CrowdService: Optimizing Mobile Crowdsourcing and Service Composition

XIN PENG, School of Computer Science, Fudan University, China; Shanghai Key Laboratory of Data Science, Fudan University, China
JINGXIAO GU, School of Computer Science, Fudan University, China; Shanghai Key Laboratory of Data Science, Fudan University, China
TIAN HUAT TAN, Acronis, Singapore
JUN SUN, Singapore University of Technology and Design, Singapore
YIJUN YU, Department of Computing and Communications, The Open University, UK
BASHAR NUSEIBEH, Department of Computing and Communications, The Open University, UK; Lero - The Irish Software Research Centre, University of Limerick, Limerick, Ireland
WENYUN ZHAO, School of Computer Science, Fudan University, China; Shanghai Key Laboratory of Data Science, Fudan University, China

Some user needs can only be met by leveraging the capabilities of others to undertake particular tasks that require intelligence and labor. Crowdsourcing such capabilities is one way to achieve this. But providing a service that leverages crowd intelligence and labor is a challenge, since various factors need to be considered to enable reliable service provisioning. For example, the selection of an optimal set of workers from those who bid to perform a task needs to be made based on their reliability, expected reward, and distance to the target locations. Moreover, for an application involving multiple services, the overall cost and time constraints must be optimally allocated to each involved service. In this paper, we develop a framework, named CROWDSERVICE, which supplies crowd intelligence and labor as publicly accessible crowd services via mobile crowdsourcing. The paper extends our earlier work by providing an approach for constraints synthesis and worker selection. It employs a genetic algorithm to dynamically synthesize and update near-optimal cost and time constraints for each crowd service involved in a composite service, and selects a near-optimal set of workers for each crowd service to be executed. We implement the proposed framework on Android platforms, and evaluate its effectiveness, scalability and usability in both experimental and user studies.

CCS Concepts: •**Information systems** → **Crowdsourcing;** *Collaborative and social computing systems and tools;* •**Human-centered computing** → **Collaborative and social computing systems and tools;** *Mobile computing;*

Additional Key Words and Phrases: mobile crowdsourcing, collaboration, service composition, reliability

1. INTRODUCTION

A variety of user needs nowadays can be automated by calling *computational services*, either remotely through a Web service or locally through a mobile application. These services are used to construct personal applications using lightweight service composition techniques such as mashup [Liu et al. 2007; Maximilien et al. 2008; Rosenberg et al. 2008]. However, not all user needs can be accomplished only by computational services. For example, one may want to check the validity of the description of a secondhand laptop by site inspection, and assess its market value by consulting experts. Beyond pure computational services and their compositions, such needs can

X. Peng is the corresponding author. This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203. It is also supported, in part, by SFI Grant 13/RC/2094, ERC Advanced Grant 291652 (ASAP), and the UK EPSRC. Author's addresses: X. Peng and J. Gu and W. Zhao, School of Computer Science, Fudan University, Shanghai, China; T. Tan and J. Sun, Singapore University of Technology and Design, Singapore; Y. Yu and B. Nuseibeh, Department of Computing and Communications, The Open University, UK.

DOI: http://dx.doi.org/10.1145/0000000.0000000

only be met by leveraging the intelligence and labor of others, e.g., those who are currently near to the laptop or have the required expertise.

An emerging way of involving humans in information seeking and computation tasks is *crowdsourcing*, which enables a crowd of undefined size to be engaged in solving a complex problem through an open call [Bozzon et al. 2013; Chatzimilioudis et al. 2012]. Existing crowdsourcing platforms such as Amazon Mechanical Turk (AMT) [1] allow a requester to hire a large number of workers from Web-based online community to accomplish short and self-contained microtasks such as tagging images or translating fragments of text. Web-based crowdsourcing, however, cannot support location-based crowdsourcing tasks such as site inspection, which require the workers to use mobile devices to enable location-based worker selection.

To compose crowdsourcing tasks into user applications, additionally, it is required that crowd intelligence and labor themselves be encapsulated and served as services, which could be invoked in a similar way as computational services. Each time such a service is invoked, a set of workers are selected to participate the crowdsourcing task and each of them gets instructions for his/her own microtasks. After accomplishing their work, the results they returned are aggregated to generate the output of the service invocation.

In this paper, we develop a framework named CROWDSERVICE which supplies crowd intelligence and labor as publicly accessible *crowd services*. A crowd service satisfies specific individual's needs via mobile crowdsourcing and can be composed with other crowd services and computational services. For each invocation of a crowd service, CROWDSERVICE launches a crowdsourcing task and aggregates the results returned by workers into the output. To the best of our knowledge, we are the first to investigate service composition that includes human services provided by mobile crowdsourcing.

Our earlier work [Peng et al. 2016] presented basic concepts and framework of crowd services. However, given the aim of crowd service composition, a technical challenge lies in how to ensure reliable service provisioning by selecting an optimal set of workers for each involved crowd service. The reliability and cost of a crowd service depend highly on how many potential workers are available and how close they are to the target locations and how much reward they expect to receive. Unlike classical service composition techniques where exactly one concrete service is selected for each abstract service, in crowd service composition a consumer may expect that multiple workers can finish their work and return their results in time. And to ensure a high probability of success the platform often needs to employ even more workers from those who bid for each task. Under given certain resource (i.e., cost and time) constraints, however, the execution can only employ a limited number of workers.

Furthermore, to maximize the reliability of a composite service, it is required that the overall resources be optimally allocated to each involved computational service and crowd service. To ensure this reliability, CROWDSERVICE actively estimates the likelihood of success in providing the service based on parameters associated with a set of potential workers, and dynamically synthesizes and updates the constraints on each involved computational or crowd service. When a crowd service is to be executed, CROWDSERVICE generates an open call to potential workers and selects a near-optimal set of responded workers based on the synthesized constraints to maximize the probability of success.

The contributions of this paper are as follows: 1) we introduce the concept of crowd service and propose a framework that can supply crowd intelligence and labor as publicly accessible crowd services; 2) we develop an approach based on genetic algorithms [Forrest et al. 1993] that can synthesize near-optimal cost and time constraints for each crowd service involved in a composite service and select a near-optimal set of workers for a to-be-executed crowd service (Notice that generating an optimal set of workers is NP-hard in the number of workers); 3) we implement the proposed framework and algorithms on Android platforms and evaluate the effectiveness, scalability and usability with experimental and user studies.

[1] Amazon Mechanical Turk: http://www.mturk.com

2. MOTIVATING EXAMPLE

Suppose that Bob would like to buy a secondhand laptop from an online market, which allows registered users to sell personal items and supports online transactions only. Following the online shopping process, Bob would first search for a desired laptop, examine its detailed information, purchase it, and finally make the payment. However, Bob is worried that the description of the laptop may be fraud. Furthermore, he is afraid that the price set by the seller might be unreasonably high. Thus, Bob would like to find someone to check the validity of the description by site inspection and take a picture of the laptop before he decides to buy it. Moreover, Bob wants to consult local experts on the market value of the laptop. If the price set by the seller is much higher than the price assessed by the experts, he would abort the transaction.

The above process could be accomplished by a series of activities. Apart from automated computational services (e.g., a Web service for online bank transaction), the crowd services have to be employed for site inspection and price assessment. We argue that in order to fully automate the above process (and its alike), one needs to combine the framework to compose computational services which has been investigated extensively [Alrifai and Risse 2009; Tan et al. 2013]) with crowd services. Our CROWDSERVICE framework can be used to address Bob's problem.

In the design of CROWDSERVICE, we maintain a set of workers, which can potentially provide crowd services. Each worker is associated with a set of attributes, e.g., his/her location, past service providing records, etc. For each invocation of a crowd service, CROWDSERVICE launches a crowdsourcing task and selects workers based on their attributes to accomplish the task. Then the selected workers accomplish their work and submit their results.

CROWDSERVICE provides composition templates of crowd and computational services for common needs, such as purchasing secondhand items. To compose crowd services with computational services, each crowd service is wrapped as a Web service and published on the platform in CROWDSERVICE. Developers of a crowd service need to define its input and output parameters and specify its execution strategy such as result aggregation method. For example, the price assessment service takes as input a series of descriptions and a picture of an item and returns as output an assessed price. The results of price assessment from multiple workers will be aggregated based on the specified aggregation method (e.g., by taking the average) to generate the final output.

Furthermore, Bob hopes that the entire process of purchasing secondhand laptops can be accomplished within 10 minutes and the overall additional cost does not exceed 10 dollars. To maximize the probability of completing Bob's composite service, given the above constraints, CROWDSERVICE needs to compute how much money and time should be spent on each involved computational or crowd service. The optimal allocation of cost and time constraints is complicated due to the dynamic and uncertain nature of crowd services: the number of potential workers who are near to the target locations (e.g., the seller's address for the site inspection service), their expected rewards and reliability are uncertain and dynamically changing. Intuitively, more cost and time need to be allocated to the crowd service that has fewer potential workers close to the target location.

When a crowd service is to be executed, ideally an optimal set of workers should be selected. For example, if the allocated cost and time constraints of the site inspection service are 8 dollars and 6 minutes respectively, and Bob hopes that at least two workers can return their results in time. This implies the optimization objective of maximizing the probability of at least two workers returning their results in 6 minutes while the money paid to all the selected workers does not exceed 8 dollars.

With a composite service and user specified constraints, CROWDSERVICE would first synthesize constraints on each involved computational service and crowd service and calculate the feasibility of composite service based on the likelihood of satisfying those constraints. Afterwards, CROWDSERVICE would automatically execute the composite service by invoking the services accordingly. Whenever a service is finished, the constraints are updated. When a crowd service is to be executed (e.g., site inspection of the laptop), CROWDSERVICE generates an open call to potential workers based on, in this example, their physical locations. After receiving feedback from the workers (e.g.,

whether a worker is willing to participate and for how much reward), CROWDSERVICE selects the workers based on the constraints and keeps executing the composite service until its completion.

3. CROWD SERVICE

In this section, we first describe the conceptual model of crowd service and then introduce different types of crowd services. Afterwards, we introduce the composition of crowd services.

3.1. Conceptual Model

A crowd service leverages human intelligence and labor via mobile crowdsourcing and is packaged in the form of computational service (e.g., Web service). It can be used for acquiring information (e.g., assessing the price of an item, querying availability of spaces/rooms in a specific building) or accomplishing real-life tasks (e.g., performing site inspection of an item, booking a table in a restaurant onsite).

The conceptual model of crowd services is shown in Figure 1. A crowd service declares zero or more input parameters, each of which specifies the name and type of an input value provided by the consumer, and one or more output parameters, each of which specifies the name and type of an output value returned to the consumer. A crowd service defines one or more task fields, each of which specifies the name and type of a result value provided by workers. Note that the output parameters of a crowd service can be different from its task fields. For example, a crowd service for price assessment has a personal price assessment and a confidence level as its task fields, but has only an assessed price as its output parameter, whose value will be calculated based on its task fields.

A crowd service has a text description specifying its task requirements for workers. For example, the text description of a crowd service for site inspection can be specified as follows.

> *Please get to the designated location and inspect the item specified below. Check the physical item and evaluate whether it is consistent with the item description given below. Take a picture of the item and upload the picture.*

This description and the input parameters such as seller address and item description constitute task instructions for workers.

Each invocation of a crowd service results in the execution of a crowdsourcing task; that is, a crowdsourcing task is an instantiation of a crowd service. It returns an output result which provides a value for each of the output parameters of the crowd service. Depending on the crowd service definition, a crowdsourcing task can be location independent, or be targeted at one or several locations.

A crowdsourcing task can be accomplished by one or more workers and a microtask is generated for each worker. In each microtask, the worker accomplishes the assigned work and returns a result which provides a value for each of the task fields of the crowd service. The results returned by the workers will be aggregated into the output result of the task.

A crowdsourcing task has three operational parameters: C specifies the maximal cost that can be spent on the task (i.e., cost constraint); T specifies the maximal time that can be spent on the task (i.e., time constraint); RN specifies the minimal number of workers who successfully return their results, which is specified by the consumer and reflects his/her expectation of necessary redundancy. For example, for a site inspection task, the consumer may expect that at least two workers return their results (i.e., $RN = 2$).

3.2. Types of Crowd Services

There are different types of crowd services. According to the content of service, a crowd service can be a *query service* or an *actionable service*. A query service requires workers to provide specific information without taking any other actions, while an actionable service requires workers to take

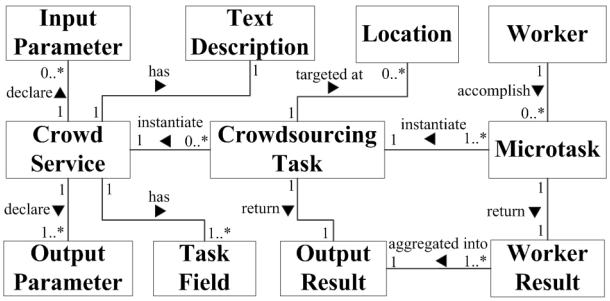


Fig. 1. Conceptual Model of Crowd Service

some real-life actions. For instance, a worker participating in an actionable task may need to move to a target location (e.g., a restaurant) and accomplish some social interactions (e.g., booking a table).

A crowd service can be *location based* or *location independent*. This dimension is orthogonal to the dimension of service content. A query service can be location based if the queried information is associated with a specific location, for example querying availability of spaces/rooms in a specific building. Otherwise, it is location independent, for example querying a reasonable price for a secondhand item. On the other hand, there are both location-based actionable services (e.g., booking a table in a restaurant without online booking or phone booking) and location-independent actionable services (e.g., paying a bill by third-party online payment).

3.3. Crowd Service Composition

Crowd services can be composed with computational services based on a predefined business process. A business process consists of a series of activities, which can be accomplished by different kinds of services such as follows.

— **Web Service (WS)**: standard Web services or RESTful Web services provided by remote servers;
— **App Service (APP)**: business services provided by mobile applications (e.g., Android App) which can provide complex user interaction and encapsulate access to sensors (e.g., camera, audio recorder) in mobile devices;
— **UI Service (UI)**: simple user interaction services for users to examine returned results, make choices, or input required information on their mobile devices.
— **Crowd Service (CS)**: human powered services accomplished via mobile crowdsourcing;

Among the service types, Web service, App service, and UI service are computational services. Figure 2 shows a business process for the composite service of buying secondhand items using an activity diagram. According to the process, a consumer first searches for and selects a desired secondhand item with an App service, which can be provided by a mobile client of a secondhand market. Based on the returned item and seller information, the process queries the price of the corresponding new product with a Web service and requests a site inspection of the item with a crowd service in parallel. Then the consumer examines the returned item information, site inspection results and product price and determines whether to continue. If the consumer chooses to continue, a crowd service is invoked to assess the price of the selected item. If the price set by the seller is lower than certain threshold (e.g., 1.1 times of the assessed price), an order is generated and submitted via a Web service.

A business process is implemented as a template that can be executed on mobile devices after being instantiated into a composite service by binding a concrete service for each activity. The process template specifies the interaction flow and parameter passing among different services. We assume that a process template can be defined by someone who knows well how to compose services for specific goals (e.g., buying secondhand items) and shared among the users who want to achieve the same goals. The CROWDSERVICE framework does not provide direct support for process template definition, but supports crowd service composition from two aspects: making crowd services

composable with each other as well as with computational services; synthesizing constraints and selecting workers for each crowd service involved in a composite service.

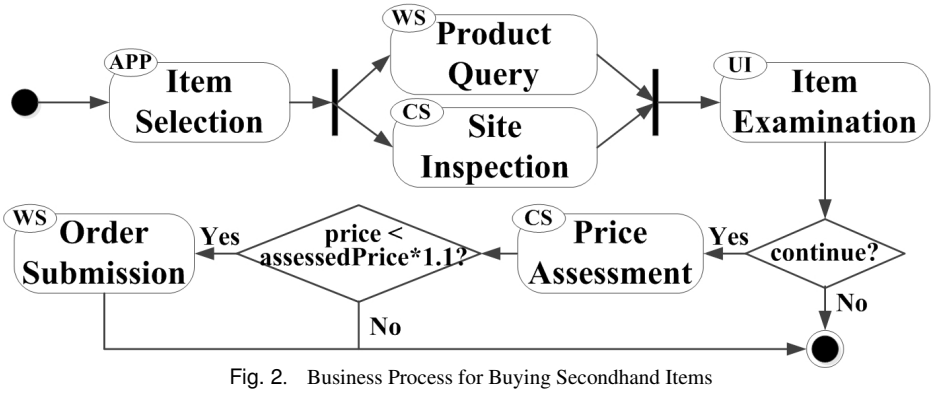


Fig. 2. Business Process for Buying Secondhand Items

4. FRAMEWORK

In this section, we first present an overview of CROWDSERVICE and then present details on the underlying techniques.

4.1. Overview

An overview of our agent-based crowd service framework is presented in Figure 3. It shows the software agents and other modules of the crowd service platform, consumer client, and worker client, together with the interactions between them.

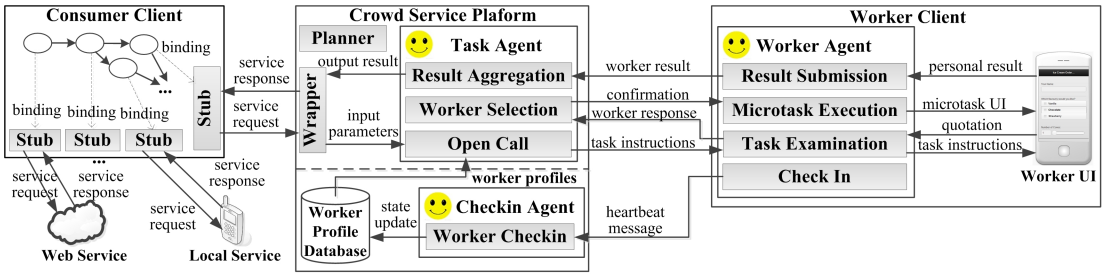


Fig. 3. Overview of Crowd Service Framework

There are two major challenges for the crowd service platform. One challenge is the transformation between service invocation and crowdsourcing task execution. From the external view of a crowd service, it is a publicly accessible Web service that can be invoked through standard service interfaces. From the internal view of a crowd service, it is a crowdsourcing task that involves the individual behaviors of a number of crowd workers. To bridge these two views, CROWDSERVICE employs an agent-based architecture, which assigns a task agent for each crowd service invocation to manage the task execution and generate the output result. This enables CROWDSERVICE to take advantage of the techniques of scalable architecture in cloud computing such as load balancing and container, which are popularly used in Internet services. The other challenge is the customization of crowdsourcing task execution for different requirements. As a genetic platform, CROWDSERVICE needs to support different kinds of crowdsourcing tasks for crowd services defined for different requirements. To this end, CROWDSERVICE provides customizable microtask execution and result aggregation based on crowd service description. Customizable microtask execution automatically generates task instructions and interaction forms on worker clients to guide the workers to accomplish assigned microtasks. Customizable result aggregation automatically generates output results

of crowd service invocations from personal results submitted by workers based on predefined aggregation policies.

Basic information of registered workers such as age, sex, and ability is stored in the worker profile database on the platform. This information is provided by the workers when they register. There is also a reliability of each worker in the profile database, which reflects the likelihood that the worker will accomplish a given task in time. It can be calculated as the percentage of finishing the assigned work and returning the results in time based on past service providing records of each worker. In order to report availability and update the real-time state (e.g., location), the worker agent on the mobile device of a worker periodically (e.g., once 2 minutes) sends heartbeat messages to the checkin agent on the platform. The checkin agent updates the real-time states of available workers in the worker profile database. Sending heartbeat messages means that a worker shares his/her location with the platform, thus raising the issue of location privacy. To protect privacy, a worker can control when to report availability to the platform or choose to send heartbeat messages without location information (which means only participating in location-independent tasks). Moreover, privacy policies, regulatory strategies, and computational algorithms (e.g., anonymity and obfuscation) [Krumm 2009; Yang et al. 2014; Barhamgi et al. 2016] could be used for protecting privacy data, which will be employed in the future work.

To execute a composite service involving crowd services on a mobile device, the consumer needs to specify the overall cost and time constraints of the composite service and the acceptable minimal number of successfully returned worker results (i.e., RN) for each involved crowd service. The execution engine on consumer client executes a composite service by invoking the crowd services and computational services involved in it.

Each time right before a crowd service is invoked, the execution engine requests an execution plan for the remainder of the composite service. The planner in CROWDSERVICE executes a planning process to produce an optimized execution plan, which allocates the resources (response time and cost) to each unexecuted crowd service or computational service and estimates the feasibility (probability of success) of the composite service (see Section 5). If the estimated feasibility is lower than the threshold (e.g., 60%) specified by the consumer, the execution engine terminates the execution and reports a failure to the consumer. Otherwise, the execution engine sends a request of the current crowd service with the allocated resources (i.e., cost and time) to the platform and gets a service response from it.

Each time a crowd service request is received, the crowd service wrapper on the platform creates a task agent and assigns the service request to it. The task agent gets input parameters from and returns an output result to the wrapper. It executes a series of behaviors to accomplish the assigned task (i.e., crowd service invocation).

— **Open Call**: The task agent sends an open call with task instructions (including task description, target locations and input parameters) to all the available candidate workers who may satisfy specific conditions (e.g., with the required capabilities or near the target locations).
— **Worker Selection**: The task agent receives responses from workers who are willing to participate and selects a near-optimal subset of workers based on the cost and time constraints (see Section 5). After that it sends a participation confirmation to each selected worker.
— **Result Aggregation**: The task agent receives results from workers and aggregates all the received results to generate an output result (see Section 4.4).

Correspondingly, the worker agent on the mobile device of a worker executes a series of behaviors to communicate with the task agent, and guides the worker to accomplish his/her microtask.

— **Task Examination**: The worker agent receives an open call from the platform and presents the task instructions on the worker's UI (User Interface). The worker examines the task instructions and inputs an offer (i.e., expected reward). Then the worker agent sends a response to the platform.
— **Microtask Execution**: The worker agent generates a microtask view presenting task instructions and an interaction form to capture the result. The worker accomplishes the microtask and returns

his/her result in the interaction form, for example, by typing in a text input, choosing in a drop-down box, or taking a picture, etc. (see Section 4.3).
— **Result Submission**: The worker agent sends the worker's result to the platform after he/she finishes the microtask.

4.2. Crowd Service Description

To develop and publish a crowd service, a developer only needs to specify a crowd service description and deploy it on the platform. A crowd service description includes input/output parameters, location information, task fields, and an aggregation method. Based on this description, the platform can automatically generate a Web service with the specified input/output parameters and execute the specified execution strategy when the service is invoked.

The description of each input/output parameter includes its name and type. A parameter can be of primitive data type (e.g., string, integer) or multimedia data type (e.g., image, audio).

Location information specifies whether a crowd service is location dependent or not and, if so, the way to get the target locations (current location, direct input, input transformation). Current location means to use the current location of the consumer (e.g., when ordering food from restaurants nearby) as the target location. Direct input means to use a specified input parameter (e.g., the location of a seller for the site inspection service) as the target location. Input transformation means to transform a specified input parameter (e.g., the name of a restaurant) into target locations (e.g., the locations of its branches) using coordinate transformation services.

Task fields specify the schema of worker results provided by workers. The name and type of each task field is described in service description. The types of task fields include primitive data type, choice type, and multimedia data type. A primitive data type can be string, integer, or float. For a string field, its length needs to be specified. For a number field (integer or float), its range and precision need to be specified. A choice field provides predefined options for a field. A multimedia field can be an image, audio, or video that needs to be recorded by a worker using corresponding sensors (e.g., camera) on his/her mobile device. Task field definition of a crowd service is used as the schema of worker result representation and microtask view generation (see Section 4.3).

Aggregation method specifies how the value of each output parameter can be aggregated from the workers' results. CROWDSERVICE provides standard aggregation methods such as computing the average or taking the result of the highest ranked expert. For an output parameter using a standard aggregation method, the developer can simply choose the corresponding aggregation method provided by the template. In other cases, the developer can define his/her own aggregation method, which can be either a standard aggregation or a customized computation (see Section 4.4).

4.3. Microtask Execution

For each assigned microtask, a worker agent generates a microtask view on the worker UI, showing task instructions and an interaction form to guide the worker to accomplish the microtask.

Task instructions consist of a task description, an optional task map, and an input value panel. The task description is the text description of the corresponding crowd service. For a location-based task, the task map shows the target locations on an online map (e.g., Google Maps), which can be used for navigating to the target locations. The input value panel shows the input values provided by the consumer client.

The interaction form allows the worker to input the required result after accomplishing the microtask. It is generated based on the task field definition of a crowd service. For each task field, a text label is generated to show its name and an input is generated based on its data type. For a primitive data type, a text input is generated for the worker to input a value. For a choice type, a drop-down box is generated for the worker to choose an option. For a multimedia data type, a button is generated, which if clicked will open a specific recorder (e.g., camera or voice recorder) on the worker's mobile device based on the media type (e.g., image or audio), with a multimedia container for preview.

Table I. A Virtual Table of Worker Results

Worker	Level	Time	AssessedPrice	Confidence
W1	9	2015-01-28 15:20:10	500	H
W2	7	2015-01-28 15:20:02	460	H
W3	7	2015-01-28 15:20:25	510	M
W4	5	2015-01-28 15:21:01	510	M
W5	6	2015-01-28 15:20:45	510	L

4.4. Result Aggregation

Worker results received from workers are stored in database and can be regarded as a virtual table. As an example, five worker results for the price assessment service are shown in Table I. In the table, each record corresponds to a worker result, including the worker, worker level, submission time, and values of task fields.

To aggregate worker results into an output result, task agent needs to generate a value for each output parameter. An output parameter can be generated based on worker results in two different ways, i.e., simple aggregation and complex computation.

Simple Aggregation. Simple aggregation generates an output value by using standard aggregation operations, which can be defined as an SQL query on the worker result table. It reports a failure if the query returns no results. For example, the following query specifies an aggregation method for output parameter $assessedPrice$, which computes the average price assessment returned by workers as the assessed price.

```
select avg(perAssessment) from $WorkerResult$
```

The following query defines a more complex aggregation method, which selects price assessments with the highest worker level from those with high confidence and computes their average.

```
select avg(perAssessment) from $WorkerResult$ T1 where confidence='H' and not exists (select * from $WorkerResult$ T2 where confidence='H' and T2.level¿T1.level)
```

Apart from returning a single value, simple aggregation can also return a list of values as an output value. For multimedia data, simple aggregation can be used to select the best result. For example, a picture that is submitted earliest or returned by a worker with the highest level. More complex aggregation of multimedia data have to be implemented by more complex computations.

Complex Computation. Complex computation generates an output value by using service-specific algorithm. For example, an algorithm can be used to select a picture of the best quality from all the submitted results based on service-specific criteria. This kind of computation can be implemented as service-specific plugin which can be registered to the crowd service platform and invoked by task agent for result aggregation.

5. CONSTRAINTS SYNTHESIS AND WORKER SELECTION

We propose an approach based on genetic algorithms [Forrest et al. 1993] for constraints synthesis and worker selection. It includes an algorithm that can be used with different settings. Each time right before a crowd service is invoked, the algorithm is executed to synthesize and update cost and time constraints for each unexecuted crowd service and computational service. When a crowd service is to be executed, the algorithm is executed again to select a near-optimal set of workers from those who bid for the current crowdsourcing task.

The algorithm takes the following parameters as input: the process model of a composite service consisting of crowd services and computational services; the cost and time constraints; a result constraint on minimal number of workers who successfully return their results for each crowd service;

a set of candidate workers for each crowd service and the cost, distance, and reliability of each candidate worker.

The outputs of the algorithm are a set of workers for each crowd service and the likelihood of successfully executing the composite service (hereafter reliability). We assume that each worker has a reliability which can be calculated based on past service providing records. For a newly joined worker without previous records, we assume a medium reliability (i.e., 50%). Without loss of generality, in this work, we assume that the cost and response time of a computational service (i.e., Web service, APP service, or UI service) are fixed and thus the reliability of a composite service is entirely on that of the involved crowd services.

For constraints synthesis, the input to the algorithm consists of the following: 1) Process model: all the unexecuted crowd services and computational services; 2) Cost and time constraints: consumer specified cost/time constraint minus the actual cost/time spent on all the executed services; 3) Candidate workers: all the available workers who are currently near to the target locations of each unexecuted crowd service; 4) Worker cost: an value estimated by the cost for walking to the target location (proportional to the distance) and the cost for finishing the assigned work after arriving at the target location (a fixed value for a crowd service). We remark the set of candidate workers is an over-approximation as not all those who could provide the service are willing to. The synthesized cost and time constraints of each crowd service can then be derived from the estimated cost and time of its selected workers.

For worker selection, the input to the algorithm consists of the following: 1) Process model: the crowd service that is to be executed; 2) Cost and time constraints: cost and time constraints allocated to the current crowd service; 3) Candidate workers: those who respond to the open call; 4) Worker cost: a worker's expected reward in his/her offer.

In this section, we first define the problem of constraints synthesis and worker selection, then present the genetic algorithm, and afterwards describe the calculation of reliability of workers.

5.1. Problem Definition

Assume there are n crowd services in the composite service *CS*. We use CR_i to denote the ith crowd service where $1 \leq i \leq n$, and use $CR_i.C$, $CR_i.R$ and $CR_i.T$ to denote the cost, reliability and response time of CR_i respectively. For each crowd service, without loss of generality, we assume all workers perform their works in parallel. Given a crowd service CR_i, suppose there are m candidate workers, we use W_i to denote the set of all candidate workers for CR_i, and w_{ij} to denote the jth worker of CR_i, where $1 \leq j \leq m$. We use $|W_i|$ to denote the total number of workers in W_i. We use RN_i to denote the result number constraint for CR_i. Our goal is to select a subset of candidate workers for each CR_i (denoted by $\widehat{W_i}$ where $\widehat{W_i} \subseteq W_i$), which would maximize the overall reliability of *CS*.

We use $w_{ij}.C$, $w_{ij}.R$, and $w_{ij}.D$ to denote the cost, reliability, and distance of worker w_{ij}. His/her response time $w_{ij}.T$ as shown in Equation 1 is divided into two parts: $w_{ij}.D/V$ is the walking time to the target location where V is a constant denoting human walking speed; T_w is a constant for a crowd service denoting the time for finishing the assigned work after arriving at the target location. The cost of CR_i is the sum of the cost of all selected workers $\widehat{W_i}$ as shown in Equation 2. The response time of CR_i as shown in Equation 3 takes the maximal response time of workers because all selected workers run in parallel. The reliability of CR_i is calculated as Equation 4, which is the probability of having at least RN_i workers returning their results.

$$w_{ij}.T = \frac{w_{ij}.D}{V} + T_w \tag{1}$$

$$CR_i.C = \sum_{w_{ij} \in \widehat{W_i}} w_{ij}.C \tag{2}$$

$$CR_i.T = \max_{w_{ij} \in \widehat{W_i}} w_{ij}.T \tag{3}$$

$$CR_i.R = \sum_{S_i \subseteq \widehat{W_i}} (\prod_{s_i \in S_i} (s_i.R) * \prod_{w_i \in \widehat{W_i}/S_i} (1 - w_i.R)) \tag{4}$$

where $|S_i| \geq RN_i$.

Similar to Web service composition, the Quality of Service (QoS) of a composite service can be aggregated from the QoS of its component services, based on the service's internal compositional structure (i.e., sequential, parallel, loop, or conditional) and the type of QoS attributes (i.e., cost, response time, or reliability). We refer interested readers to [Chen et al. 2013] for details of QoS aggregation.

We assume that a worker can not be involved in two or more tasks at the same time, which means that we do not need to consider the workload of workers. This limitation is reasonable in practice, since a user usually can only focus on one thing at a time. But in our future work we may consider to relax the limitation, for example, to allow a worker to accomplish a task on the way for another task.

We assume that the sets of workers for different crowd services are disjoint. This assumption is not that strict for candidate worker sets, since if a worker is included in the candidate workers of different crowd services, he/she gets a unique identity for each crowd service. We denote the disjoint union of all the candidate workers and selected workers for all crowd services as C_w and S_w respectively where $C_w = \biguplus_{i=1}^{n} W_i$ and $S_w = \biguplus_{i=1}^{n} \widehat{W_i}$. Suppose that the cost and time constraints of a composite service *CS* are C_G and T_G respectively. We use $CS(S_w)$ to denote the instance of *CS* where crowd services of *CS* only make use of workers given in S_w. The objective is to maximize the probability of success, which is shown as follows:

$$\arg\max_{S_w \subseteq C_w} CS(S_w).R \tag{5}$$

with

$$\begin{cases} CS(S_w).C \leq C_G \\ CS(S_w).T \leq T_G \end{cases} \tag{6}$$

Based on the set of selected workers $\widehat{W_i}$ for each crowd service CR_i, the synthesized cost and time constraints for CR_i could be calculated by Equation 2 and Equation 3 respectively.

5.2. Genetic Algorithm

The complexity of the above-defined optimization problem is NP-hard (in the number of workers). Thus CROWDSERVICE makes use of Genetic Algorithm (GA) [Forrest et al. 1993] to select workers.

First, we encode the worker selections of crowd services as chromosomes. We adopt the array-based *chromosomes*. The chromosome is partitioned to several parts. Each part is assigned to a crowd service. The length of the chromosome is the total number of candidate workers. Each gene in the chromosome represents the boolean selection of a worker, whose value is 0 or 1. The corresponding worker will be selected if and only if the value of a gene is 1.

Second, we define the fitness function F, which returns a fitness value representing the worthiness of a candidate solution. Given a composite service $CS(S_w)$, the fitness function $F(CS(S_w))$ is computed using the Equation 7.

$$F(CS(S_w)) = \begin{cases} 0.5 * CS(S_w).R + 0.5 & \text{if Constraint 6 satisfies} \\ 0.5 * CS(S_w).R - \dfrac{1}{3} * (Q_c + Q_t + Q_n) & \text{otherwise} \end{cases} \tag{7}$$

where

$$Q_c = \begin{cases} 0 & \text{if } CS(S_w).C \leq C_G \\ \dfrac{CS(S_w).C - C_G}{CS(C_w).C - C_G} & \text{otherwise} \end{cases} \tag{8}$$

$$Q_t = \begin{cases} 0 & \text{if } CS(S_w).T \leq T_G \\ \dfrac{CS(S_w).T - T_G}{CS(C_w).T - T_G} & \text{otherwise} \end{cases} \tag{9}$$

$$Q_n = \begin{cases} 0 & \text{if } |\widehat{W_i}| \geq RN_i \text{ for all i} \\ \dfrac{\sum_{i=1}^{n} RN_i - \sum_{i=1}^{n} |\widehat{W_i}|}{\sum_{i=1}^{n} RN_i - \sum_{i=1}^{n} |W_i|} & \text{otherwise} \end{cases} \tag{10}$$

The fitness value for $CS(S_w)$ ranges from 0.5 to 1 if $CS(S_w)$ does not violate the cost and time constraints, otherwise the value ranges from -0.5 to 0.5. Such a threshold is to guarantee that a plan which does not violate the constraints always has a higher probability to be chosen for crossing over. Chromosomes that violate the constraints may still be chosen as they may contain the correct genes for mating. Q_c, Q_t, and Q_n are penalty values for cost and time constraints, and the number of selected works respectively, which range from 0 to 1 (see Equations 8, 9, and 10). Their values reflect how seriously C_G, T_G, and RN_i are violated.

Since the speed of convergence in GA depends largely on the quality of the seed solutions in the initial population, we use an enhanced initialization method to generate a good initial set of solutions. First, we sort the candidate workers of each crowd service based on their response time, cost, or reliability. Second, for each crowd service CR_i, we select the first RN_i candidate workers that conform to the global time and cost constraints, and aggregate their cost and response time. Subsequently, we divide the global time and cost constraints for each crowd service, according to the ratio of the aggregated response time and cost over other crowd services. Third, for each crowd service, we greedily select workers according to the sorted sequence, as long as they conform to the allocated time and cost constraints. We repeat the process for other crowd services, and this will result in an "enhanced" initial chromosome. Then we can have three "enhanced" chromosomes by sorting the candidate workers by time, cost, and reliability, respectively. We include all of them among the randomly generated chromosomes.

Lastly, to apply GA, one needs to define crossover and mutation operations to create new chromosomes for the next generation of a population. In our approach, we use the typical one-point crossover and mutation [Gen et al. 1997].

5.3. Reliability Calculation

Note that the computation of Equations 4 and 5 is exponential, since it requires the enumeration of subsets of selected workers that have the size larger than RN. In particular, the complexity of the computation is $O(2^m)$, where m is the number of selected workers. We propose an effective way to deal with the scalability of the calculation. Suppose there is a crowd service CR_i with *m* candidate workers, and let R_{CR_i} be the probability where at least RN_i workers run successfully. R_{CR_i} is exactly the complement of the probability where at most $RN_i - 1$ workers run successfully. This is illustrated using Equation 11, where s represents the number of workers who run successfully.

The probability of having at most $RN_i - 1$ workers run successfully, is equal to the probability of having at least $m - RN_i + 1$ workers fail, as shown in Equation 12, where f represents the number of workers that fail.

$$R_{CR_i} = 1 - P_{s \leq (RN_i - 1)} \tag{11}$$

$$P_{s\leq(RN_i-1)} = P_{f\geq(m-RN_i+1)} \tag{12}$$

Our goal is to improve the efficiency and the scalability of calculation of R_{CR_i}. To achieve this goal, rather than calculating R_{CR_i} precisely, we calculate the lower bound of R_{CR_i}, which could be derived from the upper bound of $P_{f\geq(m-RN_i+1)}$ using Equation 12. We use $f_{ij} = 1 - r_{ij}$ to denote the failure probability of the jth worker, where r_{ij} is the reliability of the worker. Assume f_{i1}, f_{i2}, ... , f_{im} are failure probabilities of candidate workers where $f_{i1} \geq f_{i2} \geq ... \geq f_{im}$, an upper bound of exactly n workers fail is $C_m^n * \prod_{j=1}^{n} f_{ij} * \prod_{j=n+1}^{m} (1 - f_{ij})$. The final result is the summation of the probabilities of exactly $n \in \{m - RN_i + 1, m - RN_i + 2, ...m\}$ workers fail. The complexity of the estimation is $O(m^2)$.

The estimation approach works well when the number of selected workers is not too small (e.g., $m > 10$). When m is sufficiently large, each individual combination of successful and failure workers would result in smaller and similar probability. This makes the lower bound estimation tight. In our work, we make use of the lower bound estimation when $m > 15$. When $m \leq 15$, we enumerate all possibilities to get the precise reliability of workers.

6. IMPLEMENTATION

We have implemented the CROWDSERVICE framework on the Android platform using JADE, which is an open source agent development framework [2]. The implementation includes a crowd service platform, a consumer client, and a worker client. Task agent and checkin agent on the platform and worker agent on worker client are all implemented as JADE agents. These agents communicate via ACL (Agent Communication Language) messages.

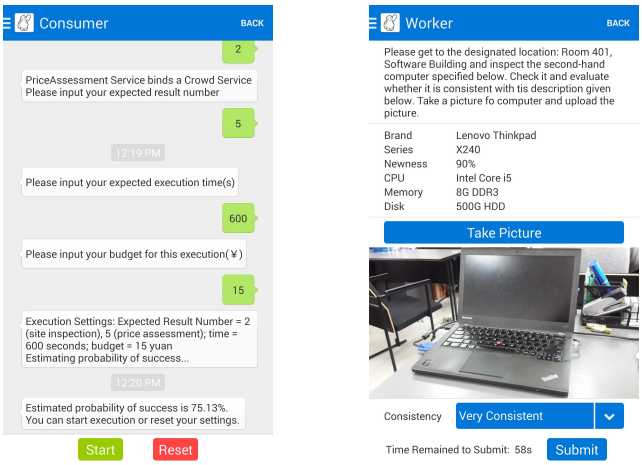
The consumer client is implemented on the Android platform using the OSGi [3] framework. Specifically, we choose Apache Felix [4], an open source implementation of OSGi. Both composite services and component services (local services or stubs to remote services) are implemented as OSGi bundles. When a consumer chooses a composite service to execute, the execution engine requests corresponding composite service bundle and required component service bundles from the service repository, and dynamically installs and activates them. After that, the engine invokes the execution method of the composite service defined in its base class.

Consumer UI includes a composite service execution list view and an execution process view. To execute a composite service, the consumer can click the plus (+) button on the execution list view and choose a composite service on a pop-up dialog. The execution process view (see Figure 4(a)) provides user interaction for the consumer to specify execution settings and gets feedback of the estimated probability of success. After the execution of a composite service starts, the execution process view shows execution logs for the consumer to understand its execution process.

Worker client is also implemented on the Android platform, including worker agent and worker UI. Worker UI includes a task examination view and a microtask view. When worker agent receives an open call, it initiates a task examination view showing received task instructions. If the worker chooses to participate, a pop-up dialog is shown for him/her to input an offer. When worker agent receives a participation confirmation, it initiates a microtask view showing received task instructions and an interaction form.

The task examination view and microtask view are dynamically generated based on received task instructions and task field descriptions. A series of Android UI controls are used to show different kinds of information and interaction fields, for example EditText (for primitive data type field), Spinner (for choice field), ImageView (for image display). Figure 4(b) shows a microtask view of the site inspection service, which asks the worker to confirm the description and take a picture of a secondhand laptop.

[2]JADE: http://jade.tilab.com
[3]OSGi: http://www.osgi.org
[4]Apache Felix: http://felix.apache.org



(a) Consumer UI (b) Worker UI

Fig. 4. CROWDSERVICE Android Client UI

7. EVALUATION

To evaluate the proposed framework and algorithm, we conducted an experimental study and a user study to answer the following three research questions:

— **RQ1 (Effectiveness)**: Is the genetic algorithm effective in improving the reliability of composite services?
— **RQ2 (Scalability)**: How efficient is the genetic algorithm? Can it scale well with the result number constraints and the number of candidate workers?
— **RQ3 (Usability)**: Can crowd services be provided reliably in real life when sufficient workers are available? How do the users think of crowd services?

7.1. Q1: Effectiveness

To evaluate the effectiveness of the genetic algorithm, we conducted an experimental study, which compared the algorithm with naive algorithms in terms of success rate of composite services.

Given a composite service, a naïve algorithm selects a set of workers for each crowd service based on one of the workers' properties (e.g., cost, time, reliability) in the following way: 1) sort the candidate workers of each crowd service based on a single property; 2) for each crowd service CR_i, select the first RN_i candidate workers that conform to the global time and cost constraints, and aggregate their cost and response time; 3) divide the global time and cost constraints for each crowd service according to the ratio of the aggregated response time and cost over other crowd services; 4) for each crowd service, greedily select workers according to the sorted sequence, as long as they conform to the allocated time and cost constraints.

Considering different worker properties, we can have the following four naïve algorithms: *Naive-C* algorithm, which orders workers by cost (ascending order); *Naive-R* algorithm, which orders workers by reliability (descending order); *Naive-T* algorithm, which orders workers by time (ascending order); *Naive-U* algorithm, which orders workers by utility (descending order). The utility of a worker is the normalized sum of cost, time and reliability (equally weighted).

The experiment was based on a composite service with a sequential composition of three crowd services. In the simulation, we created a pool of 2,000 candidate workers with independently generated cost, distance (time), and reliability, following normal distribution for cost and reliability, uniform distribution for distance (time). We then randomly selected 10 sets of candidate workers for the experiment and each set had 150 candidate workers (40, 50, 60 for the three crowd service respectively).

We compared the genetic algorithm with the four naïve algorithms under different combinations of cost and time constraints (i.e., C_G and T_G). We tested all the algorithms under each combination

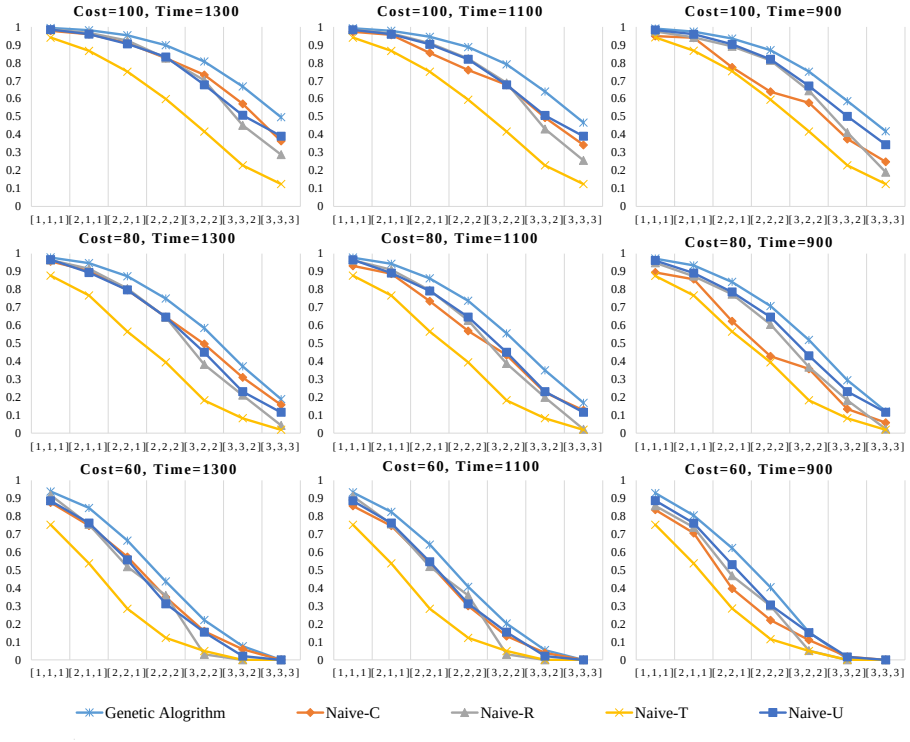


Fig. 5. Comparison of the Effectiveness of Different Algorithms

of high cost (100), medium cost (80), low cost (60) and long time (1300), medium time (1100), short time (900). Given a cost constraint and a time constraint, we evaluated the success rates of the algorithms with different result number constraints of the three crowd services from [1, 1, 1] (i.e., RN_1=1, RN_2=1, RN_3=1) to [3, 3, 3] (i.e., RN_1=3, RN_2=3, RN_3=3). For each result number constraint, we ran the five algorithms with the 10 sets of candidate workers and computed the average success rate of each algorithm. Due to the heuristic nature of genetic algorithms, we ran the genetic algorithm 50 times for each combination of inputs (cost, time, result number constraints, and candidate workers) and reported the average success rate. We set the genetic algorithm to terminate at 1,000 generations or when the fitness does not improve for 30 generations, whichever happens earlier.

The success rate of a run of the composite service was computed based on the reliability of the selected workers in the following way. For each selected worker, we treat its reliability as the probability (p) of successfully returning the result for the crowd service. We then randomly generate a value for the worker, which takes the value 1 (resp. 0) with a probability p (resp. $1-p$). A run is successful if each crowd service CR_i has at least RN_i workers whose values are 1. For each run, we repeated the above simulation 1,000 times and computed its success rate as $S/1000$, where S is the number of successes.

Figure 5 shows the success rates of the five algorithms under different constraints. The X axis is the result number constraints (i.e., RN_1, RN_2, RN_3) and the Y axis is the success rate. It can be seen that the success rate decreases when the result number constraints increase. Among the four naïve algorithms, *Naive-U* has the best results most of the times. The genetic algorithm achieves the highest success rate under all the settings. It is on average 6.2% more successful than the best naive algorithm. The results show that the genetic algorithm can significantly improve the reliability of composite services.

7.2. Q2: Scalability

We evaluated the efficiency and scalability of the genetic algorithm by analyzing its execution time with different numbers of candidate workers and result number constraints. The evaluation was conducted based on the same composite service and candidate worker pool used in Section 7.1. The main source of overhead of the approach comes from the selection of crowd workers based on their attributes using the genetic algorithm (GA). We set the genetic algorithm to terminate when the fitness does not improve for 30 generations. Given a set of candidate workers and a result number constraint, we ran the genetic algorithm 100 times and computed the average execution time in seconds. The experiments were conducted in Windows 7 on a machine with a Intel(R) Core(TM) i5-4200U, running with a dual core 1.60GHz CPU, 3M cache and 8 GB RAM.

The results are shown in Figure 6. The numbers of different curves (i.e., 50, 100, 150, 200, 250, 300) mean the numbers of candidate workers for each crowd service. The numbers on the X axis mean the result number constraints (i.e., RN_1, RN_2, RN_3). The overall tendency is that the execution time increases when the result number constraints or the number of candidate workers increases. But there are some exceptions, for example, the execution time does not always increase with the result number constraints when the number of candidate workers of each crowd service is 50. This could be due to the randomness of the genetic algorithm. In general, the execution time increases significantly with the result number constraints, but only increases linearly with the number of candidate workers when the result number constraints are fixed.

The results show that the genetic algorithm can conduct constraints synthesis and worker selection with hundreds of candidate workers in a matter of seconds. It can complete a run in less than 3 seconds when the result number constraint of each crowd service is no greater than 8. We believe the algorithm is sufficiently efficient and scalable as often in practice one would work with fewer workers and smaller result number constraints.

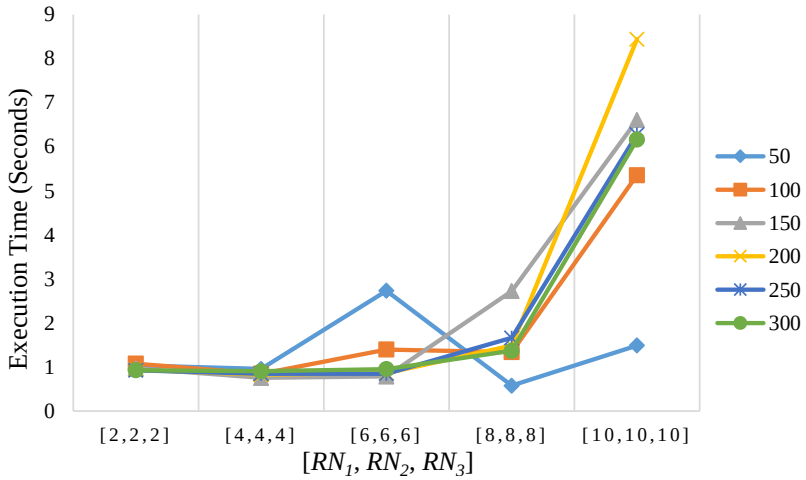


Fig. 6. Scalability Evaluation of the Genetic Algorithm

7.3. Q3: Usability

We conducted a user study to evaluate the usability of our framework and implementation. Our user study was performed by executing the composite service of buying secondhand items (see Figure 1) in a real-life scenario. We recruited 23 students (including 4 PhD, 17 master, and 2 senior undergraduate students) from the school of computer science of Fudan University to play different roles. Among them, three students played the role of secondhand laptop seller and were located in

Table II. Execution Processes and Results of User Study

Con.	Constraint		Crowd Service 1: Site Inspection									Crowd Service 2: Price Assessment								
	T	C	RN	ST	SC	Pro.	Succ.	ET(s)	Worker			RN	ST	SC	Pro.	Succ.	ET(s)	Worker		
									#OF	#SE	#WR							#OF	#SE	#WR
C1	600	25	3	388	17	0.99	Yes	145	13	6	6	3	100	8	0.99	Yes	47	17	7	7
	650	30	5	176	17	0.59	Yes	89	9	5	5	6	100	11	0.96	Yes	52	16	8	8
	400	20	1	223	13	0.99	Yes	98	9	4	4	3	100	6	0.99	Yes	100	18	6	5
	800	40	8	184	23	0.43	Yes	81	9	8	8	10	-	-	-	No	-	-	-	-
C2	750	35	3	310	17	0.99	Yes	310	9	8	7	3	100	21	0.99	Yes	89	16	12	12
	800	30	2	176	20	0.99	Yes	82	11	6	6	2	100	10	0.99	Yes	50	19	8	8
	200	10	2	120	5	0	No	-	9	-	-	-	-	-	-	-	-	-	-	-
	600	30	1	229	18	0.99	Yes	229	7	5	3	1	100	16	0.99	Yes	90	16	12	12
C3	600	30	3	415	17	0.99	Yes	162	10	5	5	7	-	-	-	No	-	-	-	-
	500	25	1	367	10	0.99	Yes	143	7	4	4	5	100	14	0.99	Yes	63	17	10	10
	600	35	3	388	24	0.99	Yes	136	9	7	7	3	100	10	0.99	Yes	53	17	7	7
	1000	25	2	311	16	0.99	Yes	186	9	6	6	4	100	8	0.99	Yes	69	17	7	7

three different buildings in the campus; 20 students played the role of worker and three of them also played the role of consumer. For the price assessment service, all the workers were set to be experts who can be selected. The reliability of each worker was equally set to 0.9.

We deployed a crowd service platform on a server in our campus network and published the two crowd services (site inspection, price assessment) on the platform. All the consumers installed consumer clients with the composite service of buying secondhand items on their mobile phones. All the workers installed clients on their mobile phones. In each execution, the consumer specified an overall time, cost constraint and a number of returned personal results for each involved crowd service.

Each consumer executed the composite service four times and was given 40 yuan (RMB) (about 7 USD) for each execution. He/she could decide the overall cost and time constraints and the result number constraint of each crowd service by himself/herself, but was told to try different settings in the executions. All the workers walked around the three buildings (within 10-500 meters) and decided whether to participate in a crowdsourcing task and the expected reward by themselves. They could get the expected rewards as bonus if they were selected and successfully finished their work. Therefore, they were motivated to provide reliable service with reasonable cost as in the real-world scenario.

After the experiments, we conducted a questionnaire survey and a group interview to get the feedback of the users.

7.3.1. Execution Processes and Results. Table II shows statistics of the execution processes and results of our user study. For each execution, it lists the consumer (Con.), consumer specified time (T) and cost constraints (C), and execution information of the two crowd services. For each crowd service, it lists the result number constraint (RN), synthesized time constraint (ST) and cost constraint (SC), estimated probability of success (Pro.), whether succeeded or not (Succ.), execution time (ET), and the numbers of worker offers (#OF), selected workers (#SE), and workers who successfully returned their results (#WR).

For Crowd Service 1 (site inspection), on average, each service execution had nine worker offers, six selected workers, and six returned results, and was finished in 151 seconds. For Crowd Service 2 (price assessment), on average, each service execution had 17 worker offers, nine selected workers, and eight returned results, and was finished in 68 seconds. The numbers of worker offers of Service 2 were usually larger and the expected rewards were usually lower than those of Service 1. This is reasonable, since Service 2 is location independent and all the workers were qualified.

Among all the 12 executions of Service 1, 11 executions succeeded (i.e., the required number of worker results were returned to the platform) and only one failed, making a success rate of 92%. Among all the 12 executions of Service 2, 9 executions succeeded and the other three failed, making a success rate of 75%. After analyzing the execution results and logs, we found that there are two main causes for failed executions. One cause is that constraints synthesis was unable to

find a feasible resource allocation plan, which was usually due to high result number constraint, insufficient workers available, low cost, or insufficient time. The other cause is that worker selection was unable to select a set of workers, which was usually due to high result number constraint, insufficient worker offers, or low cost.

The above analysis shows that crowd services can be provided as reliable services in real life when sufficient workers are available.

7.3.2. User Feedback. We analyzed the feedback of the consumers and workers collected in the survey and interview.

The consumers generally think crowd service provides a convenient and cost-efficient way to fulfill their needs by leveraging crowd intelligence and labor. Their main concerns include reliability, result quality, and privacy. They regard the estimation of probability of success is useful for them to establish confidence on crowd services and provides useful feedback for their execution settings. They would usually have enough confidence to start the execution if the estimated probability of success was higher than 75%. In 30% cases, they adjusted their execution settings (i.e., cost, time, and result number constraints) to improve the probability of success. Some consumers mentioned that they were not sure about the quality of the returned results, as they did not know whether the workers had done their work properly. Some consumers were concerned about privacy issues, as some tasks may involve private information such as home address. They suggested that CROWDSERVICE could allow consumers to set the scope of worker selection to specific groups of people such as those in the same university.

Some workers think crowd service provides an opportunity for them to create value by making use of their location conveniences and capabilities. They suggested to provide more transparency. For instance, they often felt overwhelmed when considering expected rewards as they did not know whether their expectations were too high or too low. This problem can be addressed by providing a reference price based on historical data of similar tasks or estimation formula. The workers suggested two improvements on the interaction modes supported by CROWDSERVICE. One suggestion is to provide a negotiation process between consumers and workers. This is useful for some crowd services with high cost and high quality requirements, but may introduce unnecessary disturbance for other crowd services. The other suggestion is to provide a pull mode for task participation instead of pushing tasks to them. The current task participation is push mode, i.e., the platform pushes task invitations to workers, while by pull mode workers can search for interested crowdsourcing tasks when they are available.

7.4. Threats to Validity

A major threat to the validity of the experimental study lies in the settings of worker parameters. The assumed normal distribution of the reliability, distance, and cost of workers may not be realistic, especially when the number of candidate workers is not so large (e.g., less than 10). The advantage of our constraints synthesis and worker selection algorithm may vary in different settings, but we believe it can improve the reliability of composite services in most cases.

Threats to the validity of the user study mainly lie in the setting of students as workers, the incentive mechanism, and the activity region of workers. Although students can also participate in crowd services in campus, they are not representatives of real typical workers in crowdsourcing. Our incentive mechanism allows the selected workers to get the expected rewards as real bonus, which can motivate the students to participate in crowdsourcing tasks. However, the students' motivation of participation was also influenced by the fact that they were invited to participate in a controlled user study. The workers' decisions of participation and offers in real life are influenced by more complex factors related to job stability, experience and reputation. The study was conducted in campus and the workers' distance to target locations was usually small (hundreds of meters). The routes of workers to target locations in real life are usually much more complex, for example, the traffic conditions and the time of getting to a target location are more uncertain. Large-scale evaluation in real life requires to deploy and operate CROWDSERVICE on mobile social network platforms.

However, we think the current execution results and feedback collected from students are sufficient as preliminary evidence for the practicability and usability of CROWDSERVICE.

8. DISCUSSION

As an emerging software paradigm for Internet computing, Internetware will orchestrate information, processes, decisions, and interactions in this Internet environment [Mei et al. 2012]. An essential characteristic of Internetware is the seamless integration of computing devices, human society, and physical objects. CROWDSERVICE facilitates the integration of human society with computing devices by supplying crowd intelligence and labor as reliable and publicly accessible services. This is achieved through its customizable framework and worker selection algorithm. The customizable framework allows the crowd to participate in different kinds of tasks while ensures that the submitted results can be aggregated into a standardized output. The worker selection algorithm ensures the reliability of crowd services by optimally allocating overall time and cost constraints to each involved service and selecting a set of responded workers for each involved crowd service based on their reliability, expected reward, and distance.

The worker selection algorithm of CROWDSERVICE is based on the incentive model of worker bidding, i.e., candidate workers bid for crowdsourcing tasks with expected reward. It may need to be adapted if a different incentive model is adopted. For example, if the platform provides a price when sending an open call, the worker selection algorithm can be adapted to select workers based only on their distance and reliability.

The large-scale application of crowd services in real life requires a systematic operating platform and ecosystem built on mobile social network. Reliability is the key for the wide acceptance of crowd services. The willingness of candidate workers and quality of worker results are the two major challenges.

Participation willingness of candidate workers is largely influenced by incentives and interaction design. Incentives depend on consumers' pricing on crowd services, which can be reasonably adjusted by a market mechanism. Interaction design may hinder candidate workers from participating if it bothers them. To avoid budget overrun, currently CROWDSERVICE requires a worker to first send an offer and then wait for the confirmation of being selected. This may cause a worker to give up or shift his/her focus from the crowdsourcing task. Therefore, for a crowdsourcing task that requires little effort (e.g., reporting empty seats in a building) the platform can allow a worker to directly submit his/her result with an offer. This means that workers who have submitted their results would not get expected rewards if they were not selected, but their efforts could still be recognized by their reputation.

A main challenge to the quality of worker results is worker cheating, which may reduce consumers' trust on crowd services. For example, a worker participating in a restaurant booking task may cheat by submitting a fake table number and booking confirmation. It is reported that workers recruited for financial rewards have more incentive to cheat the system [Quinn and Bederson 2011]. To assure the quality of crowd services, it is useful to provide additional verification of worker results. Some simple verification could be done automatically, for example by checking the definition of a picture. For a location-based task, the platform can track the location of crowd workers when they submit answers, and if the tracked location of a worker does not coincide with the target location, his/her answer will be recognized as a fake one [Chen et al. 2014]. More sophisticated verification may need to be done by crowdsourcing. For example, the verification of a restaurant booking can be done by launching a crowdsourcing task for confirming the booking.

Apart from optimal resource allocation and worker selection, it is possible to further improve the reliability of a crowd service by self-adaptive service provisioning. On the one hand, workers could be adaptively selected in a multi-round mode. For example, if there are not sufficient candidate workers within a short distance who respond to an open call, the platform could send another open call to available workers within a longer distance. On the other hand, the functionality of a crowd service often could be provided in other alternative ways, including computational services (e.g., automatic price assessment of secondhand laptops) and commercial services (e.g., site inspec-

tion provided by a professional company). These alternative services may be less accurate or more expensive, but could be used as backup of a crowd service.

9. RELATED WORK

In software engineering, crowdsourcing has been used for various software development tasks such as programming and testing [Peng et al. 2014]. LaToza et al. [LaToza et al. 2014] developed an approach to decomposing programming work into microtasks. Stol and Fitzgerald [Stol and Fitzgerald 2014] presented an industry case study of crowdsourcing software development at a multinational corporation. Mao et al. [Mao et al. 2013] empirically analyzed historical data from TopCoder crowdsourced software development tasks to identify potential price drivers. Saremi and Yang [Saremi and Yang 2015] proposed a systems dynamic model for crowdsourced software development platforms based on data gathered from Topcoder. They [Yang and Saremi 2015] also reported an empirical study to develop further understanding about the relationship between tasks award and associated worker behaviors.

In the database community, crowdsourcing has been used to develop crowdsourced query processing systems. Franklin et al. [Franklin et al. 2011] developed CrowdDB, a relational query processing system that uses microtask-based crowdsourcing to answer queries that cannot otherwise be answered. Others investigated a variety of crowdsourced queries such as `SELECT` query [Trushkowsky et al. 2013], `MAX` query [Guo et al. 2012; Venetis et al. 2012], and `JOIN` query [Demartini et al. 2012; Wang et al. 2013]. This kind of crowdsourced queries are location independent, thus can be processed based on Web-based crowdsourcing platforms such as AMT.

Jarrett and Blake [Jarrett and Blake 2015] proposed a descriptive architecture for a general crowdsourcing infrastructure. Their work does not consider location-based crowdsourcing, nor does it support the provisioning of crowdsourcing as services or optimal selection of workers.

There have been some researches on worker selection and quality control in crowdsourcing. Bozzon et al. [Bozzon et al. 2013] proposed a framework that supports reactive control of human computations in crowdsourcing. Li et al. [Li et al. 2014] proposed a crowd targeting framework for automatically discovering the specific group of high-quality workers. Ipeirotis and Gabrilovich [Ipeirotis and Gabrilovich 2014] proposed to use existing Internet advertising platforms for targeting users with the suitable expertise. Bernstein et al. [Bernstein et al. 2011] proposed to use synchronous crowds to create real-time crowd-powered interfaces with dramatically lower crowd latency. Ye et al. [Ye et al. 2015] proposed a trust evaluation model of crowdsourcing workers and an evolutionary algorithm for trustworthy worker selection. Yu et al. [Yu et al. 2015] proposed a reputation aware task sub-delegation approach to help workers decide when and to whom to sub-delegate tasks by considering a workers reputation, workload, the price of its effort and its trust relationships with others. Qiu et al. [Qiu et al. 2016] proposed an approach for the task assignment problem in crowdsourcing platforms, which offers a theoretically proven algorithm to assign workers to tasks in a cost efficient manner while ensuring high accuracy of the overall task. These researches focus on crowdsourcing tasks for collecting data and acquiring information and do not consider the locations of workers and tasks that are essential in mobile crowdsourcing. Moreover, these researches do not consider the optimal allocation of the overall cost and time constraints of a composite service over its component services.

Current researches on mobile crowdsourcing focus on location-based information services with crowd sensing. Ra et al. [Ra et al. 2012] proposed a crowd sensing programming framework which can coordinate crowd sensing tasks between smartphones and a cluster on the cloud. Waluyo et al. [Waluyo et al. 2013] presented a participatory mobile sensing system for estimating individual exposure to environmental pollution. Zhou et al. [Zhou et al. 2012] presented a bus arrival time prediction system based on bus passengers' participatory sensing. Chen et al. [Chen et al. 2014] developed a general spatial crowdsourcing platform, which implements a collection of related techniques such as geographic sensing, worker detection, and task recommendation. These researches do not consider the optimal selection of workers based on the cost and time constraints or the allocation of the overall constraints over a set of component services.

Our work is also related to the work on finding services with optimal Quality of Service (QoS) for a composite service. Some researchers [Alrifai and Risse 2009; Alrifai et al. 2010; Tan et al. 2016] proposed service selection approaches that make use of mixed integer programming (MIP) techniques, with other techniques such as skyline pruning, to find the optimal selection of services. Canfora et al. [Canfora et al. 2005] proposed the usage of genetic algorithm for finding services with optimal QoS. The selection of crowd workers has different characteristics from the selection of computational services. First, the crowd workers might change their distance to the target venue, which could make their response times change over time. We handle these by dividing the selection into two phases - constraints synthesis and worker selection phase. Second, human workers are prone to human errors; therefore given a crowd service, we allow users to specify the minimal number of workers for redundancy. Tan et al. [Tan et al. 2013] proposed an approach to synthesize response time constraints of component services that guarantee the global response time requirement. Our work is on selecting crowd workers that could collectively maximize the chance of success of a composite service.

10. CONCLUSION

In this paper, we have proposed the CROWDSERVICE framework to supply crowd intelligence and labor as publicly accessible crowd services. To support their composition with computational services, the framework wraps each crowd service as a Web service. The execution of these wrapped crowd services launches mobile crowdsourcing tasks, and aggregates the results returned by individual workers. For reliability, CROWDSERVICE dynamically synthesizes and updates near-optimal cost and time constraints for each crowd service involved in a composite service and selects a near-optimal set of potential workers for each to-be-executed crowd service. We have implemented CROWDSERVICE on the Android platform and evaluated its effectiveness, scalability and usability. In future work, we plan to improve CROWDSERVICE by providing more flexible user interaction, result verification, and adaptive crowd service provisioning. On the other hand, we will try to establish a mobile crowdsourcing platform in the campus, which can cover a range of scenarios in the students' everyday lives and support the evaluation of crowd services in real life.

REFERENCES

Mohammad Alrifai and Thomas Risse. 2009. Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. 881–890.

Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. 2010. Selecting skyline services for QoS-based web service composition. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. 11–20.

Mahmoud Barhamgi, Arosha K. Bandara, Yijun Yu, Khalid Belhajjame, and Bashar Nuseibeh. 2016. Protecting Privacy in the Cloud: Current Practices, Future Directions. *IEEE Computer* 49, 2 (2016), 68–72.

Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. 2011. Crowds in Two Seconds: enabling Real-time Crowd-powered Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. 33–42.

Alessandro Bozzon, Marco Brambilla, Stefano Ceri, and Andrea Mauri. 2013. Reactive Crowdsourcing. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. 153–164.

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. 2005. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO '05)*. 1069–1075.

Georgios Chatzimilioudis, Andreas Konstantinidis, Christos Laoudias, and Demetrios Zeinalipour-Yazti. 2012. Crowdsourcing with Smartphones. *IEEE Internet Computing* 16, 5 (2012), 36–44.

Manman Chen, Tian Huat Tan, Jun Sun, Yang Liu, Jun Pang, and Xiaohong Li. 2013. Verification of Functional and Non-functional Requirements of Web Service Composition. In *Proceedings of the 15th International Conference on Formal Engineering Methods (ICFEM '13)*. 313–328.

Zhao Chen, Rui Fu, Ziyuan Zhao, Zheng Liu, Leihao Xia, Lei Chen, Peng Cheng, Caleb Chen Cao, Yongxin Tong, and Chen Jason Zhang. 2014. gMission: A General Spatial Crowdsourcing Platform. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1629–1632. DOI: http://dx.doi.org/10.14778/2733004.2733047

Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: leveraging Probabilistic Reasoning and Crowdsourcing Techniques for Large-scale Entity Linking. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. 469–478.

Stephanie Forrest and others. 1993. Genetic algorithms – Principles of natural selection applied to computation. *Science* 261, 5123 (1993), 872–878.

Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. 61–72.

Mitsuo Gen, Runwei Cheng, and Dingwei Wang. 1997. Genetic algorithms for solving shortest path problems. In *Evolutionary Computation*. IEEE, 401–406.

Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. 2012. So Who Won?: dynamic Max Discovery with the Crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. 385–396.

Panagiotis G. Ipeirotis and Evgeniy Gabrilovich. 2014. Quizz: targeted Crowdsourcing with a Billion (Potential) Users. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. 143–154.

Julian Jarrett and M. Brian Blake. 2015. Collaborative Infrastructure for On-Demand Crowdsourced Tasks. In *Proceedings of the 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '15)*. IEEE Computer Society, Washington, DC, USA, 9–14. DOI: http://dx.doi.org/10.1109/WETICE.2015.31

John Krumm. 2009. A Survey of Computational Location Privacy. *Personal and Ubiquitous Computing* 13, 6 (2009), 391–399.

Thomas D. LaToza, W. Ben Towne, Christian M. Adriano, and André van der Hoek. 2014. Microtask Programming: building Software with a Crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. 43–54.

Hongwei Li, Bo Zhao, and Ariel Fuxman. 2014. The Wisdom of Minority: discovering and Targeting the Right Group of Workers for Crowdsourcing. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. 165–176.

Xuanzhe Liu, Yi Hui, Wei Sun, and Haiqi Liang. 2007. Towards Service Composition Based on Mashup. In *Proceedings of the 2007 IEEE International Conference on Services Computing - Workshops (SCW '07)*. 332 –339.

Ke Mao, Ye Yang, Mingshu Li, and Mark Harman. 2013. Pricing Crowdsourcing-based Software Development Tasks. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. 1205–1208.

E. Michael Maximilien, Ajith Ranabahu, and Karthik Gomadam. 2008. An Online Platform for Web APIs and Service Mashups. *IEEE Internet Computing* 12, 5 (2008), 32–43.

Hong Mei, Gang Huang, and Tao Xie. 2012. Internetware: A Software Paradigm for Internet Computing. *IEEE Computer* 45, 6 (2012), 26–31.

Xin Peng, Muhammad Ali Babar, and Christof Ebert. 2014. Collaborative Software Development Platforms for Crowdsourcing. *IEEE Software* 31, 2 (2014), 30–36.

Xin Peng, Jingxiao Gu, Tian Huat Tan, Jun Sun, Yijun Yu, Bashar Nuseibeh, and Wenyun Zhao. 2016. CrowdService: serving the individuals through mobile crowdsourcing and service composition. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. 214–219.

Chenxi Qiu, Anna C. Squicciarini, Barbara Carminati, James Caverlee, and Dev Rishi Khare. 2016. CrowdSelect: Increasing Accuracy of Crowdsourcing Tasks Through Behavior Prediction and User Selection. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, New York, NY, USA, 539–548. DOI: http://dx.doi.org/10.1145/2983323.2983830

Alexander J. Quinn and Benjamin B. Bederson. 2011. Human Computation: a Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. 1403–1412.

Moo-Ryong Ra, Bin Liu, Tom F. La Porta, and Ramesh Govindan. 2012. Medusa: a Programming Framework for Crowd-sensing Applications. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*. 337–350.

Florian Rosenberg, Francisco Curbera, Matthew J. Duftler, and Rania Khalaf. 2008. Composing RESTful Services and Collaborative Workflows: a Lightweight Approach. *IEEE Internet Computing* 12, 5 (2008), 24–31.

Razieh Saremi and Ye Yang. 2015. Dynamic Simulation of Software Workers and Task Completion. In *Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering (CSI-SE '15)*. IEEE Press, Piscataway, NJ, USA, 17–23.

Klaas-Jan Stol and Brian Fitzgerald. 2014. Two's Company, Three's a Crowd: a Case Study of Crowdsourcing Software Development. In *Proceedings of the 2014 International Conference on Software Engineering (ICSE '14)*. 187–198.

Tian Huat Tan, Étienne André, Jun Sun, Yang Liu, Jin Song Dong, and Manman Chen. 2013. Dynamic Synthesis of Local Time Requirement for Service Composition. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. 542–551.

Tian Huat Tan, Manman Chen, Jun Sun, Yang Liu, Etienne Andre, Yinxing Xue, and Jin Song Dong. 2016. Optimizing Selection of Competing Services with Probabilistic Hierarchical Refinement. In *Proceedings of the 36th IEEE/ACM International Conference on Software Engineering (ICSE '16)*.

Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. 2013. Crowdsourced enumeration queries. In *Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE '13)*. 673–684.

Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. 2012. Max Algorithms in Crowdsourcing Environments. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. 989–998.

Agustinus Borgy Waluyo, David Taniar, Bala Srinivasan, and Wenny Rahayu. 2013. Mobile Query Services in a Participatory Embedded Sensing Environment. *ACM Transactions on Embedded Computing Systems* 12, 2 (2013), 31:1–31:24.

Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2013. Leveraging Transitive Relations for Crowdsourced Joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. 229–240.

Mu Yang, Yijun Yu, Arosha K. Bandara, and Bashar Nuseibeh. 2014. Adaptive Sharing for Online Social Networks: A Trade-off Between Privacy Risk and Social Benefit. In *Proceedings of the 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '14)*. 45–52.

Ye Yang and Razieh Saremi. 2015. Award vs. Worker Behaviors in Competitive Crowdsourcing Tasks. *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* 00 (2015), 1–10.

Bin Ye, Yan Wang, and Ling Liu. 2015. Crowd Trust: A Context-Aware Trust Model for Worker Selection in Crowdsourcing Environments. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*. 121–128.

Han Yu, Chunyan Miao, Zhiqi Shen, Cyril Leung, Yiqiang Chen, and Qiang Yang. 2015. Efficient Task Sub-delegation for Crowdsourcing. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 1305–1311. http://dl.acm.org/citation.cfm?id=2887007.2887188

Pengfei Zhou, Yuanqing Zheng, and Mo Li. 2012. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*. 379–392.