CrowdService: Serving the Individuals through Mobile Crowdsourcing and Service Composition

Xin Peng[1,2], Jingxiao Gu[1,2], Tian Huat Tan[3], Jun Sun[3],
Yijun Yu[4], Bashar Nuseibeh[4,5], Wenyun Zhao[1,2]
[1] School of Computer Science, Fudan University, China
[2] Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
[3] Singapore University of Technology and Design, Singapore
[4] Department of Computing and Communications, The Open University, UK
[5] Lero - The Irish Software Research Centre, University of Limerick, Limerick, Ireland

ABSTRACT

Some user needs in real life can only be accomplished by leveraging the intelligence and labor of other people via crowdsourcing tasks. For example, one may want to confirm the validity of the description of a secondhand laptop by asking someone else to inspect the laptop on site. To integrate these crowdsourcing tasks into user applications, it is required that crowd intelligence and labor be provided as easily accessible services (e.g., Web services), which can be called crowd services. In this paper, we develop a framework named CROWDSERVICE which supplies crowd intelligence and labor as publicly accessible crowd services via mobile crowdsourcing. We implement the proposed framework on the Android platform and evaluate the usability of the framework with a user study.

CCS Concepts

•Information systems → Crowdsourcing;

Keywords

mobile crowdsourcing, service composition, reliability, Quality of Service (QoS)

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 [10, 11, 13]. 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 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 [7, 8, 12]. 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.

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, buy it by generating an order, and lastly 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 [5, 14]) 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.

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., $RN = 2$).

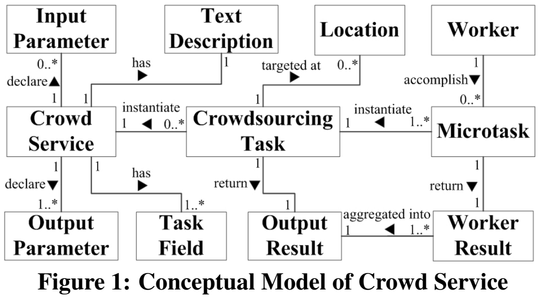


Figure 1: Conceptual Model of Crowd Service

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 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 with 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.

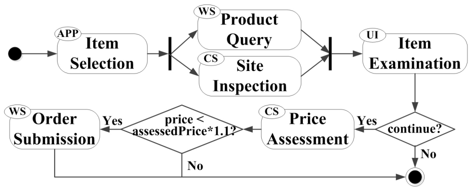


Figure 2: Business Process for Buying Secondhand Items

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.

Basic information of registered workers such as age, sex, and ability is stored in the worker profile database on the platform. 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. 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) [9, 6] 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. 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. 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

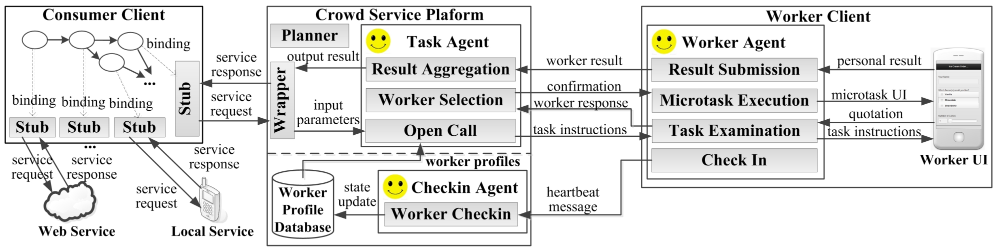


Figure 3: Overview of Crowd Service Framework

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 simple standard aggregation or a more complex 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.

Based on the task instructions, the worker accomplishes the assigned work and provides his/her result with the interaction form.

Table 1: A Virtual Table of Worker Results

Worker	Level	Time	AssPrice	Conf
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 1. 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 by simple aggregation or 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.

select avg(assPrice) 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(assPrice) from $WorkerResult$ T1 where conf='H' and not exists (select * from $WorkerResult$ T2 where conf='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.

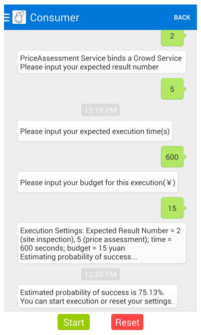
Complex Computation. Complex computation generates an output value by using service-specific algorithm. For example, an algorithm can be used to select a picture of the best quality from all the submitted results based on service-specific criteria. This kind of computation can be implemented as service-specific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

5. IMPLEMENTATION

We have implemented the CROWDSERVICE framework on the Android platform using JADE, which is an open source agent development framework [3]. The implementation includes a crowd service platform, a consumer client, and a worker client.

The consumer client is implemented on the Android platform using the OSGi [4] framework. Specifically, we choose Apache Felix [2], 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 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.



2

PriceAssessment Service binds a Crowd Service
Please input your expected result number

5

Please input your expected execution time(s)

600

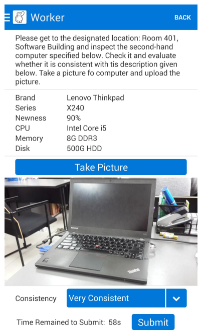
Please input your budget for this execution(¥)

15

Execution Settings: Expected Result Number = 2 (site inspection), 5 (price assessment); time = 600 seconds; budget = 15 yuan
Estimating probability of success...

Estimated probability of success is 75.13%.
You can start execution or reset your settings.

(a) Consumer UI



Please get to the designated location: Room 401, Software Building and inspect the second-hand computer specified below. Check it and evaluate whether it is consistent with tis description given below. Take a picture fo computer and upload the picture.

Brand	Lenovo Thinkpad
Series	X240
Newness	90%
CPU	Intel Core i5
Memory	8G DDR3
Disk	500G HDD

Consistency Very Consistent

Time Remained to Submit: 58s

(b) Worker UI

Figure 4: CROWDSERVICE Android Client UI

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.

6. USER STUDY

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 students, 17 master students, and 2 senior undergraduate students). Among them, three students played the role of secondhand laptop seller and were located in 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.

Each consumer executed the composite service four times and was given 40 yuan (RMB) (or roughly 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.

6.1 Execution Processes and Results

Table 2 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.

Table 2: 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

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

6.2 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. 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 developed an implementation of CROWDSERVICE on the Android platform and evaluated its usability.

Acknowledgments

This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203, and ERC Adv. Grant 291652 - ASAP.

8. REFERENCES

[1] Amazon Mechanical Turk. http://www.mturk.com. 2015.

[2] Apache Felix. http://felix.apache.org. 2015.

[3] JADE. http://jade.tilab.com. 2015.

[4] OSGi Alliance. http://www.osgi.org. 2015.

[5] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient QoS-aware service composition. In *WWW 2009*, pages 881–890.

[6] M. Barhamgi, A. K. Bandara, Y. Yu, K. Belhajjame, and B. Nuseibeh. Protecting privacy in the cloud: Current practices, future directions. *IEEE Computer*, 49(2):68–72, 2016.

[7] A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In *WWW 2013*, pages 153–164.

[8] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti. Crowdsourcing with smartphones. *IEEE Internet Computing*, 16(5):36–44, 2012.

[9] J. Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009.

[10] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards service composition based on mashup. In *Proceedings of the 2007 IEEE International Conference on Services Computing - Workshops*, SCW '07, pages 332 –339, 2007.

[11] E. M. Maximilien, A. Ranabahu, and K. Gomadam. An online platform for web APIs and service mashups. *IEEE Internet Computing*, 12(5):32–43, 2008.

[12] X. Peng, M. A. Babar, and C. Ebert. Collaborative software development platforms for crowdsourcing. *IEEE Software*, 31(2):30–36, 2014.

[13] F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf. Composing RESTful services and collaborative workflows: a lightweight approach. *IEEE Internet Computing*, 12(5):24–31, 2008.

[14] T. H. Tan, E. André, J. Sun, Y. Liu, J. S. Dong, and M. Chen. Dynamic synthesis of local time requirement for service composition. In *ICSE 2013*, pages 542–551.