Crowd-sourced Collection and Analysis of Software Packages

Version of August 21, 2023

Bowu Li

Crowd-sourced Collection and Analysis of Software Packages

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Bowu Li
born in Beijing, China

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

Cover picture: A “random” maze generated in postscript.

Crowd-sourced Collection and Analysis of Software Packages

Author: Bowu Li
Student id: 5463351

Abstract

The escalating complexity of software systems in the digital age heavily relies on reusable code collections(packages) for their development and operation. Despite the numerous advantages of pre-existing libraries, managing dependencies can be intricate and time-consuming. This thesis focuses on enhancing package management tools through a decentralized, crowd-sourced approach to distribute the preprocessing load more effectively across the software development ecosystem. We propose a novel platform comprising a back-end server and a Maven plugin, fostering an efficient and collaborative environment for developers to share computational results. This platform not only alleviates server load but also allows for the storage and reuse of frequently used artifacts, thereby avoiding redundant computations and reducing production costs for users. This crowd-sourcing model empowers developers to seamlessly request and contribute analysis results, saving time and resources while benefiting the broader community.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. S. Proksch, Faculty EEMCS, TU Delft
Committee Member:	Dr. S. Dumančić, Faculty EEMCS, TU Delft

Preface

This paper marks the end of my studies at the master's level. It has been an unforgettable journey and a valuable experience in my life. The two years have passed in the blink of an eye – I still vividly remember the first time I set foot at TU Delft, and now it's time to say goodbye. I am grateful to the people who have supported me along the way, as you have made this experience even more memorable.

First and foremost, I would like to express my gratitude to my supervisors, Arie van Deursen and Sebastian Proksch. When I was lost in finding a thesis topic, Arie guided me and provided me with the opportunity to explore such an exciting subject under Sebastian's guidance. With Sebastian's mentorship, I gradually delved into this project and he patiently addressed my various questions during our weekly meetings, offering suggestions to ensure I stayed on the right track. Under his guidance, I learned how to conduct academic research correctly, emphasizing the importance of framing research questions rather than getting lost in finding solutions. I appreciate his detailed guidance and assistance, which enabled me to successfully complete this project.

Secondly, I want to thank my parents. They shaped my character from a young age and continuously encouraged me to move forward in my life. It's also thanks to their financial support that I had the opportunity to receive an education here.

Finally, I want to thank my girlfriend, Chenfen. We shared the joy of success and supported each other during challenging times. Thank you for helping me maintain a positive outlook on life.

Bowu Li
Delft, the Netherlands
August 21, 2023

Contents

List of Figures

Chapter 1

Introduction

In the age of digitization, software systems have become the backbone of almost every industry, with the complexity of their structures escalating correspondingly [27]. These systems rely heavily on packages, which are collections of reusable code, for their development and operation. Managing dependencies, however, can be complex and time-consuming despite the various advantages of using pre-existing libraries.

In such cases, developers need to use a tool known as Package managers to assist them in performing installation, upgrade, and removal of packages on the target machines[2]. Package managers like Maven play a critical role in modern software systems, allowing developers to incorporate and manage a myriad of packages seamlessly, thus reaping numerous benefits [8]. Despite the seemingly low start-up costs provided by package managers, the process of reuse is not free due to the intricacies of dependencies, requiring developers to vigilantly track dependency updates, especially for multi-layer dependencies [4]. Developers often have a need to gain deeper insights into the dependencies they use in their projects, going beyond just importing them. This is where software analysis tools come into play. These tools provide developers with the ability to extract and analyze various aspects of their dependencies, allowing them to understand their performance, security vulnerabilities, and other relevant information.

The Fine-Grained Analysis of Software Ecosystems as Networks (FASTEN)[1] was proposed in 2018 to make package management tools smarter. By combining vulnerability information with call graphs, FASTEN introduced a fine-grained, method-level dependency tracking system. The Fine-Grained Analysis of Software Ecosystems as Networks (FASTEN) was proposed in 2018 to enhance the capabilities of package management tools. Unlike traditional tools which analyze each dependency locally, FASTEN introduces a novel approach where the server generates and stores the analytical results. It has a server that continuously downloads new artifacts from the Maven central repository and analyzes them. Whenever the developer needs to analyze the project's vulnerability, the hash value of the dependency package is sent to the server, which returns the required analysis result.

Despite its efficiency, this solution places a heavy burden on the server due to the high-performance CPU and large storage space requirements. In particular, it often leads to unnecessary computations for packages that are not frequently used by developers. If we want to analyze software that uses different package managers, we would need to cover

multiple ecosystems, further increasing the server's workload. As a result, due to the high resource load, there may be delays or increased operating costs. The scalability of using a central server to analyze all packages is relatively low.

The central motivation for this thesis is to distribute the pre-processing load more effectively across the software development ecosystem. To this end, we propose a decentralized solution based on the concept of crowd-sourcing. The term "crowd-sourcing", coined by Jeff Howe in 2006, describes a process where tasks are outsourced to a network of people, referred to as the 'crowd'[21]. This model encourages developers to share their computational results, which not only alleviates server load but also allows frequently used artifacts to be stored and reused, avoiding redundant computations [22]. In the crowd-sourcing paradigm, tasks are distributed to networked people to complete such that a company's production cost can be greatly reduced.[35]

The focus of this thesis is to design an efficient and collaborative platform that enables developers to leverage crowd-sourced analysis results for their software packages. This platform consists of a back-end server and a Maven plugin. The back-end server serves as the central repository, managing the collection and distribution of analysis tasks, as well as storing the contributed results. The Maven plugin integrates with developers' local development environments, enabling them to request and contribute analysis results seamlessly. By leveraging the collective efforts of the community, developers can save time and resources by accessing pre-computed results and contributing their analysis results to benefit others.

1.1 Research Questions

Based on the analysis of the above information and our requirements, we propose to address the following research questions in this paper to build a crowd-sourcing platform for collecting and selecting pre-computation tasks:

RQ1: How can the server aggregate and validate submissions of arbitrary preprocessing tasks?

In this thesis, the crowd-sourcing platform serves as the cornerstone. Therefore, the first step is to establish a basic framework consisting of a back-end server and a client-side Maven plugin to facilitate data exchange. Since the server does not generate the data itself but rather distributes the data generated by the clients to other clients, the primary challenge is to ensure the correctness of the data. To address this challenge, the thesis proposes the implementation of three different consensus mechanisms and two consensus integration methods. These mechanisms and methods aim to aggregate and validate the submissions of arbitrary preprocessing tasks, ensuring the accuracy and reliability of the collected results. By employing a combination of consensus mechanisms and integration methods, the platform can effectively handle diverse preprocessing tasks and facilitate collaborative data analysis.

RQ2: How many projects are needed to saturate the crowd-sourcing system?

As a crowd-sourcing system, the availability of content within the system increases with the number of user interactions. However, due to variations in dependencies used across different projects and the constant updates to artifact versions, projects that have already been analyzed may not necessarily provide assistance to future users, who may still need to compute certain tasks locally. We would like to determine the percentage of task answers that the next analyzed project can obtain when a sufficient number of projects have been analyzed.

RQ3: What is the overhead for a typical Maven run when the plugin is integrated into the Maven build?

One of the motivations behind introducing this crowd-sourcing system was to reduce the system overhead of software analysis. Therefore, it is crucial to evaluate the impact of integrating the plugin into the Maven build process. The thesis aims to assess the overhead experienced by users during a typical Maven run and determine the performance improvements achieved under different hit rates.

RQ4: What is the trade-off of computation load for the server?

One of the objectives of our thesis is to alleviate the computation load on the server and reduce the performance requirements placed on it. This is achieved by designating the server as the central hub for data collection rather than result generation. However, the comparison of different user-submitted results introduces additional overhead.

To evaluate the effectiveness of our system and determine a reasonable voting threshold, we compare the time required to reach a consensus from different submitted results with the time needed for the server to generate the results itself. This analysis allows us to assess the trade-off between the computation load on the server and the efficiency of reaching a consensus.

1.2 Contribution

This research has the following main contributions:

1. Specified the conceptual requirements for the crowd-sourcing Software Collection and Analysis platform.

2. Formalized the communication protocols that would be necessary for the crowd-sourcing Software Collection and Analysis platform.

3. Provision of multiple consensus strategies: Three consensus mechanisms and two consensus integration methods specifically designed for analyzing and evaluating the collected software package analysis results are introduced in the platform. Users can select one consensus mechanism and one consensus integration method to form a strategy. We conducted extensive evaluations across various strategies and provided recommendations based on different usage scenarios.

4. Construction of a crowd-sourced platform for the collection of various analysis results of software packages: A platform that enables users to contribute their analysis results for software packages through a maven plugin is established. Users can also retrieve existing pre-computed results from the platform.

Chapter 2

Related Work

In this chapter, we delve into the technologies associated with the crowd-sourcing platform. We begin by introducing the primary methods employed in achieving consensus within the crowd-sourcing system. Subsequently, we delve into how crowd-sourcing platforms incentivize individuals to complete tasks. Furthermore, we outline the applications of crowd-sourcing systems as an effective means of data collection across various domains. Lastly, we elucidate the concept of dependency identifier used in package managers, which we leverage in this project to design a hierarchical file system.

2.1 Consensus in Crowd-sourcing

Geiger categorizes crowd-sourcing systems into two types: individual and collective approaches, based on whether they benefit from each contribution in isolation[13]. In individual approaches, each worker's contribution is treated independently, and the correctness of the answer can be assessed in isolation. Criteria or verification rules for evaluating the answer exist in advance. An example of this approach is eBird, a platform where citizens contribute their observations for scientific research and bird population conservation[32]. In eBird, the reliability of submitted results is determined through automatic evaluation using species count limits for specific dates and locations. On the other hand, collective approaches require collecting a certain number of results before making a selection. The focus is on aggregating multiple contributions to arrive at a consensus or a final decision.

Inferring the correct answer from the answers submitted by multiple workers is a critical challenge in crowd-sourcing. As with other areas of disagreement, the most straightforward solution is Majority Voting. According to a study by Neto et al[28], Majority Voting is the most commonly used method for selecting the correct answer. Out of the 57 surveyed projects, 41 of them opted for Majority Voting as the approach for determining the correct answer.

However, Majority Voting also has a significant drawback when workers have different levels of qualities as it regards all workers as equal[36]. Castano et al. proposed a framework called LiquidCrowd, which leverages consensus and trustworthiness techniques to manage the execution of collective tasks[5]. Unlike majority voting, LiquidCrowd introduces a

differentiated approach toward the results submitted by workers. Recognizing that workers vary in their expertise and reliability, LiquidCrowd assigns a Trustworthiness value to each worker. This value is dynamically updated based on whether their submitted answers are selected as ground truth. By considering the Trustworthiness values, LiquidCrowd is able to identify and prioritize reliable and expert workers, thereby mitigating the potential negative impact of novice or untrustworthy workers.

In our system, the task results are automatically generated by the client-side code, eliminating the need for users to make judgments based on their expertise. Therefore, we have chosen the commonly used method in crowdsourcing systems, Majority Voting, as one of the consensus mechanisms. However, considering the potential for computational errors (although unlikely) and the presence of malicious users, we also intend to provide a method for identifying reliable users. Inspired by the principles of LiquidCrowd, we have incorporated a weight-based consensus mechanism in our crowdsourcing system.

However, this is not the sole method to estimate worker performance and identify unqualified workers. Gold standards are commonly employed in crowdsourcing systems to identify malicious workers[29]. They consist of questions with known answers that are provided to workers alongside regular questions[12]. If a worker fails to provide the correct answer for a gold standard question, they are flagged as untrustworthy. Ordinarily, gold standards are subtly incorporated into tasks alongside regular questions, making it challenging for workers to foresee or identify these gold questions[18]. In addition, having a gold standard set with only a small number of questions is not beneficial. On the other hand, setting a large number of gold-standard questions would increase the server load and deviate from the purpose of our crowdsourcing system design. Therefore, we have decided not to adopt gold standards in our system. Instead, we have been inspired by this approach and have designed a consensus integration method called Challenge and Response. By distributing unpredictable tasks to clients, we aim to enhance the reliability of the system without relying solely on gold standards.

2.2 Incentives

The incentives behind crowd participation play a pivotal role in the realm of crowdsourcing. Given the broad spectrum of applications ranging from rudimentary to intricate tasks, the motivations fueling the crowd vary extensively, contingent upon the specific characteristics of the task[19].

Designing appropriate reward strategies can significantly influence the productivity of the contributors and results produced by crowdsourcing systems[3]. According to Hosseini et al., incentives in crowd-sourcing can be categorized into two types: intrinsic and extrinsic[20]. Intrinsic incentives are inherent within the crowd, while extrinsic incentives are provided by the crowdsourcer in the form of financial rewards or other external motivations.

Not all individuals are motivated solely by monetary rewards when completing tasks. Ryan and Deci [30] define intrinsic incentives as ”doing an activity for its inherent satisfactions rather than for some separable consequence.” Sometimes people engage in tasks

for entertainment purposes. von Ahn and Dabbish introduced the concept of Games-With-a-Purpose (GWAP)[34], where games are developed to collect human intelligence through tasks that entertain the players. This allows individuals to derive pleasure while the crowd-sourcing platform obtains the desired results. For example, Foldit[6] is an online game from the University of Washington that allows players to solve puzzles related to protein folding. By leveraging human problem-solving and pattern recognition skills, it has made significant contributions to modeling proteins that have long puzzled the scientific community. People also engage in crowd-sourcing tasks for the purpose of learning and acquiring knowledge. For instance, through platforms like Duolingo, individuals learn new languages while also contributing to the translation of web content[3].

In the case of complex tasks, extrinsic incentives, particularly financial incentives, tend to be more dominant than intrinsic motivations[19]. Silberman et al. [31] emphasize the significance of financial incentives over other motivations, noting that the majority of participants do not engage in tasks solely for entertainment or leisure purposes. This is further supported by the findings of Neto et al. [28], who observed the predominance of monetary incentives in the surveyed crowdsourcing platforms. Clearly, monetary rewards strongly appeal to the crowd, motivating them to actively participate and complete tasks. Harris [17] observes that appropriate financial incentives play a crucial role in motivating participants to provide more accurate judgments in tasks such as resume review. However, it is important to note that financial incentives can also have unintended consequences. Kim et al. [25] highlight that from a worker's perspective, it may not be economically sensible to spend more time selecting a superior option when the potential gain is not significant. Workers may prioritize completing tasks quickly to maximize their earnings, especially when the potential reward is low. This behavior can lead to the presence of malicious users or a lack of attention to task quality. To address this, requesters need to design systematic approaches to encourage workers to perform tasks properly and discourage suboptimal behaviors.

Our crowdsourcing platform utilizes external incentives but does not provide monetary rewards. In our platform, completing precomputation tasks and accessing precomputed results from the server are considered necessary and sufficient conditions. Users gain the privilege of using the platform by completing a set of dependency analysis tasks. As more projects are analyzed, users gain access to a greater number of precomputed answers from the system, reducing the amount of work they need to perform themselves. This enhances the user experience and creates a positive feedback loop, as users benefit from the increasing availability of answers.

2.3 Crowd-Sourcing Systems in Different Fields

Crowd-sourcing has found applications in a multitude of fields, from knowledge gathering and software development to health sciences and astronomy.

In the realm of knowledge gathering, Wikipedia, one of the largest online encyclopedias, exemplifies the application of crowd-sourcing, harnessing the collective intelligence of countless individuals across the globe [14]. Similarly, OpenStreetMap has utilized crowd-sourcing to great effect in the field of mapping and cartography, creating a free and editable

map of the world [15].

The field of health sciences has also seen significant benefits from crowd-sourcing. Crowd-sourcing tasks in medical image analysis are diverse, ranging from classification, where the crowd assigns a label to an entire image or a part of the image, to localization or segmentation, where the crowd delineates the boundary of an entire healthy structure or an abnormality such as a lesion. Some studies also involve the crowd in less standard tasks, such as determining correspondence between pairs of images or deciding which image is more similar to a reference image[37]. Irshad [23] concludes that crowdsourced image annotation is a highly-scalable and effective method for obtaining nuclear annotations for large-scale projects in computational pathology.

In the realm of astronomy, the Galaxy Zoo project has enabled the classification of galaxies by harnessing the collective efforts of volunteers[26]. This crowd-sourcing project has led to several significant discoveries and has made valuable contributions to our understanding of the universe.

In the field of software development and analysis, crowd-sourcing has been leveraged to improve the efficiency and effectiveness of various tasks. Bug reporting, fixing, and testing are areas where crowd-sourcing has shown significant promise. Platforms like Bugcrowd and HackerOne have utilized crowd-sourcing for vulnerability identification, providing bounties for reported issues [11]. Similarly, Topcoder and GitHub have harnessed crowd-sourcing for software development and collaborative coding, respectively [7].

These examples demonstrate the power and potential of crowd-sourcing across various domains. They highlight the ability of crowd-sourcing to harness collective intelligence and scale up efforts, enabling large volumes of tasks to be accomplished efficiently.

2.4 Dependency Identifier

In the realm of open-source software (OSS), the need for a coherent system to mark distinct versions of a package is paramount due to the constant updates by maintainers.

Semantic Versioning is a method that assigns meaningful version numbers to distinct software releases[16]. It has gained popularity as a strategic approach for managing package evolution, employing a structured format, namely ”major.minor.micro”, to denote different versions of a package[9]. Package users can determine certain compatibility guarantees through the structure of version numbers in packages. Semantic Versioning stipulates an increment in the major version for incompatible API changes, the minor version for the addition of backward-compatible functionalities, and the micro version for backward-compatible bug fixes[33].

Building on the concept of Semantic Versioning, package managers adopt this versioning scheme as a part of their package identifiers. For some package managers, like npm, a package is identified by its name and version. For instance, lodash@4.17.21 refers to the lodash library's version 4.17.21. While in Maven, packages are uniquely identified by a triplet consisting of ”groupId:artifactId:version”[24]. In this work, as a crowd-sourcing platform developed for Maven users, we use the same version number structure to label a package's

unique release. Additionally, when storing the computation results of various tasks in the file system, we have designed a hierarchical structure based on this naming convention.

Chapter 3

Design and Implementation

To build a crowd-sourcing platform for software package analysis, it is essential to establish a clear understanding of the platform's workflow. Figure 3.1 illustrates the two main components of the platform: the Maven plugin running on the client side and the back-end server facilitating information exchange. When a client wishes to obtain the analysis results for a particular package, they send a request to the platform, and the server responds by providing the results. However, if the results are not yet available, the client is required to perform the package analysis themselves and then upload the results to the server. This completes a basic pre-computation task request. Additionally, we plan to implement the functionality for the server to allocate pre-computation tasks, initiating requests to clients who will then perform the calculations and upload the results.

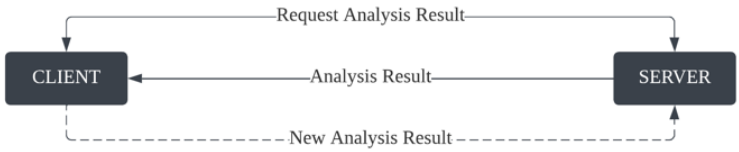

Figure 3.1: Work Flow of the Crowd-sourcing Software Analysis Platform

To realize these functionalities, it is crucial to design a system that effectively coordinates the interactions and decisions made by the server among the various data sources. In this chapter, we will provide a detailed explanation of the system design, including the domain model (3.1), the design model (3.2), consensus integration (3.3), and consensus mechanisms (3.4).

3.1 Domain Model

As shown in Figure 3.2, our system domain model is structured around four key entities: the Client, Server, Task, and Consensus Strategy.

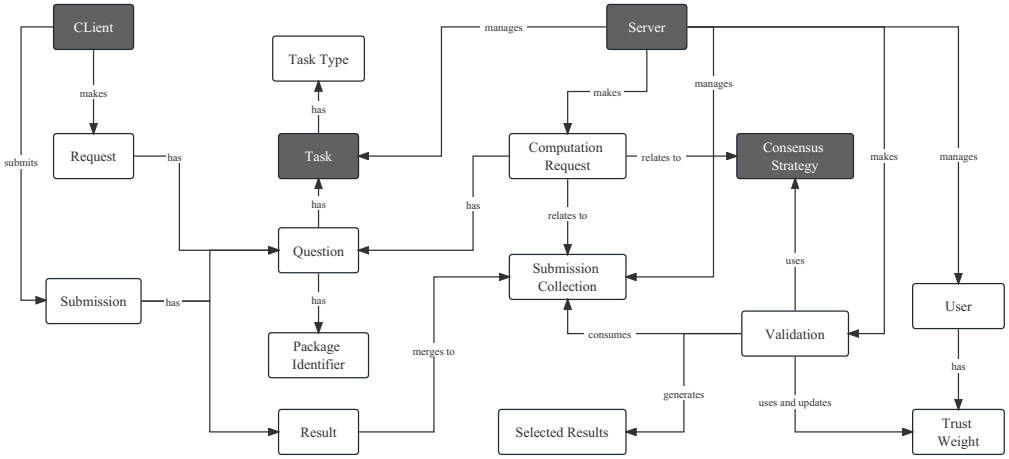

Figure 3.2: Domain Model

Client The individual or team using our platform to analyze their software packages. Clients request analysis results for their packages and contribute their results to the platform via the Maven Plugin.

Server The core of our platform, which serves as the central hub for data collection and distribution in our platform. It manages the task types provided by the platform, user information, and the correspondence between users and their contributed analysis results. The server handles the aggregation and selection of results using various consensus mechanisms. When a trigger condition is met, it employs the consensus mechanism to choose the correct result from the pool of contributed results. This selected result is then stored and can be directly returned when the server receives a subsequent request for the same task from a client.

Task Task is used to label different types of pre-computation workloads, such as ”size” for debugging purposes and ”call graph” for experimental analysis. The Task together with the specific package name forms the basis of a pre-computation workload. It serves as the main body of the client's request to the server and is an important identifier for sharing analysis results. The server also categorizes and stores submitted results based on the Task name, enabling efficient management and retrieval of different types of analysis results.

Strategy The Strategy describes how the server's decision-making system operates. It consists of two consensus integration methods and three consensus mechanisms. The former influences the communication protocol between the plugin and the server, determining whether the analysis results submitted by clients are generated by the clients themselves as pre-computation workloads or allocated by the server as pre-computation workloads. The latter determines how the server selects the most appropriate result among the various results contributed by different clients for the same pre-computation workload.

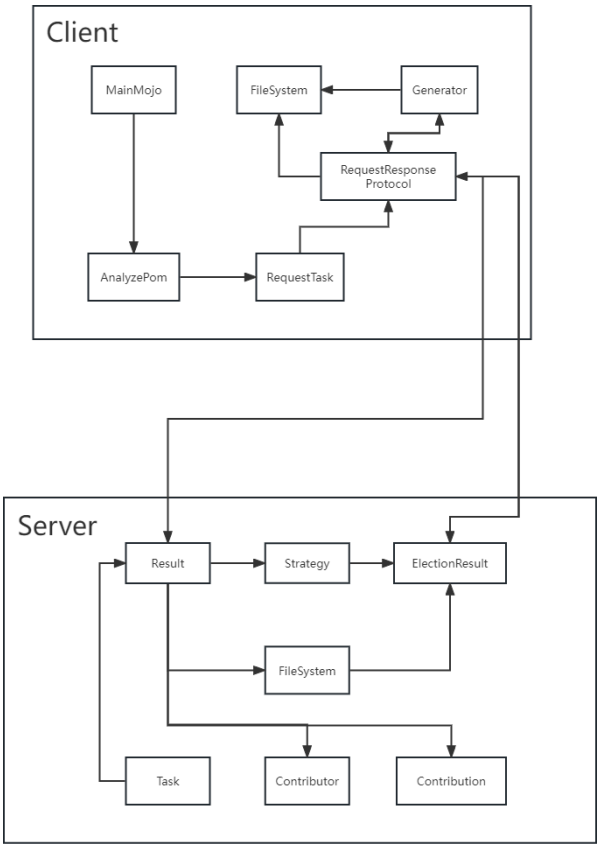

Figure 3.3: Design Model

3.2 Design Model

A design model serves as a graphical or conceptual representation of a system's structure and behavior, guiding the development process. It helps developers and stakeholders visualize the architecture, functionality, and interaction of system components, ensuring a clear understanding and facilitating collaboration throughout the project lifecycle. As shown in Figure 3.3, our system can be divided into two main components: the Client-side Plugin and the Server-side. Here's an introduction to their specific components and their interconnections:

Client In our design model, the client-side program starts with the MainMojo, which is a crucial component of the custom Maven plugin. Each Mojo represents an execution goal in Maven and serves as the entry point of our program. When the plugin starts running, the first task is to analyze the project's configuration file, the pom.xml, to retrieve the list of dependencies. Then, combining each dependency with the corresponding task name, a RequestTask is formed. The RequestResponseProtocol module is used to facilitate commu-

nication with the server by exchanging information.

Once a response is received from the server, it is parsed and if it contains valid results, they are stored in the local file system. If the response is empty, it means that the requested results are not available, and the client needs to perform the specified analysis locally. The obtained results are then shared with the server using the RequestResponseProtocol module. This completes one cycle of the task request for a particular dependency. The process is repeated for all the dependencies that require analysis, following this workflow.

Server Within the server, there are two modules responsible for communication with the plugin. The ElectionResult module handles the requests received from the clients, queries the database, and returns available analysis results to the users. If no results are currently available, it returns an empty response. On the other hand, the Result module is responsible for receiving the analysis results submitted by the clients. It stores the results in the data system based on the task type and package name and also records each submission as a contribution along with its corresponding contributor.

When the Result module receives a new submission, it checks the Strategy to determine whether it needs to update the ElectionResult. The Strategy plays a crucial role in deciding how the server selects and updates the available results. This ensures that the server maintains an up-to-date and accurate pool of analysis results for the clients to access.

3.3 Consensus Integration

Effective communication protocols are crucial for facilitating efficient and reliable interaction between the back-end server and the client-side Maven plugin in the context of crowd-sourced collection and analysis of software packages. These protocols define the rules and procedures for data exchange, command execution, and response handling, ensuring coordination and collaboration between the different components of the crowd-sourced platform.

In this study, two consensus integration that represents two communication protocols have been designed: In Process integration and Challenge and Response integration. These protocols offer different approaches to communication and have unique features tailored to specific objectives. As their names imply, these two communication protocols represent two different mechanisms for task distribution and result collection in the back-end server. In the following sections, we will introduce each of them in detail.

3.3.1 In Process

From the perspective of the server, the In Process integration is a passive approach where the server receives the results without actively selecting tasks to send to the client plugin. The server acts as a recipient, collecting all the results uploaded by the client. As depicted in Figure 3.4, the detailed workflow is as follows:

1. Artifacts List Generation: The plugin generates a list of artifacts based on the project's configuration file (pom.xml).

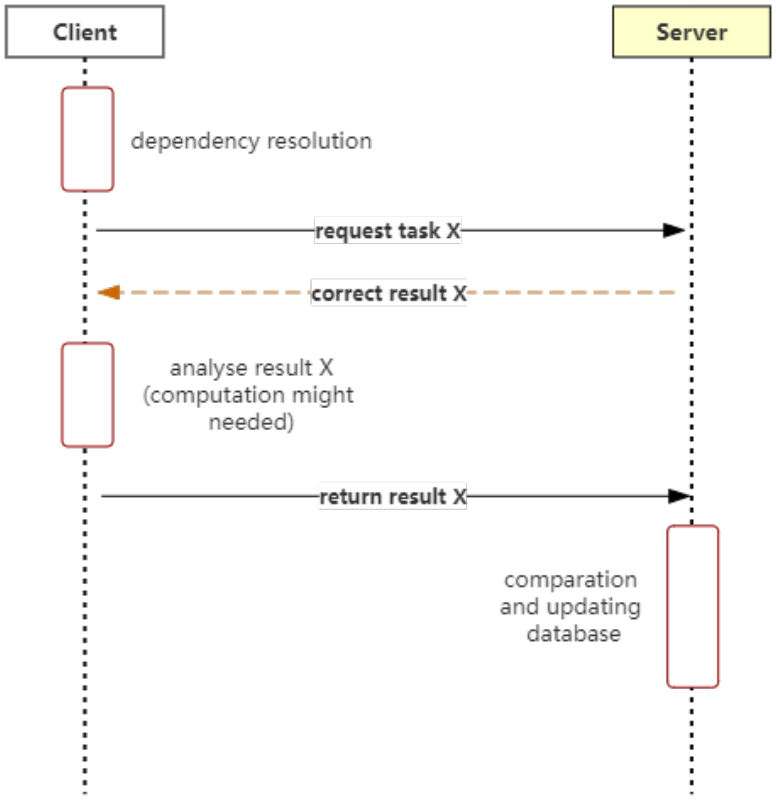

Figure 3.4: In Process Protocol

2. Task Upload: The client plugin initiates communication with the server and combines each artifact and task name from the list then send it to the server.

3. Result Retrieval: The server responds to the plugin by providing the pre-computed results for the uploaded tasks. If the server already possesses the result for a particular task, it directly returns the corresponding result to the client. However, if the server does not have the result, it returns an empty result to the client.

4. Result Parsing: The plugin parses the received results from the server. If a valid result is received, the plugin will store it for future use. However, if an empty result is received, the client needs to compute the result for that task locally and upload it to the server.

5. Result Storaging and Comparison: The server receives the results uploaded by the clients and stores them using a hierarchical structure and maintains a database to keep track of the uploaded results and their uploader. The server periodically compares the

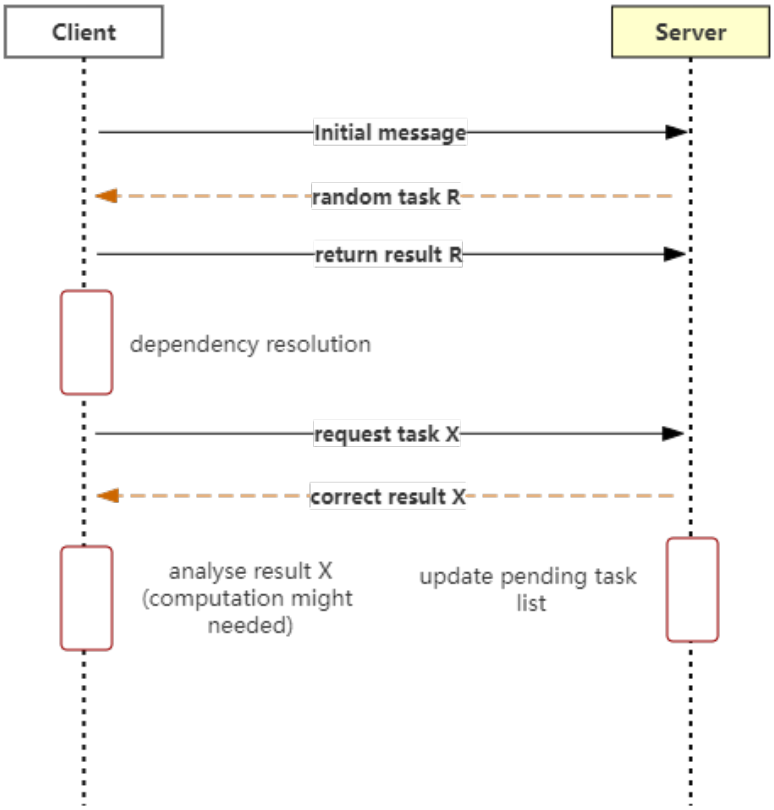

Figure 3.5: Challenge and Response Protocol

uploaded results for each dependency and task and selects the correct answer based on the chosen consensus method.

3.3.2 Challenge and Response

Unlike the In Process integration, Challenge and Response, from the server's perspective, is considered an active approach as it proactively assigns a task to the user before the user formally initiates the task request. The user is required to complete the computation and upload the answer before proceeding with subsequent task requests. As depicted in Figure 3.5, the detailed workflow is as follows:

1. Initial Stage: The client plugin sends a message indicating its intention to establish a connection with the server using the Challenge and Response Protocol.
2. Random Task Assignment: The server randomly selects a task from the list of pending tasks and sends it to the client. The client performs the computation as required and returns the result to the server.

3. Artifacts List Generation: The plugin generates a list of artifacts based on the project's configuration file (pom.xml).

4. Task Upload: The client plugin initiates communication with the server and combines each artifact and task name from the list then send it to the server.

5. Result Retrieval: The server responds to the plugin by providing the pre-computed results for the uploaded tasks. If the server already possesses the result for a particular task, it directly returns the corresponding result to the client. However, if the server does not have the result, it returns an empty result to the client. And adds this task to the list of pending tasks.

6. Result Parsing: The plugin parses the received results from the server. If a valid result is received, the plugin will store it for future use. However, if an empty result is received, the client needs to compute the result for that task locally. However, unlike in the In Process Protocol, there is no need to upload this result to the server in the Challenge and Response Protocol.

7. Pending Tasks Updating: The server maintains a pending task list that includes all the requests that have not yet been answered. This list serves as a reference for selecting random tasks to assign to clients in the second step of the process. If the server does not have the computed result for a specific request (request X) and it is not present in the pending task list (indicating that the question is being asked for the first time), request X is inserted into the pending task list.

3.3.3 Comparison

Under the In Process integration, whenever a user utilizes the plugin, each task can fall into one of two scenarios: obtaining the answer directly or computing the answer locally and subsequently uploading it. This means that every task a user performs using the plugin involves interaction with the server's database. The advantage of this approach is that it expedites the growth of content in the server's data, as each task that is requested but currently lacks a result will obtain a result during that particular user interaction. However, this also creates an opportunity for malicious users to exploit the system. They can selectively attack specific tasks associated with an artifact, repeatedly requesting those tasks to upload their own answers to the server, thereby potentially influencing the results following the execution of the consensus algorithm. To mitigate this issue, we have proposed the Challenge and Response integration as an alternative solution.

In short, the advantage of In Process integration is its faster saturation speed, while Challenge and Response integration offers higher security.

3.4 Consensus Mechanisms

Once the platform collects the results uploaded by users, the next challenge is to select reliable answers from them. In our system, we offer three Consensus Mechanisms, namely:

First-come, first-served Consensus, Majority Voting, and Trust-based Voting. In the following subsection, we will introduce their concepts and principles, explain how they are implemented in the system, and qualitatively analyze their advantages and disadvantages.

To provide background knowledge for the upcoming explanations, let's first introduce the components related to the implementation of these mechanisms in the system:

- Result Controller: Responsible for receiving the computation results uploaded by clients via the plugin and determining the subsequent processing flow based on the current consensus mechanism.
- Election Result: Stores the computed results that have been selected through a specific consensus mechanism for a particular task of a package.
- Contributor: Records user names and other relevant information.
- Contribution: Records the file names and the information of the contributors for each uploaded record.

3.4.1 First-come, First-served Consensus

The First-come, first-served (FCFS) Consensus method operates by accepting the answer provided by the first user who submits a response to a given task. The process is illustrated in Figure 3.6. Subsequent submissions from other users are not considered for consensus in this approach, as the accepted answer is determined solely based on the initial submission.

Implementation The implementation of the FCFS consensus mechanism is straightforward. In the proposed system, when the Result Controller receives a new Computation Result from the client, it consists of three parts: the Task name, the specified Package name, and the computation result itself. The server's tasks can be divided into two parts: storage and consensus status updates.

First, upon receiving the new computation result, the Result Controller checks the Contributor table in the database to see if the user is uploading for the first time. If the user is new, their information is inserted into the Contributor table. Although the Contributor table is not directly involved in the execution process of the FCFS consensus mechanism, maintaining this table ensures consistency in handling potential user management needs and the overall execution flow of various consensus mechanisms.

Next, the Task type and specified Package name, along with the uploader's ID, are stored in the Contribution table, where the first two serve as a unique identifier for the computation result. Then, based on this newly inserted identifier, the computation result is stored in the file system's hierarchical structure as a JSON format.

With the storage task completed, the next step is to check if the consensus mechanism needs to be executed. Since the FCFS mechanism is currently used, each new computation result received corresponds to a unique Question composed of the Task and Package name. This implies that the Election Result table needs to be updated by inserting the identifier of this computation result as a new entry. This way, when future requests with the same

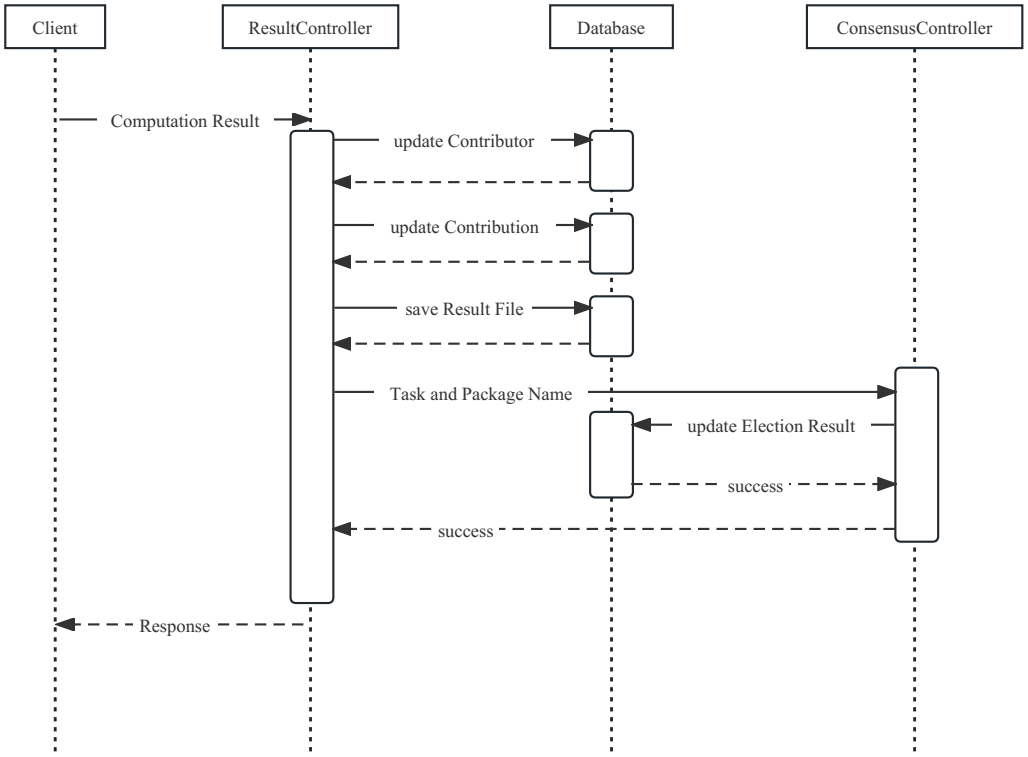

Figure 3.6: Illustration of the First Come First Serve Mechanism

Question arise, the corresponding result identifier can be directly retrieved from the Election Result table, facilitating the return of the result to the client.

Advantages

- Simplicity: This method is straightforward to implement, as it does not require complex algorithms or calculations.
- Speed: As the first submitted answer is considered correct, the consensus can be reached quickly, reducing the time it takes to complete tasks and make decisions.
- Low computational overhead: This method has minimal computational requirements since it does not involve processing or comparing multiple submissions.

Disadvantages

- Accuracy concerns: The accuracy of this method relies heavily on the first user's knowledge and judgment, which may not always be reliable. As a result, this consensus method can be prone to errors if the first user submits an incorrect answer.

- Limited user involvement: This method does not take into account the input of multiple users, which might lead to missing out on valuable insights and potentially more accurate answers from other participants.

- Vulnerability to gaming and malicious behavior: Since the first submitted answer is accepted as correct, the system may be susceptible to manipulation by users who intentionally submit incorrect answers or spam.

3.4.2 Majority Voting

The Majority Voting consensus method operates by collecting answers submitted by multiple users for a given task. Each user's submission is considered a vote for a particular response. After collecting the votes from all participating users, the system calculates the frequency of each submitted answer. The answer that receives the highest number of votes is considered the correct response. In cases of a tie, additional measures can be implemented, such as selecting a random answer among the tied options.

Implementation In contrast to FCFS, the decision-making process in Majority Voting is more complex. The process is illustrated in Figure 3.7. When the Result Controller receives a new computation result from the client, it follows a similar procedure as FCFS to determine if it needs to insert a new entry in the Contributor table and add the information of the task type and package name to the Contribution table. However, in Majority Voting, the process of updating the Election Result table does not happen immediately.

Upon inserting the computation result into the Contribution table, the Result Controller scans all the submitted results for the current Question to provide a new unique result ID (by incrementing the previous maximum result ID) for the new result, creating its identifier. Subsequently, the computation result is stored in the hierarchical file system based on this identifier.

However, in Majority Voting, the decision-making is delayed until a certain number of computation results (voting threshold) have been collected. After saving each uploaded result, the Result Controller queries the Contribution table again. If the number of results for the current Question reaches the voting threshold, the Consensus Controller module is triggered. The Consensus Controller reads all the submitted results stored under the directory composed of the task name and package name. It then compares all the results to identify the differing computations and tallies the frequency of each unique result. The computation result with the highest frequency is selected as the consensus answer and is updated in the Election Result table.

Advantages

- Improved accuracy: By considering the input of multiple users, this method can potentially lead to more accurate results, as it relies on the collective wisdom of the crowd rather than a single user's judgment.

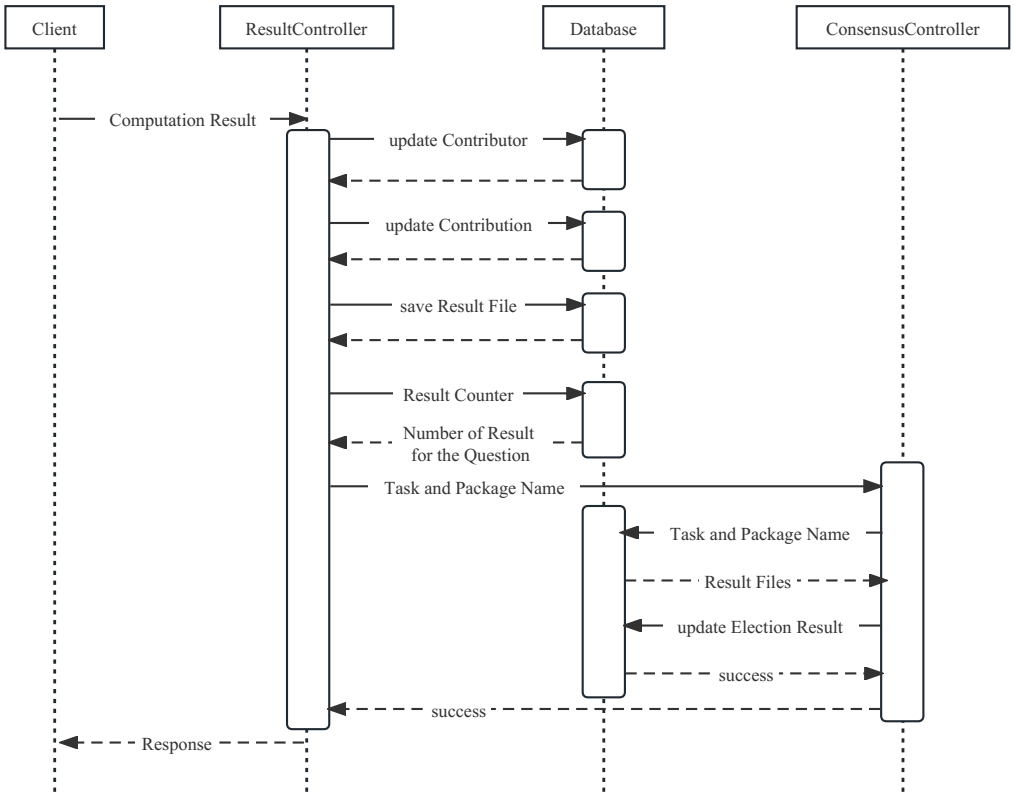

Figure 3.7: Illustration of the Majority Voting Mechanism

- Enhanced user involvement: Majority Voting encourages participation from a larger number of users, which can lead to a more diverse range of perspectives and insights.
- Resilience to manipulation: This method is less susceptible to gaming and malicious behavior, as any single user's submission has less influence on the consensus outcome compared to the First-come, First-served method.

Disadvantages

- Increased complexity: Implementing Majority Voting requires more complex algorithms and calculations compared to the First-come, First-served method, as it involves processing and comparing multiple submissions.
- Slower decision-making: This method can take longer to reach a consensus, as it requires collecting and analyzing multiple submissions before determining the correct answer.

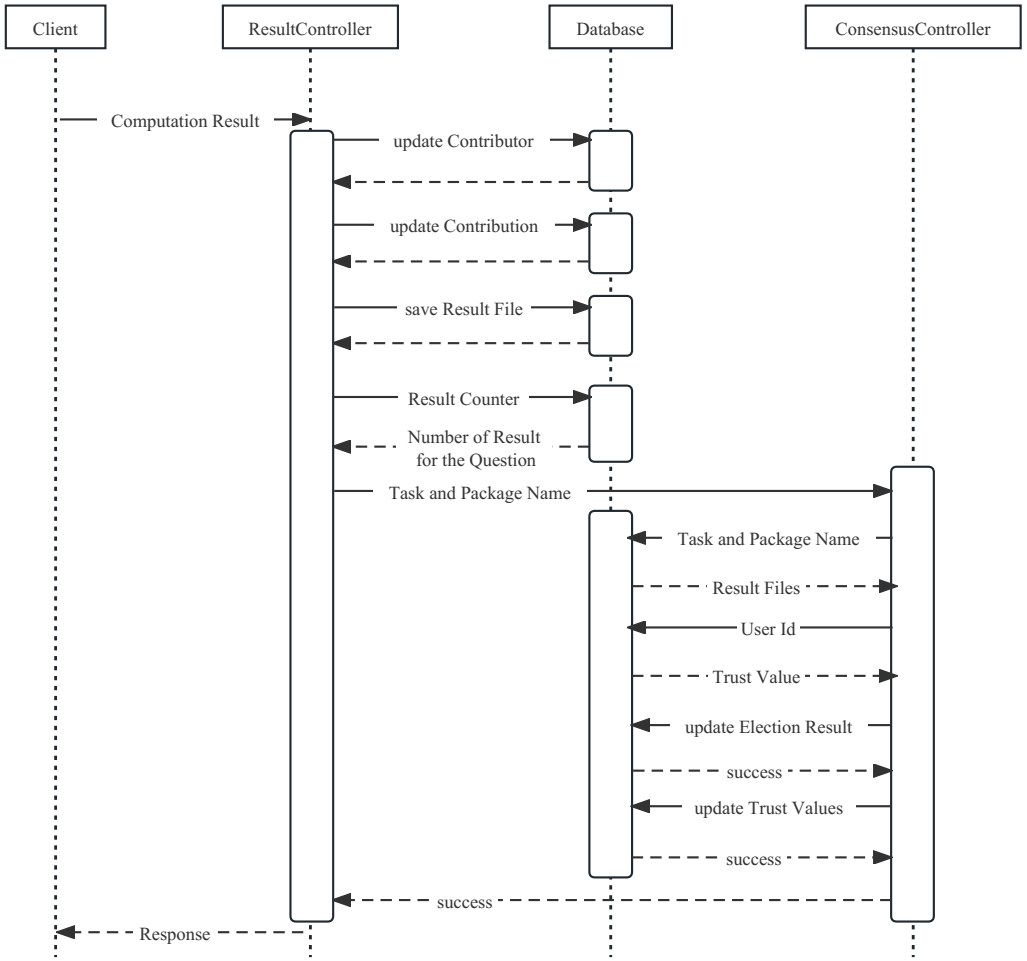

Figure 3.8: Illustration of the Trust-based Voting Mechanism

3.4.3 Trust-based Voting

The Trust-based Voting consensus method operates by collecting answers submitted by multiple users for a given task, similar to the Majority Voting method. However, in Trust-based Voting, each user's submission is assigned a weight based on their reputation within the platform. This weight reflects the level of trust or credibility assigned to the user's input. After collecting the weighted votes from all participating users, the system calculates the total score for each submitted answer. The answer with the highest accumulated score is considered the correct response.

Implementation Trust-based Voting, as an evolved algorithm of Majority Voting, shares a close resemblance in its processing flow. The process is illustrated in Figure 3.8. Similar to Majority Voting, it involves updating the Contributor table and Contribution table when

new computation results are uploaded by clients. Upon reaching the voting threshold, the Consensus Controller is activated to make a selection from all submitted answers for a particular question. However, Trust-based Voting introduces a differentiation in the weight each user's submission carries, termed as the Trust Value. In Majority Voting, all users' Trust Values can be considered as 1, simplifying the comparison and statistical process. In Trust-based Voting, the impact of each user submitting the result needs to be considered. This necessitates an additional step in the Consensus decision-making process where the Trust Value of the user submitting each result is obtained and used to calculate a score for each submission. The computation result with the highest score is selected as the consensus answer and is updated in the Election Result table.

However, the role of the Consensus Controller doesn't conclude here. It further involves updating the Trust Value of each participant based on their selection results. Users who provide correct answers are deemed relatively trustworthy and have their Trust Value increased, while those providing incorrect answers see their Trust Value decrease. For this system, we draw inspiration from the Trustworthiness update method in LiquidCrowd[5]. Initially, during the onset of crowdsourcing activities, the worker's Trust Value τ is set to an initial value of $\tau = 1$. Upon each task commitment (time $t+1$), the Trust Value of a worker W is updated according to the following scheme:

$$\tau_W^{t+1} = \begin{cases} h * \tau_W^t + (1-h) * a, & if W \in S_C \\ h * \tau_W^t, & if W \notin S_C \end{cases} \tag{3.1}$$

In this context, τ_W^t refers to the Trust Value of worker W before task execution at time t, while h symbolizes the weight assigned to historical records. Moreover, a signifies the reward of providing a correct answer, and S_C represents the collection of workers who have accurately answered. For a worker, the computation of τ_{t+1}^W considers a fusion of two vital factors: the accumulated trustworthiness built over the course of all previously executed tasks and the award earned from the last task. This dynamic process blends historical performance with immediate achievement to determine the Trust Value at time $t+1$.

Advantages

- Enhanced accuracy: By taking into account the expertise or reputation of users, Trust-based Voting can potentially lead to more accurate results, as it gives more weight to the opinions of more knowledgeable or experienced users.
- Resilience to manipulation: Trust-based Voting is less susceptible to gaming and malicious behavior, as it reduces the influence of inexperienced users or users with low reputation scores.

Disadvantages

- Increased complexity: Implementing Trust-based Voting requires more complex algorithms and calculations compared to the Majority Voting and First-come, First-served methods, as it involves processing and comparing weighted submissions.

Table 3.1: Mix Strategies

	Consensus Integration	
Consensus Mechanism	In Process	Challenge and Response
FCFS	Combination 1	Combination 4
Majority Voting	Combination 2	Combination 5
Trust-based Voting	Combination 3	Combination 6

- Slower decision-making: Similar to Majority Voting, this method can take longer to reach a consensus, as it requires collecting and analyzing multiple weighted submissions before determining the correct answer.
- Challenges in determining weights: Assigning appropriate weights to users can be challenging, as it may require a robust mechanism to evaluate expertise or reputation accurately and fairly.

3.4.4 Comparison

The advantages of the three Consensus Mechanisms are evident, and they each have different use cases. The FCFS approach has a notable advantage in rapidly growing the answer repository of the system. However, it also compromises security since the first submitter holds significant weight, making the back-end database vulnerable to potential attacks if the system's user credibility is not high enough. Thus, FCFS is suitable for small-scale settings where all users are trusted. However, for a platform targeting the general public, the latter two voting mechanisms are preferred.

With Majority Voting, a simpler approach is employed as it only requires comparing all the submitted results without considering the attributes of each user who submitted the answer. On the other hand, Trust-based Voting involves calculating the score for each result based on the different weights of users, which extends the computation time for the system to reach a consensus. Additionally, designing an appropriate weight updating formula is crucial for Trust-based Voting, as a well-designed formula can lead to more reliable system answers, while an ill-designed one may waste execution time without improving the system's reliability.

3.5 Compose Strategies

The two consensus integration methods and three consensus mechanisms mentioned earlier have their respective advantages and disadvantages. By combining them, it is possible to complement each other's shortcomings. By combining any consensus mechanism with a consensus integration method, we can have six different scenarios as shown in Table 3.1.

The In Process Integration approach maximizes resource utilization but becomes vulnerable to attacks from malicious users who can disrupt the final results by repeatedly requesting tasks to upload their own answers. Hence, unless the platform users seek extreme

data growth and rapid consensus achievement, Combination 1 is not recommended, as combining two vulnerable methods would make the system fragile. However, if the platform is used on a very small scale where each user can be ensured not to be malicious, Combination 1 will leverage its highly efficient advantage.

In general, the combination of In Process Integration approach and Voting mechanisms, i.e., Combinations 2 and 3, is a more suitable choice due to the Voting mechanisms' resilience to malicious behavior. Comparing these two combinations, Combination 2 has the apparent advantage of a low time complexity in the consensus algorithm execution. When there are many results to be analyzed or the server's performance is limited, Combination 2 can select the final result more quickly from the numerous answers submitted by users. If you desire a system that can effectively identify and mitigate the impact of malicious users, Combination 3 is a better choice.

As for the combinations involving the Challenge and Response Integration approach, it enhances the reliability of decision-making through the server's proactive task assignment. However, this may result in a slower saturation rate of the database. In Combination 4, when integrating the Challenge and Response with the FCFS approach, we can benefit from both methods. The Challenge and Response ensure proactive task assignment, maintaining accuracy and reliability in decision-making, while the FCFS approach ensures quick decision-making.

In Combinations 5 and 6, we integrate the Voting methods into the strategy. Before the number of participants for a particular task reaches a threshold, the task remains on the pending list and can still be assigned to future users for computation. However, our current Challenge and Response task assignment strategy assigns each client only one precomputation task in each request. This leads to slow database growth in Combinations 5 and 6, making them ineffective in providing efficient services. Therefore, unless platform users can propose alternative task assignment strategies that strike a balance between the additional computation time caused by users completing assigned tasks and the time for consensus achievement in the system, we do not recommend using these two combinations.

Chapter 4

Evaluation

In the previous chapter, we introduced the design of our crowd-sourcing platform. We hope that users can reduce duplicate calculations on the same task through this platform, thereby improving development efficiency. To investigate the impact of this platform on development efficiency, we will evaluate the performance of the platform with different consensus approaches from both qualitative and quantitative perspectives. In Section 4.1, we investigate the following question: How many projects are needed to saturate the crowd-sourcing system? Based on this, in Section 4.2, we analyze the improvement of development efficiency for the platform at different hit rates by investigating the following question: What is the performance gain for the pre-computation task at different hit rates? In Section 4.3, we conducted an analysis of the server-side performance and addressed the question: What is the trade-off of computation load for the server?

4.1 How many projects are needed to saturate the crowd-sourcing system?

In the initial stages of our crowd-sourcing platform, the server-side database is empty, with no pre-existing data available for users to access. As the platform gains traction and users begin to submit their answers for various tasks, the database accumulates more solutions, which can then be shared among other users. A crucial aspect of the platform's success and effectiveness lies in understanding the saturation point, where a sufficient number of projects have contributed answers, ensuring the database is well-populated and able to provide useful solutions for future users. To that end, we aim to investigate the following research question: "How many projects are needed to saturate the crowd-sourcing system?" By exploring this question, we seek to determine the critical mass of projects required to establish a comprehensive and efficient crowd-sourcing platform that maximizes its potential benefits for all users.

This question can be divided into two dimensions: horizontal and vertical analyses. The horizontal analysis focuses on examining the relationships between dependencies across different Maven projects and the vertical analysis investigates the differences in dependencies within the same project across various versions.

4.1.1 Data set

To answer the research question, we first need to prepare the data set. Our goal is to gather a comprehensive and representative sample of Maven projects to ensure that our research findings are robust and generalizable.

Due to the rapid development of open-source software, new libraries continue to emerge and gain popularity. Therefore, analyzing projects from a long time ago holds little significance in evaluating our system, as the dependencies they used may no longer be in use or have updated versions. The core principle of our project is to reuse computations for tasks that users repeatedly need, in order to minimize wastage of computational resources and time. It is advisable to analyze recently active projects to obtain overlapping relationships between their dependencies.

As we cannot directly obtain Maven projects through GitHub search, we need to filter the projects ourselves. First, we will use the GitHub API to retrieve Java projects since 2021 and gather their names and URLs. Then, we will visit the project repositories to check if there is a Maven project configuration file in the root directory for further evaluation. If the criteria are met, the project will be added to the data set.

Due to limitations on GitHub's search, a maximum of 100 pages of results can be returned for each search. Therefore, we need to divide the time period into smaller segments instead of directly searching from January 1, 2021. We grouped the projects into two-month intervals and filtered those with more than 10 stars. As a result, we obtained a total of 2144 projects.

4.1.2 Horizontal Analyses

In the horizontal analysis section, we will investigate the relationships between dependencies across different Maven projects, focusing on the overlap and differences in their dependencies. As the number of analyzed projects increases, the server-side database accumulates more answers, allowing subsequent users to reduce the number of tasks they need to compute locally and instead retrieve answers directly from the server. However, the degree of dependency overlap between projects varies, meaning that analyzing a new project may or may not increase the number of answers stored in the server (e.g. if the new project's dependencies are a subset of tasks already in the server).

Our goal in this section is to identify a saturation point where the majority of tasks can find answers, and further increasing the number of analyzed projects does not significantly improve the task hit rate. This is because different projects may have unique dependencies that are not used by other projects, which will not contribute to the saturation of the crowdsourcing system. By understanding the relationships between different Maven projects and their dependency overlaps, we can make informed decisions when designing and implementing our crowd-sourcing platform. This analysis will ultimately help us determine the most effective consensus approach for different project quantities.

Methodology

We analyzed 2144 projects and collected statistics on the package names, version numbers, and occurrence frequencies. We proceed by sequentially downloading the projects using the URLs from the dataset and reading their *Pom.xml* files to extract the corresponding dependency information. Following the statistical approach of Maven, we consider package names and version numbers together as identifiers for each dependency package. We maintain a table to record all encountered dependency packages along with their occurrence count. Upon analyzing a new project, the following steps are executed:

1. Traverse through all dependencies of the project.
2. If the dependency is present in the table, mark it as a ”Hit” and increment the occurrence count in the table.
3. If the dependency is not present in the table, mark it as ”Not Hit” and add the information of this new dependency to the table.
4. Once the traversal is complete, divide the number of ”Hit” dependencies by the total number of dependencies to obtain the hit rate for that particular project.

The analysis revealed that 10,191 different dependencies were used across these projects. The number of these artifacts may sound substantial, but it's important to note that the Maven Central Repository added 2,251,027 artifacts in 2022 alone[10]. Therefore, when compared to the total number of artifacts in the Maven Central Repository, the quantity we have collected is still relatively small. Additionally, based on other statistical data, we have created the following plots.

Results

Figure 4.1 illustrates the variation in the number of unique artifacts as the number of analyzed projects increases. As expected, it can be observed that the count of artifacts continues to rise as more projects are analyzed, but the rate of increase gradually slows down. To highlight this relationship more clearly, the change in curve slope is depicted in Figure 4.2. Overall, the slope of the curve consistently decreases, indicating a diminishing rate of new artifact additions and a trend toward saturation in the database.

For users, the hit rate is an important metric as it represents the proportion of pre-computed results they can obtain from the server. A higher hit rate indicates more computational savings for users. Therefore, we also calculated the hit rate for each project during the analysis process. The results showed significant fluctuations in this metric. This variation is due to the different numbers of dependencies in each project. For example, a project with only one dependency may have a hit rate of 1 if the pre-computed result for that task is available in the database. On the other hand, a project with 100 dependencies and 50 available pre-computed results would have a hit rate of only 0.5. However, as shown in Figure 4.3 after smoothing, we can still observe an increasing trend in the hit rate as the number of analyzed projects increases. As expected, this indicates that our crowd-sourcing

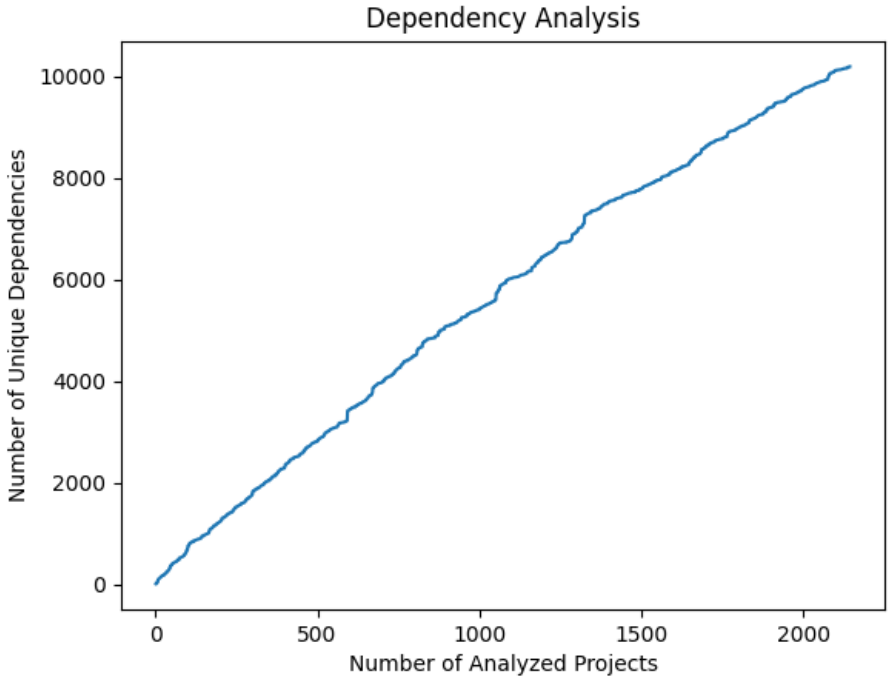

Figure 4.1: The Variation of Artifact Count with Increasing Number of Analyzed Projects

platform can provide better services with an increasing number of analyzed projects. However, due to version updates of artifacts, even if two projects use the same dependency, the subsequent project may use an updated version of the same named dependency. Therefore, the improvement in the hit rate is not linear.

So, what hit rate can users expect? To answer this question, we created a box plot 4.4 based on intervals of 300 data points. We divided the total of 2144 projects into eight groups, each containing 300 projects. Within these groups, we performed average value statistics to observe the trend of hit rates. We also notice that in each group, there are occurrences of both the maximum value 1 and the minimum value 0. Upon observing these data, we have realized that such extreme cases are typically caused by projects with a very limited number of dependencies. For instance, if a project has only one dependency and the analysis result for that dependency is present on the server, the hit rate becomes 1. Conversely, if a project has three dependencies and, unfortunately, none of them match, the hit rate becomes 0. Hence, to achieve a more accurate result, we performed average value statistics across a large number of projects. From the plot, we can observe that after the system analyzes a sufficient number of projects, the average hit rate for users tends to stabilize around 0.5.

The data presented above demonstrate that applying a crowd-sourcing platform can help users avoid redundant computations. However, the specific number of artifacts for which computation results can be obtained depends on the consensus method used. Therefore, we also conducted a frequency analysis of artifact occurrences, resulting in Figure 4.5.

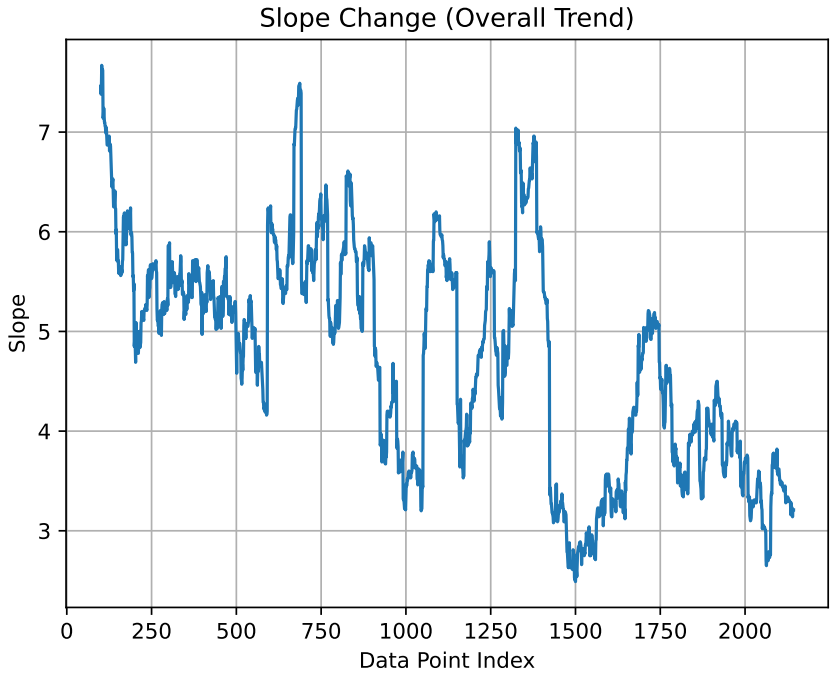

Figure 4.2: Slope change of Figure 4.1

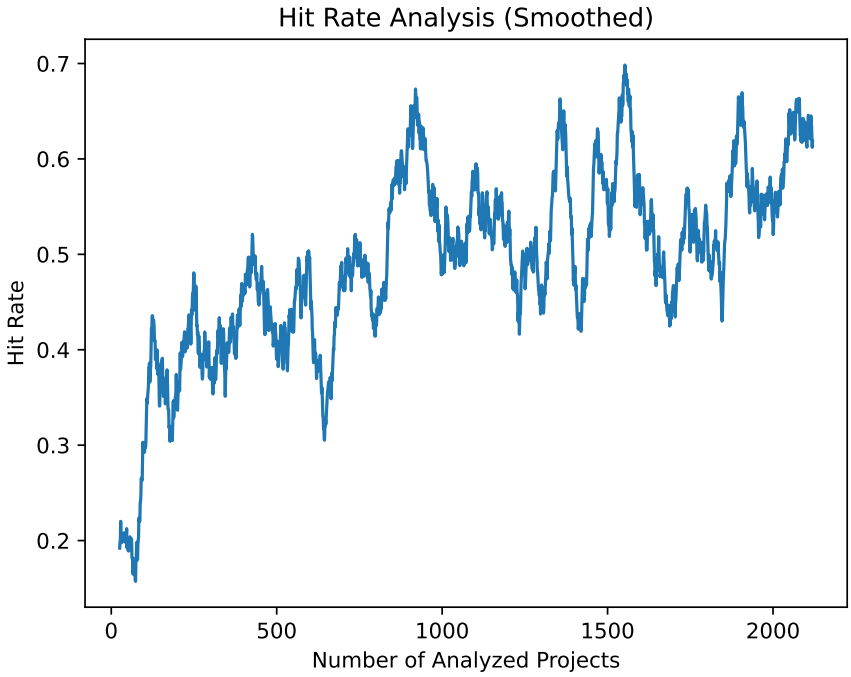

Figure 4.3: The Variation of Hit Rate with Increasing Number of Analyzed Projects

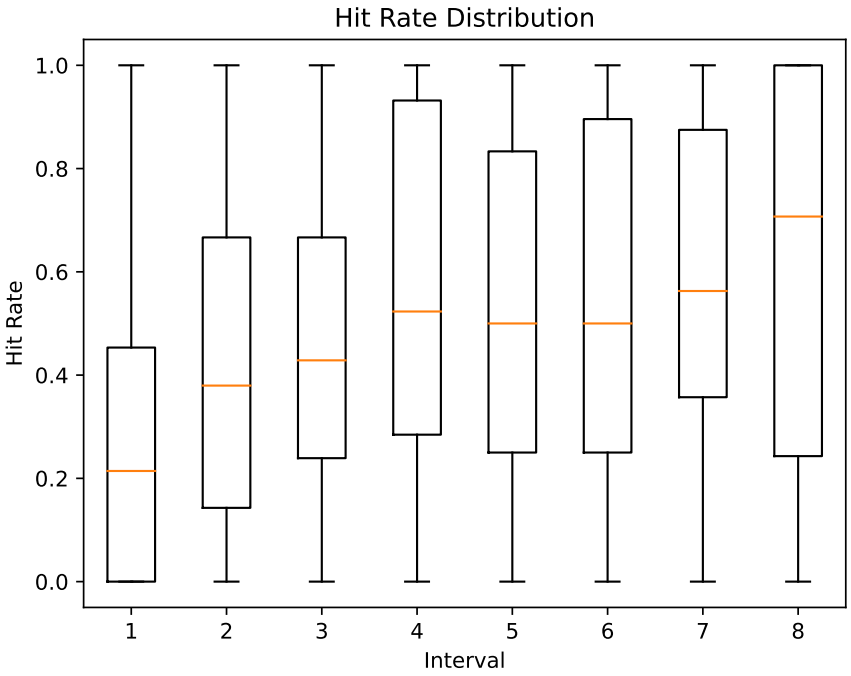

Figure 4.4: Box plot of Hit Rate

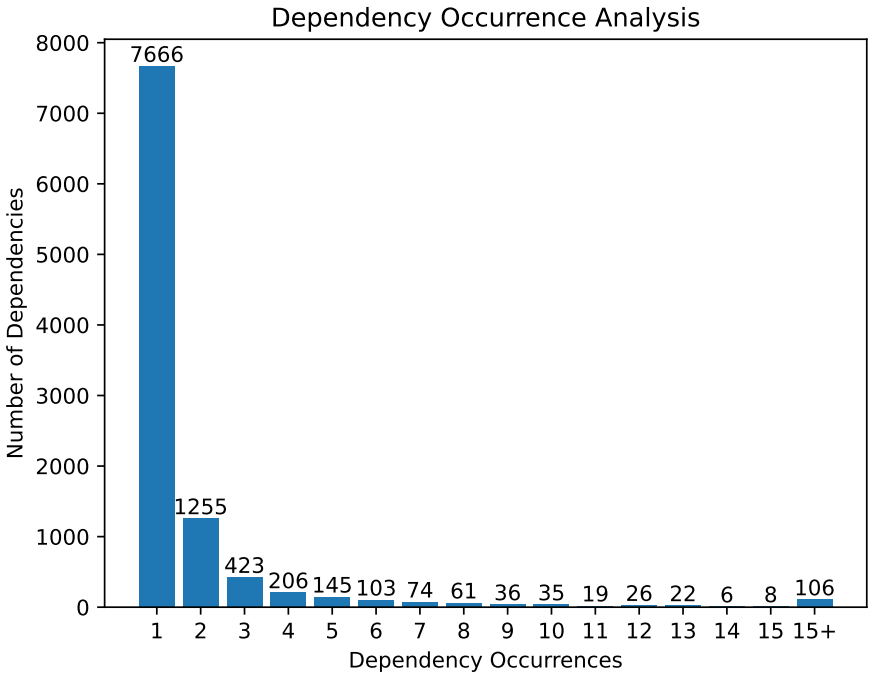

Figure 4.5: Artifacts Occurrences Frequency

From Figure 4.5, it can be observed that the majority of artifacts appear only once in the analyzed projects. This implies that only in cases where subsequent projects encounter these repeated artifacts, and when using the First-come, First-served Consensus, the calculation results for relevant tasks can be directly obtained from the server side. This result suggests that when users are reliable, the First-come, First-served Consensus offers significant performance advantages over the Voting Consensus. However, as a platform targeting the entire society, the reliability of users cannot be guaranteed. Therefore, the Voting Consensus becomes a safer option. If we set the minimum user requirement for voting to 5 individuals, only 4.67% of calculation results in the database can be utilized. If we relax this criterion to 3 individuals, then 11% of calculation results can be reused.

In addition, different consensus integration methods also have an impact on the saturation rate. When using the In Process Integration method, the server collects answers for all unresolved tasks each time. As a result, the saturation rate is generally higher compared to the Challenge and Response Integration method. Typically, we only assign a small number of tasks to each user or a number proportional to the total number of requests. However, this number should not exceed the total number of user requests and should ideally be much lower. Otherwise, users may lose motivation to participate. Therefore, while Challenge and Response Integration enhances system security, it also reduces the saturation rate.

4.1.3 Vertical Analyses

In the vertical analysis section, we will investigate the differences in dependencies within the same project across various versions. As projects evolve over time, their dependencies may change due to factors such as updates, new features, or optimizations. These changes can lead to variations in the tasks that users need to compute for different versions of the same project, affecting the efficiency and effectiveness of the crowd-sourcing platform.

Our goal in this section is to understand how dependency changes across different versions of the same project impact the saturation of the crowd-sourcing system under different consensus approaches. Through this experiment, we can observe whether software developers can significantly improve their efficiency by avoiding a large number of redundant computations when developing a project using our crowd-sourcing platform.

In this experiment, the analysis procedure remains the same as Horizontal Analyses, with the only difference being that we are now analyzing different versions of the same project.

Results

We randomly selected 10 open-source projects and conducted dependency statistics across all of their released versions, as depicted in Figure 4.6. From the graph, we can observe the overlap of dependency packages between different versions of the same project. It's evident that the average hit rate of these projects is considerably higher than the data shown in Figure 4.4. In fact, for some projects, the average value is close to 1. From Figure 4.6, it's evident that substantial overlap exists in dependencies between successive versions and their previous counterparts. In fact, the average overlap rate for certain projects between

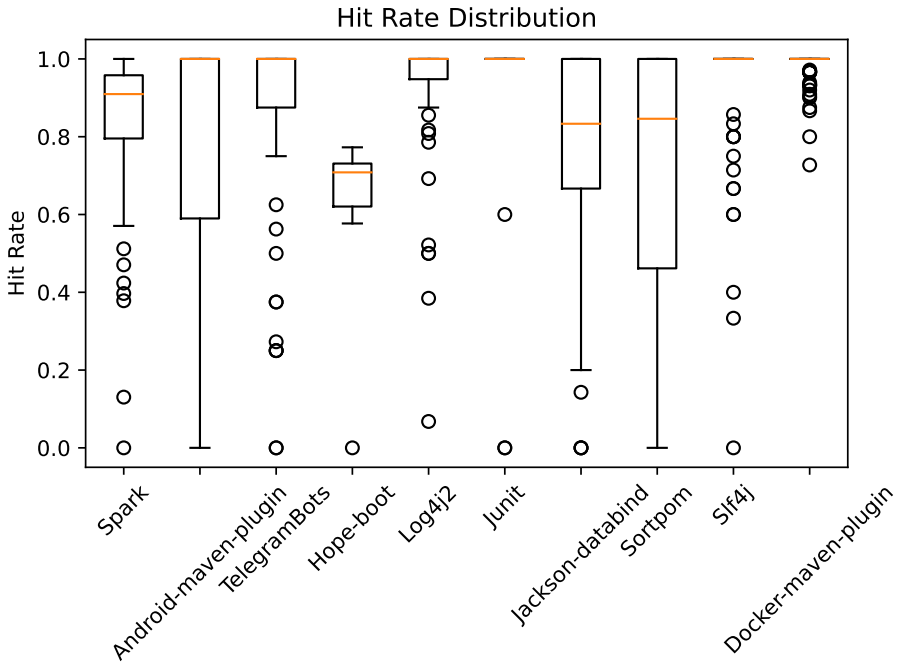

Figure 4.6: Box plot of Vertical Analysis

different versions reaches 100%. This trend enables users to substantially cut down on computation time by reutilizing the previously computed results.

4.2 What is the performance gain for the pre-computation task at different hit rates?

We know that in actual use, whenever a user needs an answer to a pre-computed task, they will first send a request for the specified task to the server. If the server has the answer to this task, it will be returned directly to the user. However, sometimes, due to changes in the user's project dependencies or the choice of the consensus approach, there may not be an available answer temporarily. In this case, the user needs to compute the task themselves and return the result. We define the hit rate as the ratio of the number of answers available in the server to the total number of requests for a project.

$$HitRate = \frac{num_{retrieved}}{num_{requests}} \tag{4.1}$$

Obviously, the higher the hit rate, the smaller the amount of local computation the user needs to complete, which helps improve development efficiency. When the crowd-sourcing

system's database is saturated, the database in the server is already near completeness, which should be the time when the hit rate is the highest.

The establishment of the database is a process, and during this process, the hit rate will affect the user experience. Therefore, in this part, we want to study the improvement of project analysis speed by the crowd-sourcing platform under different hit rate conditions.

4.2.1 Methodology

For users, the time taken to fetch from the server using the plugin is considered the *Response Time* ($t_{response}$) of the REST API, while the time taken for local computation is considered as the *Computation Time* ($t_{computation}$).

The *Response Time* is determined by several factors including the time $t_{request}$ for the client to send a request to the server and receive a response, the time $t_{download}$ to download the analysis results from the server, the time $t_{generate}$ for local analysis of the dependency packages, and the time t_{upload} for uploading the analysis results. We assume a project has a total of n dependencies, among which m dependencies don't have available results on the server and require local analysis by the user, while the remaining $n-m$ dependencies' analysis results can be directly retrieved from the server. For the crowd-sourcing platform to be effective, these factors need to meet the following conditions:

$$t_{response} = n \times t_{request} + m \times (t_{generate} + t_{upload}) + (n-m)t_{download} \tag{4.2}$$

$$t_{computation} = n \times t_{generate} \tag{4.3}$$

$$t_{response} < t_{computation} \tag{4.4}$$

This implies that a computationally intensive task is required. Therefore, in this experiment, we have chosen to use the call graph as the task.

We plan to study the differences in analysis time for projects with hit rates of 0%, 50%, and 100%. Since the size of each pre-computed task may vary, we will analyze and compare multiple projects to obtain a more comprehensive understanding of the relationship between hit rates and analysis time.

In the experiment, we will manually set the answers stored in the server according to the hit rates. We start with a project where no pre-computed results are available on the server. We run the plugin in this scenario, where it won't obtain any useful information from the server and will need to compute all call graphs locally. We record the overall execution time of this process as the baseline.

Next, we retrieve the dependency list from the project's configuration file and randomly select 50% of the dependencies to store their call graphs on the server. We then run the plugin again on the project, which should reduce the local computation by 50%, but may introduce additional time due to network transfer. Again, we record the overall execution time.

Finally, we store the call graphs of all dependencies on the server and run the plugin on the project once more, recording the time. This way, we obtain the time required to obtain call graphs using the plugin under three different hit rate scenarios. We repeat this process for 5 different projects.

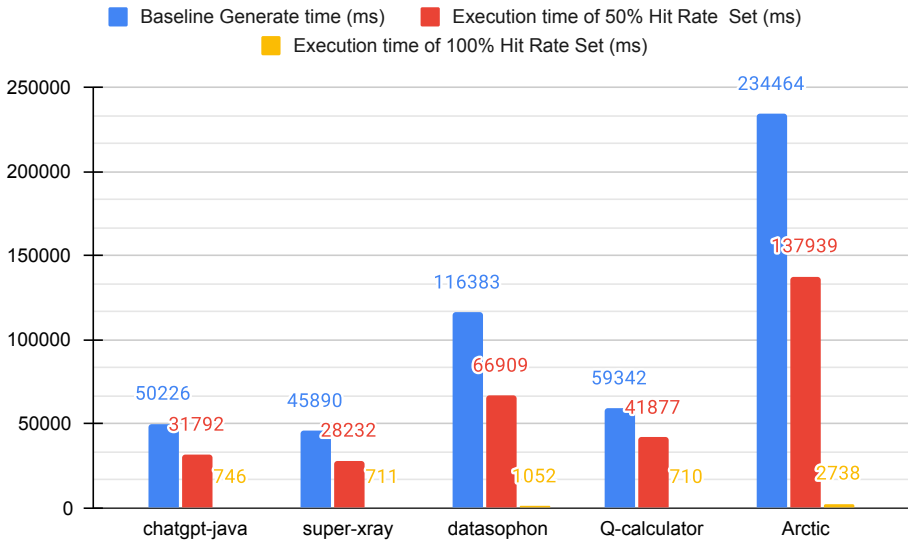

Figure 4.7: Performance at Different Hit Rates

Results

We have obtained results as depicted in Figure 4.7. In the graph, bars of three different colors represent the baseline generate time, execution time at a 50% hit rate, and execution time at a 100% hit rate, respectively. As illustrated in Figure 4.7, with the increase in hit rate, the required execution time significantly decreases. This implies that users will experience noteworthy performance improvements by employing this crowd-sourcing platform. Due to variations in the number of dependencies used in different projects, the time required for generating call graphs varies among them. Additionally, different dependencies may have different call graph generation times, which introduces some variations in the results. However, overall, with a hit rate of 50%, the average reduction in analysis time is 37.7%. This means that for a project that has not been analyzed using this crowd-sourcing platform, using the platform for software analysis only requires about 60% of the time it would take for local analysis. For projects that have been analyzed multiple times using this platform, up to 99% of computation time can be saved.

4.3 What is the trade-off of computation load for the server?

The objective is to compare the time required for completing computation tasks individually and the time taken to compare multiple results through voting. A practical concern arises when the time required for the server to compute a task's result is shorter than the time taken to compare and vote on the results submitted by users. In such cases, there may be no need for consensus mechanisms. Therefore, through this experiment, we aim to find a balance

between the server's computation time and the time required to compare user-submitted results. This will help us determine a reasonable voting threshold.

4.3.1 Methodology

We define the time it takes for the server to generate an analysis result for a dependency package as $t_{generate}$. It consists of two parts: the time $t_{computation}$ for the server to perform the pre-computation task and the time $t_{serialization_{compute}}$ to store the computed result into a file. Additionally, the time it takes to select the final result using the Consensus Mechanism is denoted as $t_{consensus}$. This involves reading and writing files to load the candidate results into memory, noted as $t_{consensus}$ and $t_{serialization_{compare}}$ respectively, and the time to compare these results, denoted as $t_{comparison}$. The objective is to ensure that $t_{consensus}$ is less than $t_{generate}$.

In mathematical terms:

$$t_{generate} = t_{computation} + t_{serialization_{compute}} \tag{4.5}$$

$$t_{consensus} = t_{deserialization} + t_{comparison} + t_{serialization_{compare}} \tag{4.6}$$

$$t_{generate} \geq t_{consensus} \tag{4.7}$$

From this analysis, we can observe that the execution time is primarily influenced by the computation intensity and serialization intensity. Larger files not only present challenges in serialization but also increase the comparison time. Consequently, we can categorize tasks into four types based on their computation intensity and serialization intensity:

1. Low Computation Intensity and Low Serialization Intensity: Tasks in this category involve simple computations and small files that can be easily serialized. These tasks generally have shorter execution times due to the low computational and serialization complexity.

2. Low Computation Intensity and High Serialization Intensity: Tasks in this category have simple computations but involve large files that are challenging to serialize. The execution time is primarily influenced by the serialization process.

3. High Computation Intensity and Low Serialization Intensity: Tasks in this category involve complex computations but have relatively small files that can be easily serialized. The execution time is primarily determined by the computational complexity.

4. High Computation Intensity and High Serialization Intensity: Tasks in this category involve complex computations and large files that are difficult to serialize. These tasks require significant processing power and may result in longer execution times.

For the first two types of tasks, "Low Computation Intensity and Low Serialization Intensity" and "Low Computation Intensity and High Serialization Intensity," the voting

algorithm may not provide any significant advantage due to their inherently small computational requirements. The server can quickly compute reliable answers and return them to users. Users may even perform the calculations locally without the need to use our system.

However, for tasks with "High Computation Intensity," our consensus platform becomes valuable. It saves computational time and reduces the demand for computer resources by avoiding redundant computations. In this experiment, we choose to compute the call graph of an artifact as an example of a task with high computational intensity. Our goal is to find a balance between computation and comparison time.

The experimental setup is as follows:

1. Select the 20 packages from the Maven repository and generate their call graphs. Record the execution time.

2. Create multiple replicas of results and measure the time required for comparing results among different replicas. The comparison process continues until the time exceeds the average time calculated in the first step. Record the number of replicas involved at that point and calculate the average number of replicas required for each task intensity.

Through this experiment, we aim to determine the threshold for selecting the number of participants in the voting process based on the task intensity. Below this threshold, the time required for comparison is lower, while above this threshold, the server's individual completion time is lower. To strike a balance between answer accuracy and server load, the actual setup should not exceed this threshold.

By analyzing the trade-off between computation load and task intensity, we can optimize the crowd-sourced collection and analysis of software packages to achieve efficient and accurate results while minimizing the burden on the server.

Results

We have obtained results as shown in Figure 4.8. From the graph, it's evident that the baseline $t_{generate}$ (represented by the blue line) is significantly higher than the $t_{comparison}$ time obtained by using the consensus mechanism from varying numbers of results (represented by the other three lines). In fact, the difference is so pronounced that we had to use a logarithmic scale to visualize the data, otherwise, the three lines' shorter durations would be barely visible. Based on the experimental results shown in Figure 4.8, it is clear that the time required for generating the call graph is significantly higher than the time spent on comparing different analysis results. Except for the Clojure package, the comparison time for other packages is almost negligible. This can be attributed to the larger size of the call graph in the Clojure package compared to the others. However, even in this case, the comparison time is still shorter.

In conclusion, the results demonstrate that the voting approach is more efficient in terms of time consumption when compared to the generation approach. This indicates that leveraging the consensus mechanisms and integration methods in the platform can effectively reduce the computational load on the server, allowing for faster and more efficient analysis of software packages.

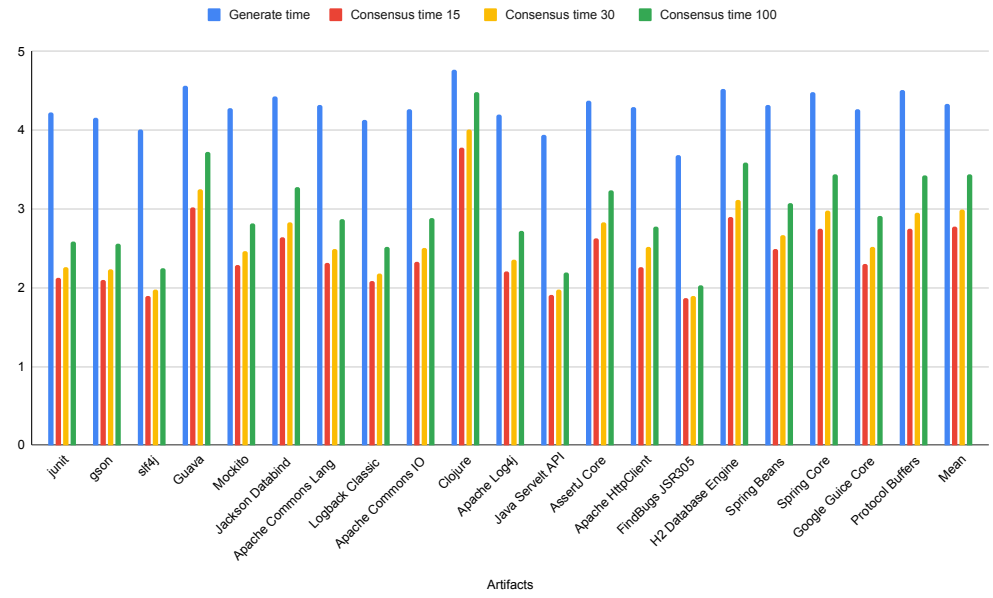

Figure 4.8: Comparison of Call Graph Generation Time and Consensus Time (Log Scale)

Chapter 5

Discussion

In this thesis, we have designed a decentralized software analysis platform with the aim of alleviating the workload of the central server. Leveraging the concept of crowd-sourcing, we have distributed the analysis tasks of each software package to individual users. In this chapter, we provide a reflection on our work. We first discuss the implications of our findings and highlight the potential areas for future improvements. Additionally, we examine the threats to the validity of our research.

5.1 Discussion and Future Works

Consensus Strategies In the crowd-sourcing platform, we offer two different consensus integration methods, namely In Process and Challenge and Response, which represent two different communication protocols. The main difference lies in whether the server proactively requests the results of a pre-computation task from the clients. The advantage of In Process integration is its faster saturation speed, while Challenge and Response integration offers higher security. Once the platform collects the results uploaded by users, the next challenge is to select reliable answers from them. Our platform provides three consensus mechanisms: FCFS Consensus, Majority Voting, and Trust-based Voting. If the FCFS approach is chosen, the answer repository of the system will grow rapidly, but it also compromises security as the first submitter holds significant weight. To enhance the system's reliability, we offer two voting-based consensus mechanisms.

As discussed in Section 3.5, the combination of these two consensus integration methods and three consensus mechanisms can form six different strategies to suit various usage scenarios. During usage, platform administrators can freely combine consensus integration methods and consensus mechanisms based on their specific needs, achieving a complementary balance between their advantages and disadvantages.

Unlike traditional crowd-sourcing platforms with subjective tasks, our platform only requires users to execute specified software analysis code and upload the results. The absence of subjective questions allows us to overlook the expertise of different users and focus on analyzing the presence of malicious users. When the user group is small and trustworthy, such as within a company or research group, choosing strategies based on FCFS can utilize

the high trustworthiness of users and alleviate the slow growth of analysis results in the database due to the small number of users. For platforms targeting the general public, we recommend the combination of In Process integration approach and Voting mechanisms. The difference between the two Voting mechanisms lies in whether all users are treated equally. Trust-based Voting assigns a separate weight value to each user, which may increase the time required to achieve consensus but further enhances the credibility of the results, especially when the voting threshold is set low. Platform developers can choose between these two voting strategies based on the specific performance requirements of the platform.

Considering practical feasibility, we do not recommend the combination of the Challenge and Response integration approach and Voting mechanisms. From the saturation experiment, we observed that even after analyzing over two thousand projects, the number of unique dependencies continues to grow. This implies that whenever a new project is analyzed, although most analysis results can be directly obtained from the server, new results to be analyzed will inevitably be introduced. Currently, we only require clients to complete a specified analysis task each time they access the service. This is not a significant issue when using the FCFS consensus mechanism, but when using Voting mechanisms, to achieve consensus on a pre-computation task, multiple analysis results need to be collected, which may lead to the number of tasks required to reach consensus growing slower than the number of new tasks to be solved, resulting in an ever-increasing pending list that can never be completed. Increasing the number of analysis tasks assigned to clients each time they access the service comes at the cost of extending the clients' execution time because of the additional tasks. This requires platform designers to find a balance between clients' extra execution time and the speed of achieving consensus, which poses an interesting research problem.

While the establishment of our crowd-sourcing platform allows users to share a variety of software package analysis results, reducing redundant work and saving development time, there is still room for further advancements to make the platform more intelligent and effective.

Decentralized Storage One major improvement we have achieved is the decentralization of computation tasks, effectively alleviating the burden on the central server. By distributing the computation tasks to users' computers, the server's role is reduced to result collection and comparison. However, while this achieves decentralization in terms of computation, the platform still relies on a central server for result collection and comparison. Future work can focus on exploring how to coordinate multiple servers to create a fully distributed system, where result collection and comparison are also distributed across different machines. This would involve developing mechanisms for data exchange and result synchronization among the distributed servers.

Dynamic Task Allocation Another aspect to consider is that the platform currently supports different task types, but these are hard-coded, requiring both the client and server to have prior knowledge of the specific tasks and how to execute them. This lack of flexibility

results in poor compatibility when expanding the range of task types. To address this issue and enable smarter load distribution, it would be beneficial to explore a system where task instructions are distributed directly by the server. This would allow for more dynamic and adaptable task allocation, reducing the need for frequent updates to the Maven plugin on the client side.

Platform start-up As a platform that employs the concept of crowd-sourcing, it's evident that with an increasing number of users joining, more and more analysis results are contributed. Consequently, the users of the platform ultimately benefit from having access to a wider array of computed results. However, this characteristic also implies that the platform may not perform optimally in its initial stages, as there would be minimal results available in the database. Therefore, we seek to determine the point at which the platform becomes reasonably stable and functional after analyzing a certain number of projects. First, we analyzed the dependency overlap in different open-source projects to explore how many projects the system needs to analyze to achieve a relatively stable hit rate, known as saturation. We found that after analyzing around 1000 projects, the hit rate tends to stabilize. While some variations exist among different projects, a new project can expect a hit rate of around 50%. This means that a new user can reduce their computation workload by half upon their first usage, which is highly appealing. In practical usage, developers typically analyze projects multiple times, especially with each new version release. Therefore, we also examined the dependency overlap between different versions of the same project. The results showed that the vast majority of dependencies remain consistent across versions, indicating that long-term usage of this crowd-sourcing platform can save significant computation time, leading to improved user retention.

Platform effectiveness In addition, we also evaluated the system from the perspective of platform users. We believe that as users of the platform, clients are not concerned about the server's workload or the process of obtaining analysis results. Their primary goal is to obtain the analysis results for specific dependencies as quickly as possible. Therefore, we conducted an experiment to test the performance improvement achieved by using the crowd-sourcing platform. With three different hit rates, the experiment demonstrated that users can save a significant amount of computation time by using our Maven plugin. The focus of the experiments was on a computationally intensive task, call graph generation, and it was found that the time required for transmitting the results can be negligible compared to the time required for performing the actual call graph computation. In other words, users can expect nearly linear performance improvement, indicating that the more local computation time is saved, the more overall cost is reduced. Exploring and testing different levels of software analysis tasks in terms of their performance on the platform would be an interesting avenue for future work. By varying the complexity and intensity of the tasks, we can better understand the platform's efficiency and effectiveness in handling different types of analyses. This research can help identify the optimal use cases and potential limitations of the crowd-sourcing platform, providing valuable insights for platform developers and users.

Finally, our experiments have shown that comparing different analysis results submitted by users places lower demands on performance compared to generating the call graph on the server. The most significant difference between our crowd-sourcing platform and FASTEN is that our server does not perform package analysis itself; it only serves as a collection and integration platform for analysis results. During the process of obtaining consensus through voting mechanisms, there is a step involving the deserialization, comparison, and serialization of all analysis results. We aimed to find a balance between the time required for comparing a limited number of results, where voting mechanisms perform better, and the time required for the server to analyze dependencies, where it might perform better when dealing with a larger number of results. Thus, we selected the time required for the server-generated call graph as the baseline and compared it with the time for choosing the best answer from different result quantities. Surprisingly, in our experiments, even when comparing 100 results, it only took roughly one-tenth of the baseline time. Considering that repeating a single package more than 100 times in practice is almost impossible, setting the voting threshold to 100 is excessive and not reasonable. Our evaluation demonstrated that for computationally intensive tasks, using a voting-based consensus mechanism significantly reduces the computational overhead compared to generating the results on the server.

5.2 Threats to Validity

While the experiments described in this thesis yield promising results, demonstrating that using a crowd-sourcing platform for software analysis can alleviate server workload and enhance user experience, there are some limitations that may affect the experimental outcomes and real-world scenarios.

Firstly, our platform is currently designed for limited use cases, specifically Java and Maven ecosystems. The collected dataset consists of projects developed using Maven, and the analysis results are tailored to this context. Therefore, when extending the crowd-sourcing platform to other ecosystems, the saturation rate of the system database and client-side performance improvements may vary. However, we still believe that introducing a crowd-sourcing platform would yield positive results.

In the experiments evaluating system database saturation rates, we analyzed the overlap of dependencies in different projects on GitHub. The conclusions drawn were based on the FCFS consensus mechanism. However, if voting-based consensus mechanisms are chosen in real-world usage, the saturation rate may significantly decrease. Nonetheless, this does not necessarily mean that users' experiences will be significantly worse than expected. In the development process, developers often analyze their projects multiple times, falling into our vertically analyzed region, meaning they can save analysis time by reusing their submitted results.

In our experiments, we chose to use call graph generation as the workload, which is an effective method for providing fine-grained software analysis. However, as a crowd-sourcing platform aimed at different types of tasks, the actual workload in real-world usage may vary, leading to potential deviations from the experimental results. We believe that

computationally intensive tasks are more likely to benefit from using this platform. For tasks that only require milliseconds or seconds of analysis, the benefits may be minimal or even negative due to the presence of network latency.

During the experiments, we used a single computer (my PC) for analysis. However, in practical usage, different users have different computer performances, which could lead to variations in user experience. Clearly, the performance gains will be more significant on computers with lower performance, as they can avoid local analysis. Additionally, users' network conditions are also an important variable. While analyzing the performance improvement in RQ3, we did not consider the impact of network speed on users. Nonetheless, since the size of the call graph files used in the experiments was relatively small compared to the generation time, we believe that the effect of network speed on the overall results is negligible. Therefore, users can expect substantial performance improvements using the crowd-sourcing platform, regardless of network speed, unless the network conditions are exceptionally poor.

Chapter 6

Summary

Software analysis is a fundamental aspect of software development and maintenance, involving a systematic examination of code to identify errors, assess quality, and enhance performance. It empowers developers with insights into software's underlying structure and behavior, aiding in the detection of vulnerabilities and inefficiencies. However, software analysis encounters challenges due to the intricacies of modern software systems, which often result in time-consuming analysis due to intricate dependencies.

In this study, we integrate the concept of crowd-sourcing into the design of a software package collection and analysis platform. Our objective is to distribute analysis tasks, originally conducted solely on servers, among diverse users to alleviate server load. We create a crowd-sourcing Software Collection and Analysis system consisting of a Maven plugin for users and a back-end server. We define two consensus integration methods and three consensus mechanisms to adapt to diverse application scenarios.

To validate the efficacy of this crowd-sourcing approach, we perform an array of evaluations. First, we analyze dependency overlaps among open-source Maven projects on GitHub over the past three years. The results reveal significant dependency overlaps among distinct projects, suggesting that newcomers can reuse previously uploaded analysis results. Even projects analyzed for the first time on this platform can save approximately half of the local analysis task time. Moreover, we analyze the performance impact of introducing this crowd-sourcing platform from the perspectives of both platform users and providers. Evaluating client-side plugin execution times at different hit rates demonstrates that the platform expedites users' access to analysis results. Additionally, experiments involving server-side call graph generation and selection of answers by comparing call graphs with a specific count indicate that employing Voting mechanisms effectively reduces the server's workload.

Our research underscores that constructing a software analysis platform via crowd-sourcing is advantageous for both platform providers and users.

Bibliography

[1] Fasten. https://www.fasten-project.eu/view/Main/, 2020.

[2] Pietro Abate, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli. A modular package manager architecture. *Information and Software Technology*, 55(2):459–474, 2013.

[3] Shahzad Sarwar Bhatti, Xiaofeng Gao, and Guihai Chen. General framework, opportunities and challenges for crowdsourcing techniques: A comprehensive survey. *Journal of Systems and Software*, 167:110611, 2020.

[4] Paolo Boldi and Georgios Gousios. Fine-grained network analysis for modern software ecosystems. *ACM Trans. Internet Technol.*, 21(1), dec 2020. ISSN 1533-5399. doi: 10.1145/3418209. URL https://doi.org/10.1145/3418209.

[5] Silvana Castano, Alfio Ferrara, Lorenzo Genta, and Stefano Montanelli. Combining crowd consensus and user trustworthiness for managing collective tasks. *Future Generation Computer Systems*, 54:378–388, 2016. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2015.04.014. URL https://www.sciencedirect.com/science/article/pii/S0167739X15001065.

[6] Seth Cooper, Firas Khatib, Ilya Makedon, Hao Lu, Janos Barbero, David Baker, James Fogarty, Zoran Popović, and Foldit Players. Analysis of social gameplay macros in the foldit cookbook. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 9–14, 2011.

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, page 1277–1286, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310864. doi: 10.1145/2145204.2145396. URL https://doi-org.tudelft.idm.oclc.org/10.1145/2145204.2145396.

[8] Alexandre Decan, Tom Mens, and Maëlick Claes. An empirical comparison of dependency issues in oss packaging ecosystems. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 2–12, 2017. doi: 10.1109/SANER.2017.7884604.

[9] Jens Dietrich, David Pearce, Jacob Stringer, Amjed Tahir, and Kelly Blincoe. Dependency versioning in the wild. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 349–359. IEEE, 2019.

[10] Fernando Rodriguez Oliver. Mvnrepository. `https://mvnrepository.com/repos/central`, 2023.

[11] Matthew Finifter. Exploring the relationship between web application development tools and security. In *2nd USENIX Conference on Web Application Development (WebApps 11)*, 2011.

[12] Ujwal Gadiraju, Ricardo Kawase, Stefan Dietze, and Gianluca Demartini. Understanding malicious behavior in crowdsourcing platforms: The case of online surveys. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 1631–1640, 2015.

[13] David Geiger, Michael Rosemann, and Erwin Fielt. Crowdsourcing information systems–a systems theory perspective. 2011.

[14] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438:900–1, 01 2006. doi: 10.1038/438900a.

[15] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing*, 7(4):12–18, October 2008. ISSN 1536-1268. doi: 10.1109/MPRV.2008.80. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4653466&tag=1`.

[16] Michael Hanus. Semantic versioning checking in a declarative package manager. In *Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[17] Christopher Harris. You're hired! an examination of crowdsourcing incentive models in human resource tasks. In *Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 15–18. Hong Kong, China, 2011.

[18] Danula Hettiachchi, Vassilis Kostakos, and Jorge Goncalves. A survey on task assignment in crowdsourcing. *ACM Comput. Surv.*, 55(3), feb 2022. ISSN 0360-0300. doi: 10.1145/3494522. URL `https://doi-org.tudelft.idm.oclc.org/10.1145/3494522`.

[19] Mokter Hossain and Ilkka Kauranen. Crowdsourcing: a comprehensive literature review. *Strategic Outsourcing: An International Journal*, 8(1):2–22, 2015.

[20] Mahmood Hosseini, Alimohammad Shahri, Keith Phalp, Jacqui Taylor, and Raian Ali. Crowdsourcing: A taxonomy and systematic mapping study. *Computer Science Review*, 17:43–69, 2015.

[21] Jeff Howe. The rise of crowdsourcing. *Wired Mag*, 14, 01 2006.

[22] James Howison, Keisuke Inoue, and Kevin Crowston. Social dynamics of free and open source team communications. In Ernesto Damiani, Brian Fitzgerald, Walt Scacchi, Marco Scotto, and Giancarlo Succi, editors, *Open Source Systems*, pages 319–330, Boston, MA, 2006. Springer US. ISBN 978-0-387-34226-9.

[23] Humayun Irshad, Laleh Montaser-Kouhsari, Gail Waltz, Octavian Bucur, JA Nowak, Fei Dong, Nicholas W Knoblauch, and Andrew H Beck. Crowdsourcing image annotation for nucleus detection and segmentation in computational pathology: evaluating experts, automated methods, and the crowd. In *Pacific symposium on biocomputing Co-chairs*, pages 294–305. World Scientific, 2014.

[24] Mehdi Keshani, Simcha Vos, and Sebastian Proksch. On the relation of method popularity to breaking changes in the maven ecosystem. *Journal of Systems and Software*, 203:111738, 2023.

[25] Sung-Hee Kim, Sensen Li, Bum chul Kwon, and Ji Soo Yi. Investigating the efficacy of crowdsourcing on evaluating visual decision supporting system. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 55, pages 1090–1094. SAGE Publications Sage CA: Los Angeles, CA, 2011.

[26] Chris Lintott, Kevin Schawinski, Steven Bamford, Anže Slosar, Kate Land, Daniel Thomas, Edd Edmondson, Karen Masters, Robert C. Nichol, M. Jordan Raddick, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg. Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies*. *Monthly Notices of the Royal Astronomical Society*, 410(1):166–178, 12 2010. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.17432.x. URL https://doi.org/10.1111/j.1365-2966.2010.17432.x.

[27] Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems – a systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306, 2013. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2012.12.026. URL https://www.sciencedirect.com/science/article/pii/S016412121200338X.

[28] Fábio R Assis Neto and Celso AS Santos. Understanding crowdsourcing projects: A systematic review of tendencies, workflow, and quality management. *Information Processing & Management*, 54(4):490–506, 2018.

[29] David Oleson, Alexander Sorokin, Greg Laughlin, Vaughn Hester, John Le, and Lukas Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *Workshops at the Twenty-Fifth AAAI conference on artificial intelligence*. Citeseer, 2011.

[30] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

[31] M Six Silberman, Lilly Irani, and Joel Ross. Ethics and tactics of professional crowdwork. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):39–43, 2010.

[32] Brian L Sullivan, Christopher L Wood, Marshall J Iliff, Rick E Bonney, Daniel Fink, and Steve Kelling. ebird: A citizen-based bird observation network in the biological sciences. *Biological conservation*, 142(10):2282–2292, 2009.

[33] T. Preston-Werner. Semantic versioning 2.0.0. `https://semver.org/`, 2018.

[34] Luis Von Ahn and Laura Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[35] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. A survey of crowdsourcing systems. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 766–773, 2011. doi: 10.1109/PASSAT/SocialCom.2011.203.

[36] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: Is the problem solved? *Proc. VLDB Endow.*, 10(5): 541–552, jan 2017. ISSN 2150-8097. doi: 10.14778/3055540.3055547. URL `https://doi-org.tudelft.idm.oclc.org/10.14778/3055540.3055547`.

[37] Silas Ørting, Andrew Doyle, Arno van Hilten, Matthias Hirth, Oana Inel, Christopher Madan, Panagiotis Mavridis, Helen Spiers, and Veronika Cheplygina. A survey of crowdsourcing in medical image analysis. *Human Computation*, 7:1–26, 12 2020. doi: 10.15346/hc.v7i1.1.