On Providing Crowdsourcing as a Service

Thomas Springer, Tenshi Hara, Alexander Schill
Computer Networks Group
Technische Universität Dresden, Faculty of Computer Science
Dresden, Germany
{*firstname.lastname*}*@tu-dresden.de*

I. Introduction

During the last years, crowdsourcing has been increasingly used as means for solving complex tasks with the help of a flexible group of contributors. As a result, more and more crowdsourcing systems appear in the web. Compared to a large number of crowdsourcing solutions build from scratch, crowdsourcing platforms like Amazon Mechanical Turk (AMT) or Open Turk (OT), an extension of AMT, offer reusable services for crowdsourcing requesters and contributors. Usually hosted in the cloud they provide user management, allow requesters to create tasks from a predefined set of task templates, and upload these tasks to the platform. Contributors can search for appropriate tasks and submit their contributions to the platform using a web-based GUI. Submission processing is performed by the crowdsourcing platform in the cloud, requesters can just fetch results. In addition, rewarding of contributors is handled by the platform.

While supporting reuse of crowdsourcing functionality, these platforms suffer from serious limitations. Requesters can define their tasks only dependent on the predefined task templates of the platform which limits the type and complexity of crowdsourcing tasks. In addition, since processing of submissions is done at the platform, requesters do not have full control over submission processing, but have to rely on the algorithms provided by the platform. Moreover, these platforms mainly support *explicit* crowdsourcing where users actively contribute to the crowdsourcing process.

Many problem domains require more flexibility for task definition and submission processing than provided by these platforms. Especially, *implicit* forms of crowdsourcing are required where users solve a problem as a side effect of something else they are doing. A prominent example is environmental sensing where a smartphone app submits data sensed in the background. In this way users can contribute to a crowdsourcing process just by using the app. No other explicit activity is required by the user to participate in the crowdsourcing process.

To overcome these limitations, we propose to separate submission processing from basic crowdsourcing functionality for user and submission management. By introducing a crowdsourcing proxy, reusable crowdsourcing functionality can be provided as a service while submission processing is left to the requester to support a higher flexibility. Crowdsourcing clients can be implemented based on a generic library for communication with the crowdsourcing proxy to support arbitrary types of explicit and implicit crowdsourcing.

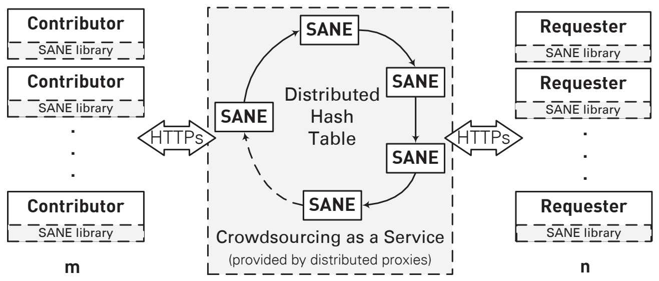

Figure 1. Architecture of the Crowdsourcing Platform

II. Approach

To allow the reuse of functionality for crowdsourcing, we propose a general crowdsourcing architecture providing crowdsourcing as a service. To separate submission processing from user and submission management we introduce a proxy component, called Server Access Network Entity (SANE), designed to be easily set up by anybody operating a web server with database and SSL components. Its purpose is to encapsulate generic crowdsourcing functionality and provide it as a service to contributors and requesters. In particular, the SANE handles the entire user, client device and submission management for multiple crowdsourcing projects to ensure anonymity, security, fault tolerance and performance. Different to approaches as followed by AMT or OT, submission processing is not part of the SANE functionality. Instead, submissions are forwarded to the requesters computing infrastructure, as shown in figure 1.

1) Crowdsourcing project setup: Since the SANE supports multiple crowdsourcing projects in parallel, as a first step requesters have to set up a crowdsourcing project by implementing and deploying the logic for submission processing. Either the requesters can use their own server infrastructure or rely on cloud resources. In addition, the submission capturing logic has to be implemented. This is usually done by implementing an entire application or integrate crowdsourcing into an existing application[1]. Thus, the requester can foresee any type of explicit or implicit crowdsourcing.

As a next step, the requesters have to register the submission types they expect to be forwarded by the SANE. A submission type is defined by an unique identifier, a description, a version number, a complex data structure defining the submission, the data returned in case of successful submission handling, as well as a set of error information data sets. As a final step, the contributor has to gather the crowdsourcing application from the requester.

During the deployment of the submission types requesters and SANE also exchange keys for asymmetric signing and encryption. Contributors also have to register with the SANE to be able to participate in crowdsourcing projects. The Key exchange is part of that registration procedure. After that contributors can start to submit data to the crowdsourcing project.

2) Communication flow for submission handling: As already mentioned, submissions are always indirectly forwarded via a SANE instance. Contributors create submissions according to the submission type definition provided by the requester. They are signed, encrypted and than sent to a SANE instance. Submission data includes also the contributors credentials.

The SANE instance decrypts the submission, verifies the signature and contributors credentials and stores the submission in a local database. It also verifies the submission data using the submission type definition[2].

Before forwarding, the SANE replaces the contributors credentials with an anonymous submitterID. In this way, requesters can assign submissions to anonymous contributors without the knowledge of their identity. After that, the modified submission is signed and encrypted by the SANE and forwarded to the requester. The requester decrypts the submission, verifies the signature and start submission processing which may include storing submission data.

According to the processing, an acknowledgement message with result data is sent to the SANE which is again signed and encrypted. Part of the result data could be an assessment of the submission quality which would allow the SANE to rate, reward or ban contributors. Finally, the SANE forwards the acknowledgement to the contributor.

3) Self-organization of SANE instances: To ensure scalability and fault tolerance, instead of a single proxy a set of SANE instances is used which cooperate to offer the crowdsourcing services.

[1]Based on the generic SANE access library applications for different device types can be implemented. Currently, Android apps, Web and Java applications are supported.

[2]The requester can also provide a key to the contributor so that the contributor can encrypt the submission data. However, in this way the SANE is unable to read or subsequently verify the submission.

Therefore, SANE instances organise themselves by utilising a distributed hash table (DHT) as shown in figure 1. Proven to be an efficient solution for the organisation of vast numbers of nodes, DHTs support simple setup of additional proxies as well as removal of proxies. The DHT contains the hashed clientID. According to the DHT approach each SANE instance maintains a hash area corresponding to the client's ID-hash. To ensure fault tolerance neighbouring SANE instances act as backups for each other.

A general design problem results from the fact that simple DHT distribution does not consider users' geographic positions. To overcome this issue, we propose to divide the SANE distribution into regional hash areas. Clients in different regions can than be forwarded to the local region by using DNS entries (For details please refer to [1]).

III. Evaluation

To evaluate our approach we implemented the SANE as a web server application in PHP (application logic) and MySQL (data storage), and deployed it on a standard LAMP web server. In our evaluation setup a cloud virtual private server with up to 3GHz CPU and 4GB RAM, running Linux 2.6.32-41-server x64, Apache httpd 2.0, MySQL 5.1.63, and PHP 5.2.12-nmm4 has been used. As a scenario for evaluation the MapBiquitous[3] project – an integrated indoor/outdoor location-based system – has been used. In our setup an Android client has been used for capturing explicit and implicit crowdsourcing submissions for updating Wi-Fi fingerprints and position corrections.

In the course of measurements carried out based on 500 to 1000 clients and 3 SANE components, our SANE implementation showed a strictly linear correlation between the amount of parallel client requests, successful replies and the packet losses up to a saturation point, which was reached at 5000 parallel requests for our setup. Thus, measurements proved that performance and scalability of our implementation follow the same characteristics as any website with database usage deployed on a comparable server.

IV. Conclusion

In this paper, a generic approach for providing crowdsourcing as a service has been described. Using a crowdsourcing proxy, we propose to separate submission capturing and processing from basic crowdsourcing functionality. In this way a more flexible, secure and reusable crowdsourcing service can be created.

References

[1] T. Hara, T. Springer, G. Bombach, and A. Schill, "Decentralised approach for a reusable crowdsourcing platform utilising standard web servers," in *PUCAA 2013: First International Workshop on Pervasive Urban Crowdsensing Architecture and Applications (UbiComp 2013 Workshops)*, 2013.

[3]http://goo.gl/eqJkG