take advantage of the GPS location in order to find nearby tasks. In our approach, assigning tasks based on the location can be achieved by considering the distance among the task and the worker in the weight function.

Authors in [27] consider a general model of crowdsourcing tasks and pose the problem of minimizing the total price, in terms of the number of assignments, that must be paid to achieve a target overall reliability. They propose an algorithm to decide which tasks to assign to which workers and to infer the correct answers for each tasks from the workers' answers. Their algorithm is inspired by belief propagation and low-rank matrix approximation and they show that it outperforms majority voting. Similarly, in [28], they propose a two-phase technique, based on voting, in order to maximize the accuracy of the results by selecting multiple users to process them dynamically, while minimizing the total budget needed to achieve that accuracy. However, our technique manages to define the most suitable workers before the execution of the tasks and thus to reduce the cost of the multiple assignments.

Khazankin *et al* in [21] propose a crowdsourcing model, where they assign the tasks based on the worker's "quality" for a specific task category and they suggest that the task assignment should be done based on the worker's availability to overcome deadline issues. Our approach improves that since we take the individual user profile into consideration to prevent deadline misses. Moreover, our technique uses real time measurements to predict a deadline miss and reassigns the tasks to improve efficiency.

Authors in [16] propose an image search engine whose results are being validated using crowdsourcing. After retrieving a set of similar images based on the image features, they send them as a task to the Amazon's Mechanical Turk and expect from a number of workers to validate if they are similar to the original one. They use a probability model to predict the needed delay to receive a number of responses on runtime. Furthermore, they try to predict the result of the majority vote from the responses that will be received by all workers. In contrast to this work, our goal is to predict whether the response will be received before the deadline to reassign the task if needed. Another system that uses human input via crowdsourcing on a traditionally computer-based mechanism such as a Database, is CrowdDB [29], where authors propose a scheme to process queries that neither database systems nor search engines can adequately answer.

One of our main contributions in this paper is to apply real-time requirements in the crowdsourcing platforms. While, there are several studies to understand the quality of results from AMT solvers [30], we are not aware of any other work that attempts to provide Quality-of-Service, in terms of real-time requirements, in such systems.

A few approaches exist in the literature that solve the problem of the maximum/minimum weighted bipartite graph matching. However, most of the algorithms that offer optimal

solutions such as [31], [9] are used offline and need a lot of time to execute due to their complexity. The Metropolis Matching algorithm that was discussed earlier is presented in [23]. Their approach is based on the "Metropolis Algorithm". They consider a selection of edges as a state and for every iteration they choose randomly a different set of edges which is selected as a new state, if it provides a better output than the previous one. Their technique provides results quickly however the results are not optimal. As we proved, our algorithm outperforms the Metropolis algorithm since it produces better results at almost the same time.

VII. Conclusions

In this paper we have presented REACT, a system that aims to exploit the wisdom of the crowd to efficiently schedule tasks to Crowd workers under time constraints. REACT makes it easy to define the most appropriate workers out of the crowd to process a task in real time. We propose a novel weighted bipartite graph matching approach that is able to define an approximate matching rapidly and we consider real-time constraints in the execution of the tasks, using a probabilistic method. The advantage of our technique over the existing task assignments methods for crowdsourcing, is that it considers real-time constraints and it manages to define the most suitable workers for each task. Detailed experimental results over PlanetLab illustrate that our approach is practical, efficient and depicts good performance. For our future work, we plan to look at the working of our assignment process during overloading conditions to further improve the performance of our approach.

Acknowledgment

This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the NSRF - Research Funding Program:Thalis-DISFER, Aristeia-MMD and the FP7 INSIGHT project.

References

[1] D. C. Brabham, "Crowdsourcing as a model for problem solving," *Convergence*, vol. 14, no. 1, pp. 75–90, Feb. 2008.

[2] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Commun. ACM*, vol. 54, no. 4, pp. 86–96, Apr. 2011.

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

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

[5] "Crowdcloud," http://www.crowdcloud.com/.

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

[7] "Gigwalk," http://gigwalk.com/.

[8] "Field agent," http://www.fieldagent.net/.

[9] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, pp. 83–97, 1955.

[10] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani, "Adwords and generalized online matching," *J. ACM*, vol. 54, no. 5, 2007.

[11] J. Feldman, M. Henzinger, N. Korula, V. S. Mirrokni, and C. Stein, "Online stochastic ad allocation: Efficiency and fairness," *CoRR*, vol. abs/1001.5076, 2010.

[12] S. Penmatsa and A. T. Chronopoulos, "Dynamic multi-user load balancing in distributed systems," in *IPDPS*, Long Beach, California, USA, March 2007, pp. 1–10.

[13] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 485–497, April 2007.

[14] Y. Drougas and V. Kalogeraki, "Distributed, reliable restoration techniques using wireless sensor devices," in *IPDPS*, Long Beach, California, USA, March 2007, pp. 1–10.

[15] Y. Drougas and V. Kalogeraki, "Accommodating bursts in distributed stream processing systems," in *IPDPS*, Rome, Italy, May 2009, pp. 1–11.

[16] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones," in *MobiSys*, San Francisco, CA, USA, June 2010.

[17] PlanetLab Consortium, "http://www.planet-lab.org," 2004.

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

[19] 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.

[20] 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.

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

[22] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, no. 4, pp. 661–703, Nov. 2009.

[23] H.-P. Shih, "Two algorithms for maximum and minimum weighted bipartite matching," Master's thesis, National Taiwan University, Republic of China, 2008.

[24] "Mechanical turk tracker," July 2012. [Online]. Available: http://mturk-tracker.com/arrivals/

[25] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *Computer Vision and Pattern Recognition Workshops*, Anchorage, Alaska, USA, June 2008.

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

[27] D. R. Karger, S. Oh, and D. Shah, "Budget-optimal task allocation for reliable crowdsourcing systems," *CoRR*, vol. abs/1110.3564, November 2011.

[28] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "Cdas: A crowdsourcing data analytics system," *CoRR*, vol. abs/1207.0143, 2012.

[29] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: answering queries with crowdsourcing," in *SIGMOD*, Athens, Greece, June 2011.

[30] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with mechanical turk," in *CHI*, Florence, Italy, April 2008.

[31] S. Khuller, S. G. Mitchell, and V. V. Vazirani, "On-line algorithms for weighted bipartite matching and stable marriages," *Theor. Comput. Sci.*, vol. 127, no. 2, pp. 255–267, 1994.