Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Rewarding reviews with tokens: An Ethereum-based approach

Andrea Lisi [a,b,*], Andrea De Salve [c], Paolo Mori [b], Laura Ricci [a], Samuel Fabrizi [a]

[a] University of Pisa, Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy
[b] Consiglio Nazionale delle Ricerche - IIT, via G. Moruzzi, 1, 56124 Pisa, Italy
[c] Consiglio Nazionale delle Ricerche - ISASI, Campus Universitario Ecotekne, 73100 Lecce, Italy

ARTICLE INFO

Article history:
Received 22 January 2020
Received in revised form 5 January 2021
Accepted 6 February 2021
Available online 15 February 2021

Keywords:
Distributed Ledger Technology
Recommender System
Blockchain
Smart contract
Rewarding system

ABSTRACT

Recommender Systems (RSs) are becoming increasingly popular in the last years. They collect reviews concerning several types of items (e.g., shops, professionals, services, songs or videos) in order to rank them according to a given criterion, and to suggest the most relevant ones to their users. However, most of the currently used RSs exhibit two main drawbacks: they are based on a centralized control model and they do not provide reward mechanisms to encourage the participation of users. To deal with these challenges, the architectures of current RSs could be enhanced through blockchain technology, thus providing novel solutions to decentralize them. As a matter of fact, the blockchain technology could be successfully adopted in this context because smart contracts would allow the decentralization of system control, while cryptocurrency and tokens could be used to implement the reward mechanism.

In the light of the above considerations, this manuscript presents a decentralized rating framework aimed to support the users of RSs based on blockchain technology, providing a token-based reward mechanism that remunerates users submitting their reviews to incentivize their participation. Moreover, the proposed system provides a flexible strategy to rank items, allowing users to choose among different functions to combine reviews to obtain item ranking. The performance and the cost of using the proposed system have been evaluated on the Ropsten Ethereum test network. For instance, our experiments have shown that the median time required to store a batch of 35 ratings is about 47 s, while the average time required to obtain the score of an item having 6000 ratings is less than 2.5 s.

1. Introduction

Nowadays, information and communication technologies allow users to share any kind of content quickly and easily. As a result, the amount of information available on the World Wide Web is huge and continuously growing, and this makes it difficult for users to find the most relevant and reliable data for them. Reviews concerning the most disparate types of items, both real and virtual (e.g., shops, professionals, services, songs or videos), are among the contents that are commonly shared on the web,[1] which result of limited use if they are not properly organized. To alleviate the issue, Recommender Systems (RSs) have been proposed [1]. Essentially, a RS collects from its users the reviews (consisting of ratings and opinions) about a set of typically homogeneous items (e.g., restaurants), and exploits such reviews to compute a score for each item, ranking them by order of liking, and to forecast the most favored items to users according to the ratings, the preferences expressed, and the actions they performed.

For instance, reading or writing reviews on TripAdvisor,[2] currently one of the most popular RSs, has become a regular activity among Internet users and, based on TripAdvisor Global Report [2], about 49% of travelers were inspired to visit a new destination by a personalized recommendation. Besides products, restaurants, and hotels, RSs are also used to rate and recommend other types of items such as movies [3], professionals and companies (LinkedIn, Glassdoor), healthcare providers [4], videos uploaded on YouTube [5], and songs on Spotify [6].

The collection of reviews about items is one of the most important steps for a RS, and it is typically carried out by allowing each user to express a rating, which is a numerical vote (e.g., a number of stars in the range [0, 5]), and/or an opinion, which is a textual evaluation of the item based on the user experience. The cooperation and altruism of individuals are vital for the above

* Corresponding author at: University of Pisa, Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy.
E-mail addresses: andrea.lisi@phd.unipi.it (A. Lisi), andrea.desalve@cnr.it (A. De Salve), paolo.mori@iit.cnr.it (P. Mori), laura.ricci@unipi.it (L. Ricci), s.fabrizi1@studenti.unipi.it (S. Fabrizi).

[1] Statista 2019: https://bit.ly/2BYJf1U [Accessed 15-January-2020].

[2] TripAdvisor: https://www.tripadvisor.com.

https://doi.org/10.1016/j.future.2021.02.003

systems to provide an effective service because they rely on the reviews submitted by their users. The cooperation of individuals can be *community based* or *incentive based* [7]: in the former, the cooperation is socially driven, where people accomplish a task as a group; when the social component is missing the latter comes in, where an incentive is necessary to push people's motivation. The incentive can have the form of *reputation*, i.e., improves the status of an individual, *reciprocity*, i.e., improves mutual trust, and *monetization*, i.e., an economic reward.

Since the presence of rewards is a necessary condition to generate a willingness to people to cooperate [8], modern digital systems often utilize a reward mechanism, such as the metaphor of likes (social networks), collaborations (between YouTube authors), and virtual currency (games and Bitcoin). However, in several cases (such as fast-food [9]), a large fraction of users prefers monetary rewards (discounts or cashback) to non-monetary [10].

Most of the currently available RSs are based on a centralized control model, i.e., they are controlled by a single service provider who manages the collection and the storage of the reviews submitted by their users, as well as the computation of the scores of the items based on such reviews. Such scores have an economic value, because they are used to build ranked lists of items meant to influence the future purchasing decisions of the user requesting them [11]. As a matter of fact, most of the users which ask a RS for the ranked list of restaurants nearby their positions will probably have their meals in one of the best ranked restaurants on that list. Hence, having a high score, thus being among the first items of the ranked lists returned by a RS, would lead to an increase in the number of customers for the corresponding item. One of the main issues of current centralized control RSs is that they do not provide to their users any evidence of the reviews that have been taken into account for the computation of the score of an item and of the algorithm that has been used to compute such score. Because of this lack of transparency, it is difficult for a user to prevent the provider of a centralized RS from manipulating the score of the items, even if the service provider claims to be honest or exposes the algorithms used to compute the score of the items [12,13]. Consequently, users of RSs are forced to trust the service providers, since they have no means to verify the item score calculation process. In order to avoid control of the service provider over data, decentralized RSs [14] have been proposed and they aim to avoid unnecessary centralized entities.

This paper proposes an alternative approach, based on the Distributed Ledger Technology [15], to address the previous issue by building a decentralized rating framework aimed to support the users of RSs with a transparent review collection and item score calculation processes. In particular, the proposed framework is built on top of a platform that runs smart contracts on a public blockchain, without any authority controlling it, and supporting a decentralized collection of reviews and ranking of items. Since it is based on a public blockchain that provides an immutable, public, and ordered ledger, it prevents censorship, downtime, and alteration of the submitted reviews in a second moment. The framework implements a straightforward authorization system to regulate the recording of the reviews submitted by its users. A preliminary version of the proposed approach has been presented in [16]. Since user participation is crucial for RSs to provide an effective service, in this paper we have extended the framework capabilities in this direction by integrating a token economy based on the notion of user experience. In particular, we introduced a reputation mechanism meant to compute the experience of users when reviewing items, and a reward mechanism that grants to users reviewing items a reward proportional to their current experience. The reward is modeled by an internal virtual currency exploiting ERC20 compliant tokens. To prove the feasibility of the proposed approach, we developed and deployed a prototype of such improved decentralized rating framework on an Ethereum test network, and we conducted an extensive set of experimental evaluations to measure its performance.

The resulting framework promises a higher level of participation and it applies to any retail and e-business infrastructure, with the only changes being the integration of the related data and of the functions for the computation of the score of an item.

Briefly, the main contributions of this paper can be summarized as follows:

- Starting from the results of our previous work [16], we define a general decentralized rating framework to model the backbone of a recommender system built on top of a public blockchain;
- We enrich the framework by adding a novel local reward strategy based on both reputation and monetization. Users are incentivized to submit their reviews because they are rewarded with both reputation increases and tokens, where such tokens can be subsequently spent for benefits. The item owners decide, based on their strategies, the value of their tokens. Moreover, a cryptocurrency payment method has also been added to the framework, supporting token-based discounts;
- We provide a prototype of the proposed framework on the Ethereum ecosystem and conducted an extensive evaluation of its performance on the Ropsten testnet;
- We perform a detailed comparison against both our previous version of the framework and other state-of-arts approaches, discussing their strengths and weaknesses.

The rest of the manuscript is organized as follows: Section 2 introduces the reader to fundamental concepts related to Blockchain, Section 3 provides the model and the architecture of our framework, while in Section 4 we provide the reader the details of a prototype developed on the Ethereum platform and an evaluation of the performance of that prototype. Section 5 presents the related works and compares the proposed approach with them. Instead, Section 6 focuses on critical analysis of the proposed framework and presents the threat model. Finally, in Section 7 we draw the conclusions and outline avenues for future research.

2. Background

In this section we introduce the reader to the fundamental concepts related to Blockchains, Smart Contracts, and Ethereum Tokens.

2.1. Blockchain and smart contracts

Recently, Distributed Ledger Technology (DLT) has gained increasing attention from both the scientific community and industry due to the important applications and results achieved in different contexts (such as IoT [17] and Health [18]). A blockchain implements a DLT through an append-only list of blocks that contains immutable records. Immutability is achieved by a combination of cryptographic techniques and a P2P consensus protocol, such as Proof of Work (PoW) or Proof of Stake (PoS), which is a collaborative approach that defines who will be elected to update the ledger (e.g. miners or validators) by appending the new block and will be finally rewarded for its contribution to the transactions validation. Other approaches characterizing permissioned ledgers [19], i.e., ledgers whose validator's set is closed, are known as Byzantine Fault Tolerant (BFT). In this case, a collaborative process establishes the next state of the ledger through a voting procedure between the participants: the proposed next state is valid if a certain quorum of votes is reached.

As a consequence, the blocks of the chain are protected against modification: consensus algorithms like PoW do not require the knowledge of the identity of the validators and reach a probabilistic consensus, i.e., a block may be invalidated by a chain fork that does not contain that block with a probability which decreases over time; consensus algorithms like BFT reach a deterministic consensus, meaning that a new valid state has no risk to change, but they require the knowledge of the identities of the validators.

The underlying peer-to-peer network is used to replicate the blockchain on different peers. The first use case adopting the blockchain technology is the decentralized currency named Bitcoin proposed around 2009. Bitcoin [20] uses the blockchain to mint and transfer a cryptocurrency, the *bitcoin*, in a distributed fashion. A newer approach is adopted by Ethereum [15], a blockchain-based network to build a platform to execute decentralized stateful applications (named smart contracts) written through a Turing-complete programming language (e.g., Solidity[3]) on a virtual machine called Ethereum Virtual Machine (EVM). In particular, a smart contract [21] is a piece of software with the property of running transactions without involving third parties. Bitcoin uses smart contracts as well, coded in Script, a Forth-like language that is intentionally simple and mostly limited to the verification of digital signatures of Bitcoin transactions. Entities in Ethereum are referred to as "accounts", and to any account corresponds an address generated by a private–public key pair. These accounts can be either *Externally Owned Accounts* (EOAs), which are managed through their private keys, or *contract accounts*, which are controlled by code: each account has a balance and can send transactions, which represent a change from a state to another. The cryptocurrency used by Ethereum is named Ether (ETH) and it must also be used to pay the fees for the execution of smart contracts. A smart contract needs to be compiled, and the resulted bytecode is executed by the EVM. Every opcode has associated a cost measured in *units of gas*. The summation of the costs of all the opcodes in a function determines the cost in gas of such function. The caller decides how much ETH to assign to every unit of gas (*gas price*) and how many units of gas to provide for that execution (*gas limit*). Miners are incentivized to execute and validate a transaction with a higher gas price. A smart contract function that changes the state of the network needs to be embedded in a transaction, and therefore stored in a block: in that case, the caller pays a fee in ETH equal to the amount of gas effectively consumed multiplied by the gas price. The gas concept is a measure to prevent abuse of a Turing-complete computation, since endless loops could freeze the entire network. In particular, if the execution of a contract exceeds the gas limit, the contract will be reverted to the original state and all gas spent will not be refunded. In contrast, a state query does not need to be placed in a block and thus does not consume ETH to the caller, even though the gas is computed and the gas limit rule still applies. At the time of writing both Ethereum and Bitcoin use as consensus the PoW algorithm [22].

2.2. Ethereum token standards

In Ethereum, *tokens* are a digital representation of a value associated with an asset, a service, or the right to do something, and they are implemented through smart contracts. The smart contract of a token automatically reacts to specific events and it regulates the exchange of tokens unit. In general, the tokens can be classified into two categories based on their application field: *utility tokens* and *security tokens* (also known as *equity tokens*). The former acts as a key to access some goods through a blockchain and are used to pay the access to a service, application, or resource. Instead, security tokens are similar to equity shares and they are considered financial investments. Put in another way, the key difference is that equity tokens give to the holder of the token ownership rights, while utility tokens act as coupons and do not provide holders with an ownership stake in any asset, like a company's asset and so on.

Given the widespread adoption of the Ethereum tokens in different decentralized applications, several standards (referred to as Ethereum Request for Comment or ERC) have been proposed by the Ethereum official community.[4] Each ERC defines the signature of a set of functions (or interfaces) that the smart contract of the token must provide. In the following we describe such standards and the most relevant proposals of new standards.

ERC20. It is the first standard provided by the Ethereum community to ensure interoperability between tokens. An ERC20-compliant smart contract must implement a set of 6 functions and 2 events. The first two functions, `totalSupply` and `balanceOf` return, respectively, the total amount of available tokens in the contract and the total amount of tokens owned by an account. The function `transfer` allows transferring tokens from the caller of the function to another address. An address can allow another address to transfer its tokens on its behalf, in such a case the `transferFrom` and the `approve` functions are used in combination. In particular, through the function `approve`, the token owner authorizes another user, the receiver, to spend their tokens. The `transferFrom` function, instead, allows the receiver to transfer tokens on behalf of the token owner. Finally, the function `allowance` allows to query the amount of tokens that the receiver can still withdraw from an account that has previously allowed it. The events are issued as a result of the execution of the `transfer`, `approve`, and `transferFrom` operation. A token implementing the ERC20 standard has the property of fungibility because individual units are interchangeable, and each unit of token is indistinguishable from another part.

However, transferring ERC20 tokens should be performed with attention. Sending tokens to an address of a smart contract with the `transfer` operation may lead to a token loss if the target smart contract does not have the code to interact with tokens. Meanwhile, while this operation is safe with EOA, when invoked by a smart contract, it is suggested to use the `approve` + `transferFrom` combination.

ERC223. This standard aims to safely send tokens to smart contracts with a single `transfer` operation discriminating if the target is a smart contract or an EOA. In the former case, a smart contract designed to work with tokens needs to implement a `tokenFallback` function, otherwise the transaction fails.

ERC777 and ERC820. The standard ERC777 improves ERC20 by defining a procedure to check whether the recipient is allowed to receive tokens. To do that, the contract requires to be ERC820-compliant, a standard allowing anyone to check whether a given smart contract defines or not some functions.

ERC721. The ERC721 standard is used to define unique and non-fungible tokens. Given a set of tokens from an ERC721-compliant smart contract, each token represents a unique object of a collection (also a physical good).

[3] Solidity docs: https://solidity.readthedocs.io/en/v0.6.1/index.html. [Accessed 15-January-2020].

[4] Ethereum improvement proposals: https://eips.ethereum.org/erc. [Accessed 15-January-2020].

3. The rating framework for recommender system

To address the issues of current centralized RSs (see the related work in Section 5) and to provide transparency and flexibility to users, we propose a decentralized rating framework that brings together blockchain technology and recommendation systems. In the following of this paper, we focus on the review management process, while we do not consider recommendation strategies because they are meant to be executed on off-chain. In particular, the proposed rating framework supports the submission, the storage, and retrieval of reviews, the modeling and implementation of the internal virtual currency to be used as a reward for the users, and the implementation of a reward mechanism through the notion of skills and reputation.

The integration of a reputation based reward mechanism has proven to be an effective technique to facilitate engagement [23], and it is an enhancement to the version presented in [16]. Indeed, several works demonstrated that services providing a reward or remuneration for users' actions promote engagement and cooperation [23,24]. In particular, the rating framework proposed in this paper consists of a set of smart contracts for the recordkeeping and management of the typical data characterizing rating platforms, implementing a reward system based on user reputation and monetization. Since the proposed framework is implemented through smart contracts on top of a public blockchain, it ensures the following benefits:

- Public: all the data composing the framework (e.g., collected reviews, user reputations, and smart contracts) is visible to all users;
- Decentralized: users do not need to trust the honesty of the entity running the proposed framework, because such framework is not executed by a central authority but, instead, it runs on the blockchain;
- Tamper-proof and Persistent: the data is replicated among all the nodes of the blockchain and its integrity and availability are protected through cryptographic techniques. Hence, the ratings submitted by the users, once registered in the blockchain, cannot be altered or removed by a single entity.

3.1. System model

As shown by Fig. 1, the proposed solution is built on top of a blockchain protocol supporting smart contracts, and it consists of several components that cooperate to implement the system functionalities. In the following, we describe in more details the entities, the concepts, and the operations involved in our rating framework:

Items An item is a virtual representation of a place (such as a restaurant, a hotel, or a store) or of a service (such as a financial instrument or other web services) and it is paired with a set of properties characterizing it, each of these corresponding to a skill that is acquired by users rating items having such property.

Skills After a review, the user unlocks the item property as a personal skill. For instance, the *Chinese cuisine* skill is given to users who have reviewed restaurants having the property *Chinese cuisine*. A skill is a nominal value and the set of skills of a user represents his area of expertise.

Users A user registered to the system is identified by a pseudonym. Each user is allowed to review the used items and, consequently, improves his skills on the properties of such items. For each skill, a user has a reputation representing the experience on that skill obtained from the review mechanism. Moreover, each user holds tokens obtained from the reward mechanism.

Tokens A token is a currency used as an economic reward mechanism for users. Each user gains some tokens when reviews an item, and such tokens can be spent as a discount for a future payment.

Reviews This mechanism allows authorized users to leave a rating and an opinion on a specific item. Only the users who used an item are authorized to rate it (e.g., a restaurant should be rated only by users who have eaten there). After reviewing an item, the user acquires experience on the corresponding skills and receives a certain amount of tokens proportional to such experience.

User Reputation Keep reviewing items with the same property improves the experience of a user on the corresponding skill, and such experience is modeled through the reputation of the user on that skill. The reputation is a numerical value.

Rating Function A rating function computes the overall liking of items. Our approach supports a set of distinct rating functions, and each of these functions processes the reviews stored on the blockchain according to a distinct criteria.

The reputation mechanism is meant to measure the experience of a user in reviewing items. Each time a user reviews an item, he improves his experience on the skills corresponding to the properties of that item and, consequently, the system increments his reputation on such skills to represent the experience accumulated so far by the user with this kind of item. For example, in the case of a restaurant rating framework, the skills could correspond to the cuisine or cooking style, e.g., fusion cuisine, nouvelle cuisine, vegan cuisine, or vegetarian cuisine. A restaurant specialized in vegan cuisine will be paired with the *Vegan Cuisine* property, and its customers will improve their reputation in the *Vegan Cuisine* skill when eating there and rating the restaurant. The growth of the reputation of a user every time a new review is submitted can be computed with the help of several reputation models that have been defined in the literature [25,26]. However, in this paper we do not focus on the definition of a new model, but we observe that the chosen one should prevent a collapse of the system due to the uncontrolled growth of the discounts.

The reward mechanism is based on user reputation and monetization. As a matter of fact, the user who reviews an item receives a reward in tokens proportional to his current reputation on the skills corresponding to the properties of the item. The tokens are used as an internal payment method in the application. We exploited the decision-aid tool proposed in [27] to decide on which token (see Section 2.2) is the most appropriate for implementing the previous functionalities. Our analysis indicates that the most appropriate token to be used by our reward mechanism is the utility token (such as, ERC20 in Ethereum), which is interchangeable (fungible) and indistinguishable from another token of the same type. Two approaches for token management can be adopted:

1. A token economy global to the system: all the items, e.g., restaurants, share the same typology of tokens, and a user who gains a token from restaurant A can spend it in restaurant B. This model is simple but it does not allow the item owners to decide the discount system policy, like the total number of tokens in the system or the value of a single token, that are instead decided by the system owner.
2. A token economy local to the item: each item creates and manages its tokens and, consequently, a token issued by restaurant A can be used at restaurant A only. In this way, item owners have more control over their token economy, but the system is more complex.

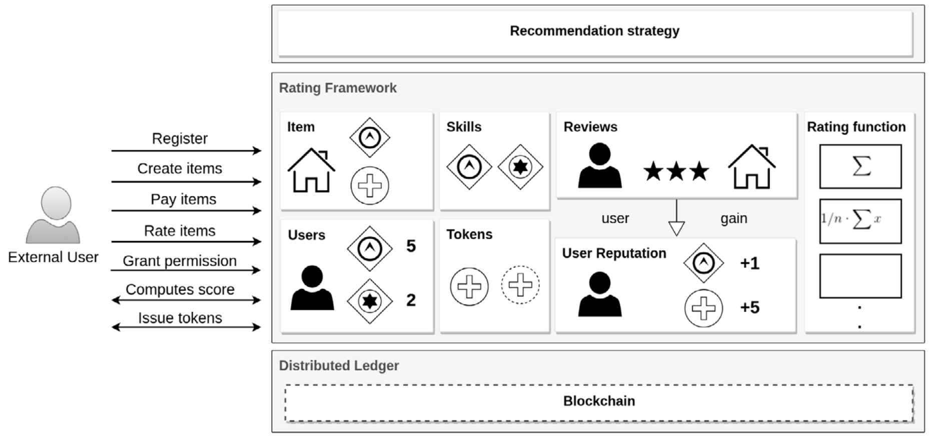


Fig. 1. Global view of the proposed decentralized rating framework with internal components and operations, bringing together blockchain and recommendation strategy.

In our framework, we adopt the local token management method.

Unlike centralized control RSs, our system allows the definition of custom evaluation metrics called *Rating functions* to rank items. Users can propose their methods to compute the final score of an item exploiting the data available on the blockchain. In Section 3.2 we show how such functions apply in our system, and in Section 4.3 we show a few concrete examples.

As shown in Fig. 1, the proposed framework provides its users with the following set of operations implementing the system functionality. The `Register` operation takes care to instantiate a new user in the system and assigns a new identifier to such a user. A new user starts with no tokens and no experience, i.e., his reputation has value 0 for all the existing skills.

Each user of the system can add new items with the `Create item` operation, assigning to such items the public information needed to recognize them, and the properties characterizing those items. The other users can review such items.

After using the service provided by an item (e.g., after having eaten at a restaurant), a user can pay using the blockchain native currency. In this case, to enable the payment, the item owner issues the `Grant permission` operation, specifying the user and the amount to be paid. Subsequently, the user exploits the `Pay item` operation to pay for the service. If the user has a positive balance of tokens, such tokens can be converted as a discount to that payment. If the payment is successful, the right to review the item is automatically given to the user by the `Pay item` operation. However, in case the payment is not executed exploiting the blockchain native currency (e.g., the user pays with fiat currency[5]), our framework provides to the item owner an alternative version of the `Grant permission` operation which directly gives the right to review the item to the user, without being subordinated to the `Pay item` operation. In this case, the token discount will not be applied, and the item owner should issue the `Grant permission` operation only when he verified that the payment has been successfully executed.

The `Rate item` operation stores a review record that includes the identifier of the user, a timestamp, the current reputation value of the user on the item's skill, and the customer review (a score and textual information). The `Rate item` operation automatically triggers the `Issue Tokens` operation, which rewards the user by issuing him an amount of tokens proportional to his current reputation value on the item's properties and improving such reputations. The user reputation is meant to grow following a model global to the system, decided by the system owner. In case an item has more than one property (e.g., Vegan and Wine), the system can be modeled to improve the user's reputation on all the corresponding skills. However, this detail could be also left to the item owner who, at item creation time, decides the number of user skills to improve after a review of his item.

Users can explore the available items and see how they are rated through the `Computes score` operation. In particular, the user can choose which rating function (among the available ones) the system must use to compute the score of an item. Each rating function computes the score of an item using a different formula. For instance, a rating function could compute a simple average of the ratings given by the users, while another rating function could compute a weighted average, using the user reputation as weights for the ratings. Since the information stored on the blockchain is public, each user could even decide to implement custom rating functions.

A *recommendation strategy* component is also represented in Fig. 1, which has the goal of implementing recommendation technique exploiting the information stored in the blockchain. Given the complexity of such algorithms, we assume this component to run off-chain. As a result, the owner of the rating framework can use any recommendation strategy to execute predictions and recommendations on the items, for example with content-based or collaborative filtering. In addition to the information stored on the blockchain, the recommendation strategy can also use other sources of information: for instance, recent recommendation strategies have increasingly incorporated social, demographic, or geographic information to improve the accuracy of the prediction [28].

3.2. System architecture

In this section we present the architecture of the proposed system, following the model previously described. Fig. 2 lists the contracts implementing the entities, the concepts and the operations shown in Fig. 1. Each box represents a smart contract and shows the attributes and the functions it supports. Each smart contract has an `owner` field that stores the address of the creator, omitted in Fig. 2 for simplicity. The relationships between the smart contracts are modeled following the UML standard for class diagrams.

The `RatingSystemFramework` (RSF) contract is responsible of creating, storing, and deleting `User` contracts that model the users of the rating framework. The `User` contract is responsible of creating, storing, paying and rating items. The `Item` contract models an item by storing the related reviews (named `Rating`),

[5] Fiat money, Investopedia: https://www.investopedia.com/terms/f/fiatmoney.asp [Accessed 15-January-2020].

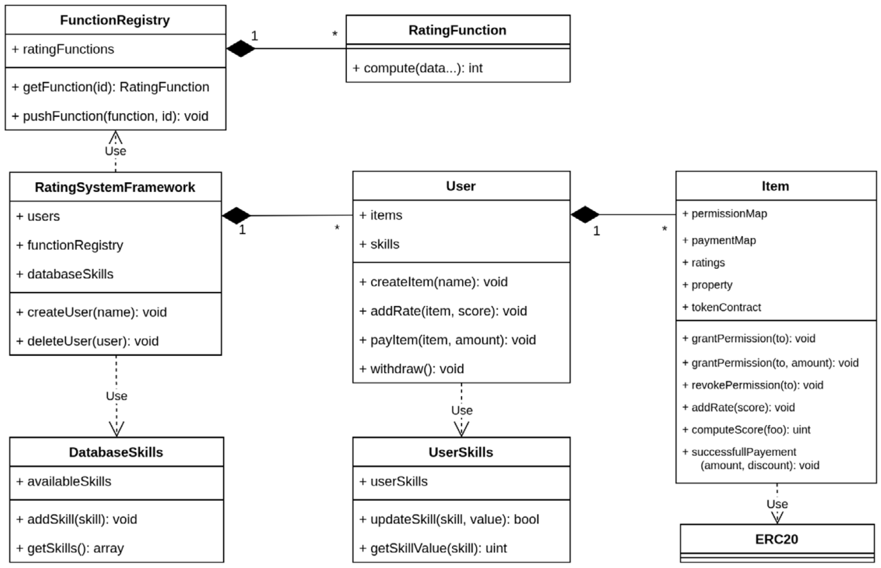


Fig. 2. Overview of the main contracts that implement the system model.

the map (named paymentMap) recording which Users need to pay for having used the item, and a permission map describing which Users have the permission to rate the item (named permissionMap).

As shown in Fig. 2, the relationship between the RSF and User contracts is one-to-many, as well as between User and Item. The contract UserSkills stores the skills and the corresponding reputation values of a User, modeling the reputational reward mechanism of the framework. The contract ERC20 implements a token following the ERC20 standard and models the economical reward mechanism of the framework. Each UserSkills smart contract is paired one-to-one to a User smart contract, while a ERC20 smart contract is paired one-to-one to an Item. The DatabaseSkills contract is a registry storing the items' properties supported by the system: therefore, the users' skills match those stored in this registry. The FunctionRegistry contract is in charge to store the rating functions. The latter two contracts should be singleton within a RSF.

To simplify the description of the architecture, we introduce the following actors: (i) Alice, who wants to deploy a recommendation service for different types of establishments in the restaurant field (such as restaurants, bars, and pubs); (ii) Bob, who is the owner of a bar; (iii) Carl, a customer of Bob's bar. We assume that Alice, Bob and Carl have addresses *0xalice*, *0xbob* and *0xcarl*, respectively.

System deployment. Fig. 3 shows the system owner, Alice, who is in charge of the deployment of the RSF contract which automatically creates the registries FunctionRegistry and DatabaseSkills. All the contracts will have stored *0xalice* as their owner. The RSF contract has initially no data, as well as the two registries. Therefore, Alice needs to populate the registries with a few rating functions and properties (skills) to bootstrap her system.

User module. Once the system has been deployed, the bar owner named Bob can subscribe himself with the createUser() operation provided by RSF smart contract: the operation deploys a new instance of User contract, at address *0xbobUser*, and its personal skill contract UserSkills: both of the contracts store *0xbob* as their owner. We assume Carl to be registered to Alice's RSF as well, thus having his own User contract, at address *0xcarlUser*, and UserSkills as shown in Fig. 3. Now both Bob and Carl are officially users of Alice's platform, with no items and with no reputation on any skill.

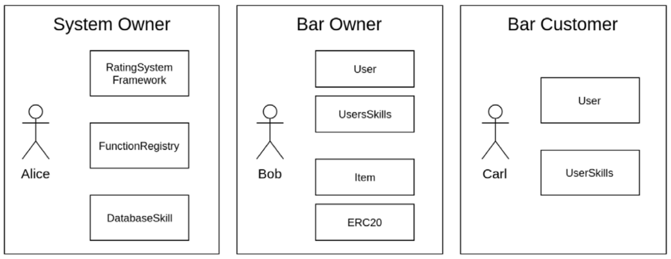


Fig. 3. The actors and their deployed contracts.

Item. In order to create an item representing his own bar, Bob needs to call the createItem() operation on his User contract. It is worth noting that only the owner of a User contract is allowed to create items. The createItem() operation deploys a new instance of Item contract (storing *OxbobUser* in the owner field in our example), as well as a new instance of token contract related to that item. Our framework pairs a distinct token contract to each item, meaning that these tokens can be used as a discount to that item only (the local model described in Section 3.1). Each token is identified by the token name, symbol, and its value in cryptocurrency. The maximum number of tokens available for an item is defined at deployment time. The value of each token is a fundamental point for the item owner because it can significantly influence the success and competitiveness of the system. For instance, a low value for a token may result in loss of interest and demotivation for users. On the other hand, a high value for a token can result in monetary damages and risk for the item owners due to the reward mechanism. Now is possible, for other users, to rate Bob's bar.

Skill module. The skill module involves three different actors, the system owner (Alice), the item owner (Bob), or the user (Carl). The system owner populates the DatabaseSkill registry with the proper set of skills to be used to characterize items. When creating a new item, the item owner (Bob) selects the skill to be assigned to his item. We remark that during the creation of a User contract, the operation deploys also a UserSkills contract. This contract is a personal database of the user, storing the reputation value of his skills. After reviewing an item (see Rating module paragraph below), the user's skill reputations matching the item's properties are improved.

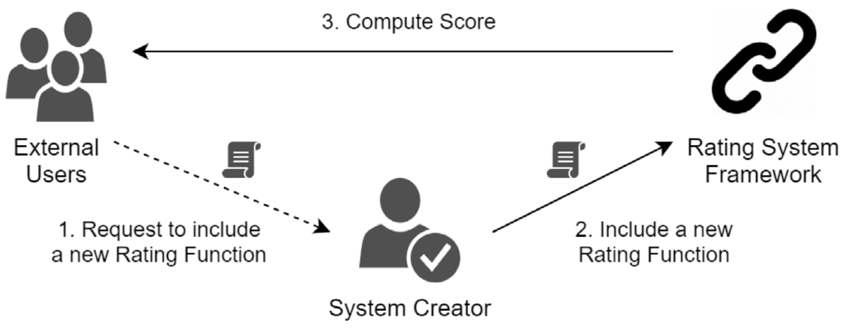

3. Compute Score
External Users
Rating System Framework
1. Request to include a new Rating Function
2. Include a new Rating Function
System Creator

Fig. 4. Workflow to include a new rating function.

Payment. The proposed system allows users to make payments for an item using the official cryptocurrency powering the underlying ledger. In such a case, the payment operation executed by the blockchain guarantees that the rating permission is granted to users only after they have successfully paid the bill. As for instance, after Carl had breakfast at Bob's bar, he decided to pay exploiting the `PayItem()` function provided by his `User` contract. The operation computes the discount (if Carl has a positive balance of tokens), transfers the currency from Carl to Bob's `Item` contract and, finally, it grants to Carl the rights to rate such item. However, in case the payment is not executed exploiting the blockchain native currency (e.g., Carl pays with fiat currency), our framework provides the item owner Bob with a function to allow Carl to rate the item as well. In this case, Bob is in charge to check that the payment has been successfully performed.

Rating module. To rate an `Item`, Carl needs permission from Bob. Bob, as item owner, unlocks the permission to other users to rate his items through the `grantPermission()` operation provided by the `Item` contract. As previously explained we have two cases: (a) Carl pays with the blockchain cryptocurrency; (b) Carl pays with other means of payment (e.g. fiat currency). To discriminate these two cases the `Item` contract provides two distinct versions of the `grantPermission()` operations:

(a) `grantPermission(to, amount)`: Bob commits the `Item` contract to set the permission to rate it to the user specified as first parameter of the operation only after he has paid a given `amount` with the system currency through the `payItem(item, amount)` operation of his `User` smart contract;
(b) `grantPermission(to)`: Bob sets the permission to rate his item to the user specified as parameter of the operation. This operation is used when Carl pays outside the blockchain system, and it requires Bob to check that the payment has been successfully executed.

Once Carl has the permission set, he can rate Bob's item by invoking the `addRate()` operation on his `User` contract, at address *0xcarlUser*, within a time window expressed in number of blocks. The `addRate()` operation expects as input parameters the address of the `Item` contract to rate and a score value. The function calls the `addRate()` operation of the input `Item` contract who checks Carl's permissions and, if positive, it resets them, it updates its state with the new `Rating` and rewards Carl with an amount of tokens proportional to the value of his reputation on the item's property. Lastly, the function updates the reputation on Carl's skill corresponding to the item's property.

Rating function module. A *rating function* computes the score of an item according to a specific formula. In our framework, the `RatingFunction` interface is a smart contract exposing only one

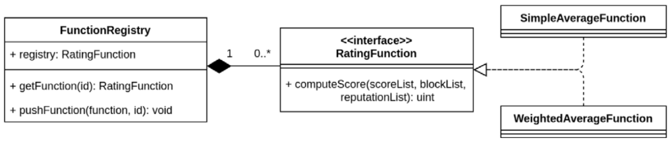

FunctionRegistry
+ registry: RatingFunction
+ getFunction(id): RatingFunction
+ pushFunction(function, id): void
1
0..*
<<interface>>
RatingFunction
+ computeScore(scoreList, blockList, reputationList): uint
SimpleAverageFunction
WeightedAverageFunction

Fig. 5. `RatingFunction` implementations.

method called `compute()` that calculates the overall score of an item. A rating function can use various parameters, such as the ratings registered in the blockchain, their block indexes as time references, and the reputation values associated with the users when they submitted such ratings. The `FunctionRegistry` contract is in charge to store `RatingFunction` implementations and can be populated only by who deployed the `RSF` (Alice in our example): this should keep the registry clean by unnecessary implementations, such as bad or repeated ones. Fig. 5 shows the relationships among the listed contracts, while Fig. 4 shows the workflow to include a new rating function: the continuous arrows represent smart contract functions, while the dashed arrow an action outside the system. To compute the score, the `Item` contract provides the `computeScore()` operation that accepts as input a `RatingFunction`: Carl can choose a `RatingFunction` from the registry and use it to compute the score of Bob's bar.

Workflow of the review process. Let us suppose that Carl orders a glass of Chianti at Bob's bar. The sequence diagram in Fig. 6 shows the workflow of the review process in terms of smart contract calls, and the OR box distinguishes the case of payment with the native blockchain cryptocurrency or payment with other means (e.g. fiat currency). Let us suppose that Carl pays for his glass of Chianti using the blockchain cryptocurrency.

1. At first, Bob invokes the `grantPermission()` operation on his `Item` contract, at address *0xbobItem*, passing as parameters the address of Carl's `user` contract, *0xcarlUser*, and the amount to be paid (step `1a.grantPermission(`*0xcarlUser*, `amount)` in Fig. 6). This operation will allow Carl to obtain the permission to rate Bob's bar as soon as he pays that amount to Bob's `Item` contract.
2. To pay his glass of Chianti, Carl invokes the `payItem()` operation on his `User` contract, at address *0xcarlUser*, passing as parameters the address of the Bob's `Item` contract, *0xbobItem*, and the `amount` he must pay (step `2.payItem(`*0xbobItem*, `amount))`. This operation first retrieves the tokens owned by Carl from Bob's item (this requires multiple interactions summarized in step `3.getTokens()`), it computes the discount that will be granted on the `amount` by using these tokens, it authorizes Bob's `Item` contract to transfer such tokens to itself through the `approve()` operation of the ERC20 token contract (step `4.approve(`

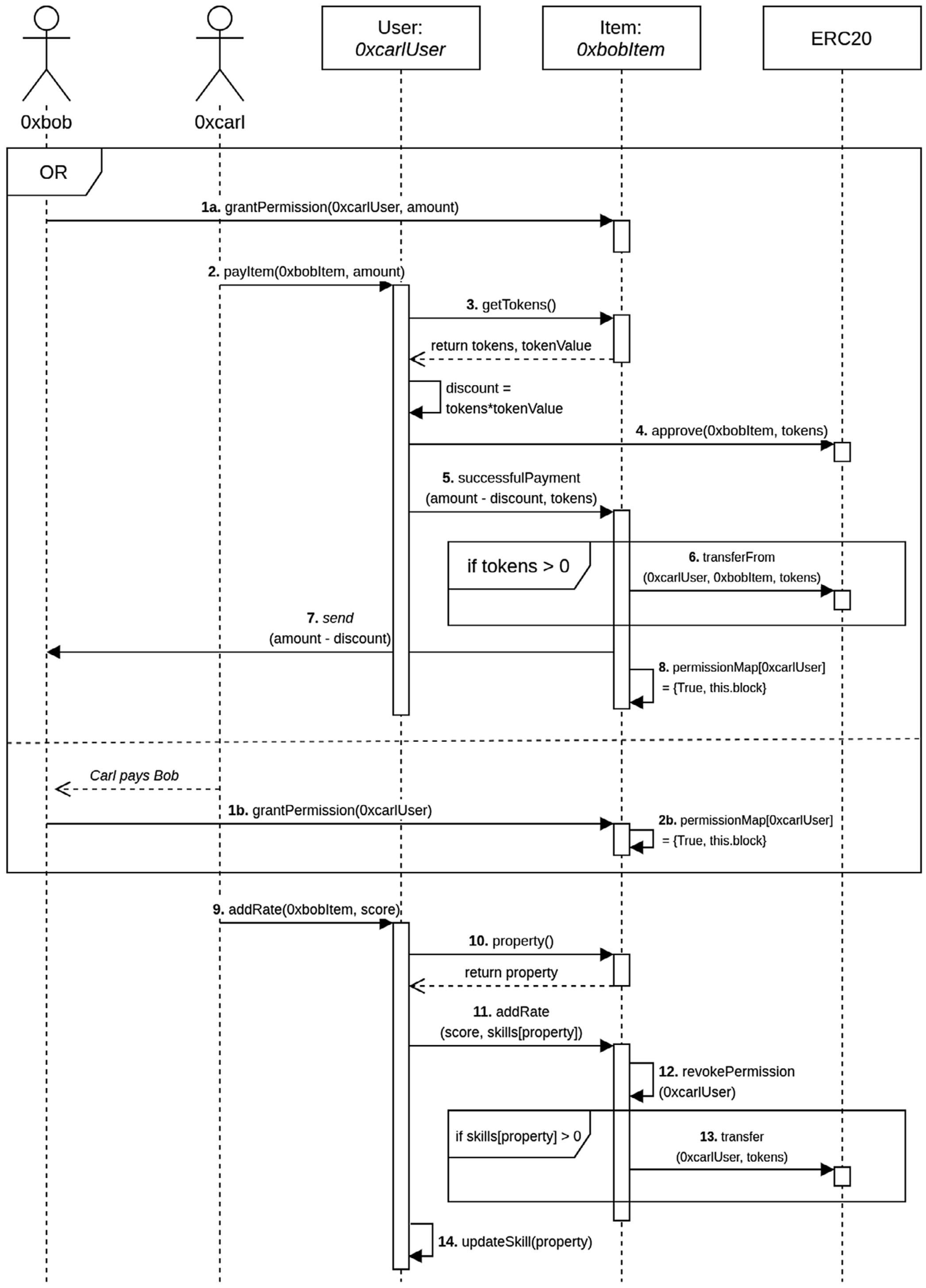

User: 0xcarlUser
Item: 0xbobItem
ERC20
0xbob
0xcarl
OR
1a. grantPermission(0xcarlUser, amount)
2. payItem(0xbobItem, amount)
3. getTokens()
return tokens, tokenValue
discount = tokens*tokenValue
4. approve(0xbobItem, tokens)
5. successfulPayment (amount - discount, tokens)
if tokens > 0
6. transferFrom (0xcarlUser, 0xbobItem, tokens)
7. send (amount - discount)
8. permissionMap[0xcarlUser] = {True, this.block}
Carl pays Bob
1b. grantPermission(0xcarlUser)
2b. permissionMap[0xcarlUser] = {True, this.block}
9. addRate(0xbobItem, score)
10. property()
return property
11. addRate (score, skills[property])
12. revokePermission (0xcarlUser)
if skills[property] > 0
13. transfer (0xcarlUser, tokens)
14. updateSkill(property)

Fig. 6. Workflows of review process operations.

0xbobItem, `tokens`)), and finally it invokes the `successfulPayment()` function on the Bob's `item` contract, at address *0xbobItem*, (step `5.successfulPayment(amount-discount, tokens)`). This last function triggers the transfer of Carl's tokens from the ERC20 smart contract to *0xbobItem* (step `6.transferFrom(`*0xcarlUser*, *0xbobItem*, `tokens)`), operation authorized in step 4. Supposing that such tokens are successfully transferred to Bob's `Item` contract, the contract checks whether the amount sent by Carl matches the residual amount expected by Bob. Supposing that the amount sent by Carl covers the residual amount to be paid, then Bob's `Item` contract transfers to Bob such amount (step `7.send(amount-discount)`) and, finally, it gives the permission to Carl to rate Bob's item by adding to the permissionMap a new entry (step 8).

3. In order to rate Bob's item, Carl invokes the `addRate()` operation on his `User` contract, passing as parameters the address of Bob's `Item` and the related score (step `9.addRate(`*0xbobItem*,`score)`). The `addRate()` operation, at first, retrieves from the `Item` contract the related property, which is *Wine* in our example, (step `10.property()`),

and then calls the homonym function of the `Item` contract, passing as parameters the received score and Carl's reputation value on Bob's item property (step `11.addRate(score,skills[property])`). Bob's `Item` revokes the permission to `0xcarlUser` (step `12.revokePermission(`*0xcarlUser*`)`) and issues the right amount of tokens to Carl's `User` contract if his reputation value for the *Wine* skill is positive (step `13.transfer(`*0xcarlUser*`,tokens)`). Finally, Carl's `User` contract increases Carl's reputation related to *Wine* through the `updateSkill()` operation (step `14.updateSkill(property)`). This operation also involves the `UserSkills` contract (not shown for simplicity).

At any time in the future, Carl can get, through the `withdraw()` operation of his `User` contract, the cryptocurrency unspent because of the discount. Instead, in case the payment is not executed exploiting the blockchain native currency (e.g., Carl pays with fiat currency), Bob invokes the (b) version of the `grantPermission()` operation that gives to Carl the right to rate the item without requiring him to execute the `payItem()` operation (step `1b.grantPermission(`*0xcarlUser*`)`). Obviously, in this case Bob is in charge to check that the payment has been actually executed.

4. A prototype on Ethereum: implementation and evaluation

In this section we present a prototype implementing the proposed system on Ethereum, and we show an extensive evaluation on a live test network. In particular, Section 4.1 describes the prototype and shows the gas price of the main operations. Section 4.2 shows the evaluation of the operations `payItem()`, `addRate()`, `createItem()` and `grantPermission()`, while Section 4.3 shows the evaluation of the rating functions.

4.1. Prototype description

The prototype has been implemented on top of Ethereum [15] and is publicly available on Github.[6] Stable and participated testing Ethereum networks are currently available, as well as tools for the development of smart contracts, and a standard definition of tokens (see Section 2.2). The smart contracts have been implemented in Solidity programming language following the system architecture shown in Fig. 2. The local development and testing have been done with the help of the Truffle[7] framework, which also provides to the developer commands to upload the compiled version of the contracts (i.e., EVM bytecode) onto the main and testing Ethereum networks. The accounts have been created and managed using the Metamask[8] wallet.

The prototype is a simplified version of the model and architecture presented in the previous sections. For simplicity, a `Rating` is an integer in the range [1, 10] and does not store textual information as this field is not fundamental to our evaluations. Moreover, due to the gas system of Ethereum, unbounded textual information can be costly, and we cannot make a-priori assumptions on the textual length. The tokens follow the ERC20 standard, and each `Item` contract creates its ERC20 token as explained in Section 3.1. For simplicity, the current implementation of the system supports a single property per `Item`. The prototype implements both the versions of the `grantPermission()` operation described in Section 3.2.

Table 1
Analysis of the gas consumed by the operations considering a gas price of 15 Gwei.

Contract	Operation	Cost in gas	Var %	Eth-15Gw
RSF	*deploy*	7,675,723	+ 76.5%	0.1151
RSF	createUser	4,576,056	+ 100.4%	0.0686
User	createItem	2,313,822	+ 122.7%	0.0347
User	payItem	96,687	–	0.0014
User	addRate	347,004	+ 108.6%	0.0052
Item	grantPermission[a]	79,593	+ 1.1%	0.0012
Item	grantPermission[b]	78,785	+ 0.06%	0.0012
DatabaseSkill	addSkill	68,992	–	0.0010
FunctionRegistry	pushFunction	75,637	+ 0.7%	0.0011

[a] grantPermission(to, amount).
[b] grantPermission(to).

4.1.1. Gas evaluation

In this section, we provide the reader the costs in gas of the functionalities of our prototype. Indeed, with respect to our previous work, [16], the addition of the support for managing skills and tokens increased the complexity of the framework and, consequently, the gas used to execute most of the operations.

Table 1 shows the cost in gas (column *Cost in gas*) for the most relevant operations of our framework and the related cost in Ether (column *Eth-15Gw*) that will be charged to the caller for the execution of such operations with a gas price of 15 Gwei. The bytecode of the `RSF` contract needs to be manually loaded onto the network by means of a client: in Table 1 we use the term *deploy* to indicate this operation. All the other contracts of our framework, instead, are created by contracts of the framework itself, as we will show in the following.

Considering the change ether:euro of 1:110.01 at the time of writing, we have that the `payItem()` and both the `grantPermission()` operations cost about 0.15 and 0.13 euros respectively for a user, comparable to the cost of `addSkill()` and `pushFunction()` for the system owner, while the cost of the `addRate()` operation is about 0.57 euros. Instead, the operations `createUser()` and `createItem()` cost about 7.54 and 3.81 euros respectively. For item owners these costs represent an investment for their activities; for customers these costs are paid back by the introduction of the tokens.

Table 1 also shows the percentage of the gas overhead (column *Var %*) with respect to the gas consumed by the corresponding operations in the previous version of the system [16], introduced by skill and token management for the implementation of the reputation and the reward mechanisms. The overhead of the `payItem()` and `addSkill()` operations is absent because these operations were not implemented in the previous version. The overhead introduced in the `grantPermission()` operation is very small because its main logic is identical to the previous version except for some small changes. The execution of the token transfer and the increasing of the user reputation is performed in the `addRate()` function: this is the reason for the relevant cost increase with respect to the previous version. Finally, the cost of deployment of the `RSF` and the cost of the create operations provided by `User` and `Item`, raises because these operations involve the deployment of contracts on the blockchain, and such contracts have larger bytecode because they embed additional

[6] Prototype repository: https://github.com/DistributedSystemsSocialNetworkAnalysis/Decentralized-Rating-Platform.[Accessed 15-January-2020].
[7] Truffle: https://www.trufflesuite.com/. [Accessed 15-January-2020].
[8] Metamask: https://metamask.io/. [Accessed 15-January-2020].

code for skills and tokens management. Although we expect, for a single user, the need to deploy the `User` contract only once, and the `Item` contract only a few times.

4.2. Evaluation of the state update operations

In this section we provide the reader with a detailed evaluation of the performance in terms of throughput of the main operations of our framework involving a transaction. This section is structured as follows: Section 4.2.1 describes the setup of our experimentation, Section 4.2.2 lists the evaluation metrics, and finally Section 4.2.3 describes the results achieved.

4.2.1. Experimental setup

We deployed the smart contracts composing the system on an Ethereum testing network and we used transactions to trigger the execution of each operation. We chose Ropsten[9] because it uses the same consensus (PoW) of the Ethereum main network at the time of writing.[10] Our experiments rely on the Infura service,[11] which acts as a proxy to a remote Ethereum node to interact with the Ethereum main and test networks. For each operation we evaluated, we created n batches of, respectively, $[N_1, N_2, \dots, N_n]$ requests, and for each batch i we sent to the network a sequence of N_i transactions and we waited for all of their resolutions. Each test is repeated multiple times, and performed for gas prices of 10 and 20 GWei. The input of each of these transactions is randomly generated within a valid range of values.

4.2.2. Evaluation metrics

For each batch size N_i we aim to measure four different metrics: (a) the average number of blocks required to execute/store all the transactions; (b) how many blocks elapsed on average from the time all the transactions are submitted to the time all the transactions are stored in the blockchain; (c) the average number of our transactions in each block (taking into account only the blocks containing at least one of the submitted transactions); (d) the time elapsed, in seconds, to process the batch. The time required for executing a transaction depends on both the network latency to propagate requests and the mining. At the time of writing, Ethereum has an average mining time of 13–14 s.[12] An important factor is that, given multiple transactions, their mining latencies do not necessarily add up since a single block may contain several transactions whose total gas cost stays below the block gas limit, limiting the computation that can occur per block. At the time of writing the blocks of Ropsten test network can contain transactions for at most 8M units of gas.[13] We chose to test the following operations: `payItem()`, `addRate()`, `createItem()` and `grantPermission()`. The gas costs of these operations have different orders of magnitude (see Table 1).

4.2.3. Results

Given the cost of the operations shown in Table 1 and the gas limit of Ropsten, we expect that each block can contain at most 3 `createItem()` transactions, or 100 `grantPermission()` transactions (both versions), or 82 `payItem()` transactions, or 23 `addRate()` transactions. Because of that, for each operation we used different batch sizes.

Fig. 7 shows the experimental results concerning the `payItem()` operation. Each plot in the figure shows on the x axis the size of the batches N_i, and on the y axis the measurement as described in Section 4.2.2. Fig. 7(a) shows how the number of blocks required to store all the `payItem()` transactions we requested in the same batch increases when the batch size goes from 1 to 100 transactions. For instance, on average, to process a batch consisting of 25 transactions are required about 1–2 blocks, while to process a batch of 100 transactions are required about 3 (gas price 20GWei) or 2 (gas price 10GWei) blocks. Since the blocks containing the transactions we issued in the same batch could be not contiguous, we show in Fig. 7(b) how many blocks elapsed from the time all the transactions of the batch are submitted to the time all the transactions are stored in the blockchain. In this case we observe only a slight increase of no more than half a block on average. We investigate in more detail the average number of transactions recorded in each block in Fig. 7(c). The results indicate that the average number of transactions per block goes up to about 45 (with gas price 10GWei) with a batch of 100 transactions while we previously saw that a block could theoretically host up to 82 `payItem()` transactions. Fig. 7(d), considers the time Ropsten takes to execute all the transactions on a batch. The figure shows that the median time to mine a batch of 100 transactions is about 53.5 s.

Fig. 8 shows the experimental results concerning the `addRate()` operation. Like in the previous case, the number of blocks required to store all the transactions of a batch increases with the size of the batch (see Figs. 8(a) and 8(b)). For instance, our experiments show that to process a batch of 35 `addRate()` transactions are needed about 2–3 blocks and elapsed about 3–4 blocks on average. Fig. 8(c) shows that in our experiments the average number of transactions processed in the same block ranges between 12 and 14 with a batch of size 35, while we previously saw that a block could theoretically host up to 23 `addRate()` transactions. The median time of the execution of a batch of 35 `addRate()` operations is about 47.5 s.

Fig. 9 shows the same evaluations concerning the `createItem()` operation. Fig. 9(c) shows that each block contains, on average, about 1.5 transactions, while we previously said that each block could theoretically host up to 3 `createItem()` transactions. As a matter of fact, Fig. 9(a) shows that the number of non empty blocks containing the transactions of a batch is lower than the number of transaction of the batch itself. Such blocks are not contiguous: for example, for a batch of 8 transactions, Fig. 9(b) indicates that the number of blocks created during our experiment, about 9–10 blocks, is higher than the total number of blocks containing the transactions of our batch, i.e., about 5 blocks in Fig. 9(a). Furthermore, independently from the gas price, the number of blocks mined with such transactions are very similar. Fig. 9(d) represents the time it takes to mine a set of transactions. The high variation of the time spent by the mining process is reflected in our measurements.

Finally, Fig. 10 represents the evaluations obtained from the execution of the `grantPermission()` operation. Fig. 10(c) shows that, submitting a batch of 100 `grantPermission()` operations, the number of transactions per block is about 26 transactions for 20GWei, and about 32 for 10 GWei, while the theoretical maximum number of transactions that can be hosted in a single block is about 100. Fig. 10(a) shows the number of blocks containing the transactions composing our batches. Finally, Figs. 10(b) and 10(d) show the elapsed time to issue a batch of transactions in terms of elapsed blocks and seconds. The number of blocks mined for the execution of a batch of 100 transactions is about 4 (gas price 20 GWei) or 3 (gas price 10 GWei), while the median time necessary to execute all those blocks and transactions is about 45 s.

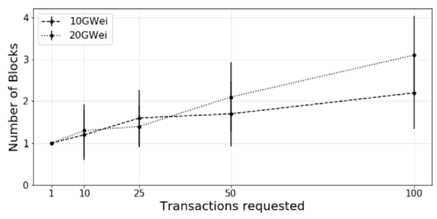
[9] Ropsten: https://ropsten.etherscan.io/. [Accessed 15-January-2020].

[10] We refer to Ethereum 1.0, to avoid confusion with the upcoming Ethereum 2.0.

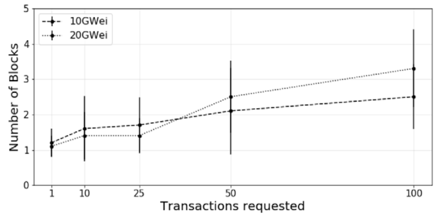
[11] Infura: https://infura.io/. [Accessed 15-January-2020].

[12] From Etherscan: https://etherscan.io/chart/blocktime. [Accessed 15-January-2020].

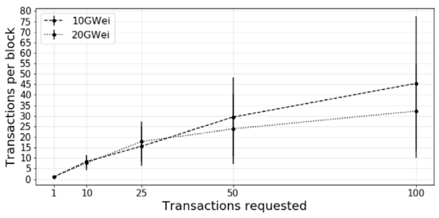
[13] With web3 connected to Ropsten: `web3.eth.getBlock("latest").gasLimit`.



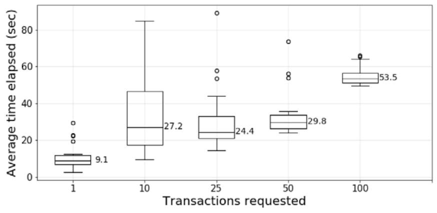
(a) Blocks containing transactions



(b) Elapsed number of blocks

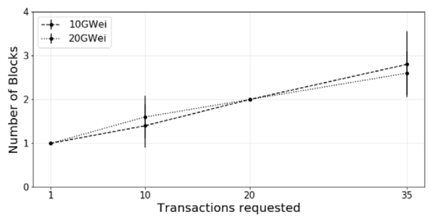


(c) Transactions per block

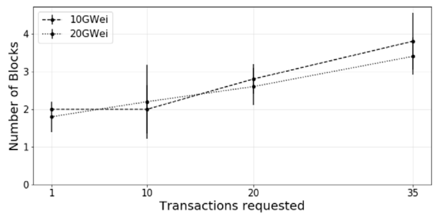


(d) Elapsed time

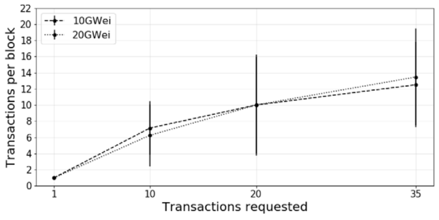
Fig. 7. Testing the resolution of multiple `payItem()` transactions.



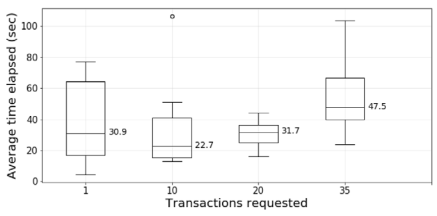
(a) Blocks containing transactions



(b) Elapsed number of blocks

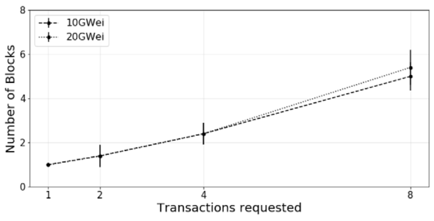


(c) Transactions per block

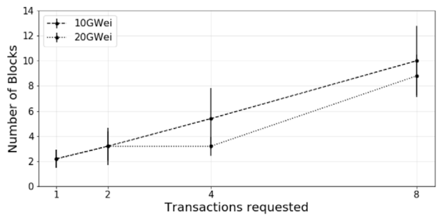


(d) Elapsed time

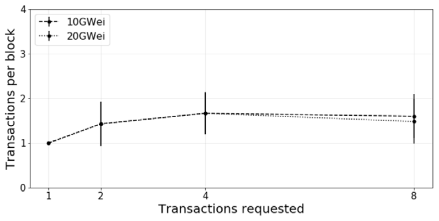
Fig. 8. Testing the resolution of multiple `addRate()` transactions.



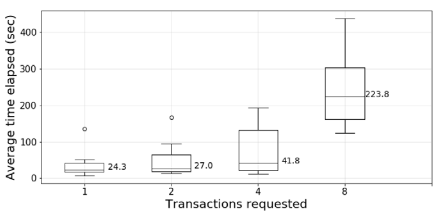
(a) Blocks containing transactions



(b) Elapsed number of blocks

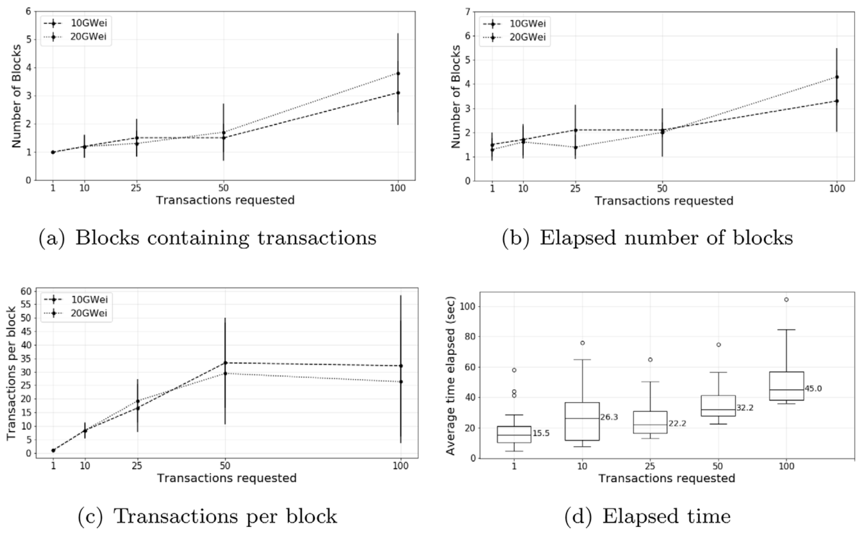


(c) Transactions per block



(d) Elapsed time

Fig. 9. Testing the resolution of multiple `createItem()` transactions.



(a) Blocks containing transactions
(b) Elapsed number of blocks
(c) Transactions per block
(d) Elapsed time

Fig. 10. Testing the resolution of multiple `grantPermission()` transactions.

4.3. *Evaluation of rating functions*

In this section we provide the reader a detailed evaluation of the performance in terms of latency and computational limitations of the rating functions that, differently from the previous operations, are read-only and therefore they do not involve a transaction. We expect the query of the score of an item to be the most frequent read operation in a rating system. This section is structured as follows: Section 4.3.1 describes the setup of our experimentation, Section 4.3.2 lists the evaluation metrics, and finally Section 4.3.3 describes the results achieved from our experiments.

4.3.1. *Experimental setup*

We performed the following measurements in the Ropsten test network using Infura as described in Section 4.2.1. We implemented some rating functions and, for each one, we deployed the corresponding contract. We remark that a rating function is a contract exposing a single operation called `computeScore()` with input three history arrays: the array of scores, that of timestamps, and that of reputation values (see Fig. 5). This allows a user to dynamically plug a rating formula based on their needs. To be able to implement different functions we cannot aggregate the data each time an item receives a new rating, therefore such operation requires the history of all the ratings. Even if such operation does not cost a fee, the execution is subjected to the gas limit anyway, and cannot scale indefinitely. In our experiments, for each rating function f_i, we call the `computeScore()` operation with arrays of lengths [1, 500, 1000, 2000, 3000, 4000, 6000] each filled with random data within the valid range (e.g. [1, 10] for scores). Each test has been repeated multiple times, and we reported the average value.

4.3.2. *Evaluation metrics*

For each function f_i we measure the latency to get the result, and the maximum size of the arrays (i.e., the maximum number of reviews) that is possible to process with a gas limit of 7M units. The formulas for computing the score of an item are summarized in Table 2: formula *(f1)* computes a simple average on the scores; formula *(f2)* computes the average value weighted on the age of the blocks (recent blocks impact more on the score); formula *(f3)* computes the average value weighted on the reputation value of the selected skill (the highest impacts more on the score); formula *(f4)* calculates the average weighted on both blocks ages and skills (a mixed weight of the last two approaches); finally, formulas *f5* and *f6* consider only the scores and compute the quadratic average and the median respectively. The quadratic average involves additional operations than the simple average, while the median is a measure often used. Since Solidity does not support float number computation, in our implementation the weights are in the range [1, 100], and the resulting score is truncated in the range [1, 10].

Table 2
Table showing the rating function.

Label	Description	Formula	
f1	Avg on scores	$\frac{1}{n}\sum_{i=1}^{n} s_i$	(1)
f2	W. avg on blocks	$\frac{\sum_{i=1}^{n} s_i \cdot w_i}{\sum_{i=1}^{n} w_i}; \quad w_i = \frac{(b_i \cdot 100)}{maxB}$	(2)
f3	W. avg on skills	$\frac{\sum_{i=1}^{n} s_i \cdot v_i}{\sum_{i=1}^{n} v_i}$	(3)
f4	W. avg on blocks and skills	$\frac{\sum_{i=1}^{n} s_i \cdot w_i}{\sum_{i=1}^{n} w_i}; \quad w_i = \frac{(b_i \cdot v_i \cdot 100)}{maxB}$	(4)
f5	Quadratic avg on scores	$\sqrt{\frac{1}{n}\sum_{i=1}^{n} s_i^2}$	(5)
f6	Median on scores	*median*(*scores*)	(6)

n = number of ratings received by an item; s_i = the score of the ith rating; b_i = the block of the ith rating; $maxB$ = the block of the latest rating; v_i = the reputation of the skill of the ith rating.

4.3.3. *Results*

Fig. 11 shows on the y-axis the average time required for computing the score of an item for each of the rating functions, varying the number of ratings taken into account (shown on the x-axis). The plot indicates that the latency of each rating function is very similar to each other, and some are subjected to a higher standard deviation, especially when the input size is larger. The gas used by each function depends on the complexity of the function itself. For instance, the function *f4* runs out of gas with 7000 ratings as input, *f2* and *f3* run out of gas with 10,000 ratings, while *f1*, *f5* and *f6* run out of gas with 12,000 ratings. In the city of Pisa, Italy, the main restaurants have a number of reviews that goes from a few hundred to a few thousand. As a result, at the time of writing, rating functions in Solidity cannot

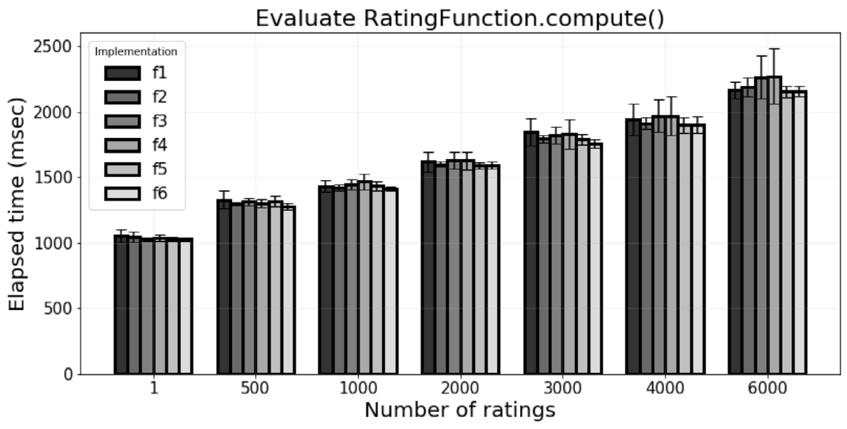


Fig. 11. Time required by the rating functions to return the result.

implement formulas too complex. As concerns the best formulas to be used as a rating function, there is no standard solution that is adopted by current RSs. Indeed, a rating calculation formula is domain-dependent [29] and most of them are based on modified versions of the previous formulas. For instance, the method used by Revain [30] is based on weighing the ratings with the user's experience and time elapsed (as we do with our function $f4$ in Table 2) while the approaches used by Amazon and Tripadvisor are unknown to the public. Authors in [31] compare various methods to estimate the quality of a product, and they find that the accuracy of a simple average formula is comparable to more complex methods (e.g., lower bound on the normal confidence interval, weighting authors according to maximum likelihood, and others). Similarly, the authors in [32] prove that the simple average produces a better estimation of the quality of the product than the median and the majority[14] rules when a product has more than 100 ratings.

5. A comparative analysis of recommender systems

This section provides the reader with a description of the works which have been done in the field of blockchain-based RS, and compares their rating supports with our work following both a qualitative and quantitative approach. In particular, in the following of this section we list the related works and we compare their general characteristics (Section 5.1), their architectures (Section 5.2), and their services (Section 5.3) with our work. Finally, in Section 5.4 we quantitatively compare our approach to those related works for which it is possible.

Several approaches have been proposed to manage the recommendation of items to users in different contexts. In the following, we consider some popular centralized RSs, as well as experimental proposals involving blockchain solutions. For instance, Tripadvisor and Amazon are among the most popular centralized RSs that collect feedbacks on destinations (e.g., restaurants or hotels) and products to provide a rating for such items and to recommend them to users. LinkedIn, instead, acts like an Online Social Network where companies are recommended to users who fit required skills. Gastroadvisor [33], a Tripadvisor-like system focusing only on restaurants, stores on the blockchain only special type of reviews, called gold reviews, that are submitted by users who booked the restaurant through the platform and paid the bill through FORK tokens, i.e., the platform currency. Friendz [34], an Instagram-like application focusing on product campaigns, exploits the blockchain to make the system transparent and decentralized. Both Gastroadvisor and Friendz have their tokens built on top of the public blockchain and a reward system to make the platform more appealing to users.

The authors in [36] propose the blockchain as a recordkeeping system providing a proof of the existence of educational contents introducing the Kudos as a "educational reputation digital currency". Educational centers can use such tokens to award students or educational achievements, who in turn can pay personal tutors with such tokens.

Another relevant work is proposed in [39], where blockchain is used as Personal Data Management System. The personal data of users relevant for the recommendation task are stored in encrypted form on the blockchain and they can be used by companies, i.e., third-parties interested in recommending their products and services.

Authors in [37] focus on preserving users' identity disclosure in RS by exploiting a blockchain-based solution. In particular, they use locality sensitive hashing classification as well as a set of recommendation methods using obfuscated ratings stored in the blockchain.

The approach we presented in this paper extends our previous work presented in [16] and proposes a methodology to increase the user motivation with reputation and economic rewards to minimize the user's economic impact due to the transaction fees characterizing public ledgers. Other works tackle the problem with different approaches, for example defining a novel Bitcoin-like general purpose blockchain where transactions are used to store reputation scores of users sharing files with each other [38], or creating a private network to limit the authority of participating nodes [40].

The approach proposed in [12] consists of an Ethereum smart contract with incentives to reviewers. The service provider generates a token for the user to write a review and the smart contract will pay Ether to the author of reviews. To highlight the novelty of our approach w.r.t. existing ones, we have summarized in Table 3 the properties of current RSs in terms of three categories: *general characteristics*, *system architecture*, and *provided services*.

5.1. General characteristics

The category *general characteristic* provides information about the type of items taken into account by the RS, the type of review, and the type of rating. Table 3 shows that the most of current RSs are designed around a specific set of item type: restaurants and destinations (Tripadvisor and Gastroadvisor), products (Amazon), social media contents [38], Friendz), business (Revain, [40], works and industry (Lina.Review, LinkedIn, [36]). Instead, there is a subset of approaches [12,37,39] designed to be general and to accommodate any type of item. As concerns the review, it

[14] To assign a product the rating equal to the most popular one.

Table 3
Comparison of current Recommendation Systems. (C = Centralized; B = Blockchain-based; L = Local; G = Global; R = Reputation; M = Monetization).

Approach	Characteristics			Architecture			Service			
	Items Type	Review Type	Rating	Control	Avoid Censorship	Auditability	Reward	Reward Policy	Payment	Reputation
Tripadvisor	restaurants and destinations	text	5-star	C	✗	✗	✗	✗	✗	✓
Amazon	products	text	5-star	C	✗	✗	✗	✗	✓	✓
LinkedIn	work positions based on skills	–	–	C	✗	✗	✗	✗	✗	✓
Gastroadvisor [33]	restaurants	text	5-star	B	✓	✓	ERC20 - RM	G	✓	✓
Friendz [34]	posts, profiles, app review	–	5-star	B	✓	✓	ERC20 - M	G	✓	✗
Revain [30]	cryptocurrency services	text	5-star	B	✓	✓	ERC20 - RM	G	✓	✓
Lina.Review [35]	industry, supply chain, healthcare	text	5-star 0-N	B	✓	✓	ERC20 - RM	G	✓	✓
[36]	intellectual works	–	–	B	✓	✓	kudos - R	G	✓	✓
[37]	any type of item	–	0-N	B	✓	✓	✗	✗	✗	✗
[38]	users	–	0–1	B	✓	✓	bitcoin - M	L	✓	✓
[39]	any type of item	–	–	B	✓	✓	discount - M	L	✓	✗
[40]	bids and offers for trade emission	–	–	B	✓	✓	✗	✗	✗	✓
[12]	any type of item	–	–	B	✓	✓	ether - M	L	✓	✗
[16]	any type of item	–	0-N	B	✓	✓	✗	✗	✗	✗
Our approach	any type of item	–	0-N	B	✓	✓	ERC20 - RM	L	✓	✓

mainly consists of a text and a rating, as in the case of Tripadvisor, Amazon, Grastroadvisor, Revain, and Lina.Review. Instead, some approaches either require only the rating (such as, Friendz, and [38]) or they do not specify whether the text is necessary (i.e., [36,37,39,40]). Lina.Review is designed to be general because it allows specifying a set of criteria to be reviewed from users. Instead, the approach proposed in [12] does not specify the structure of the review because it exploits off-chain storage to collect them while their hash pointers are stored on-chain. Most of the RSs require that the rating of a review is denoted by using either the 5-star system (e.g., Tripadvisor, Amazon, Gastroadvisor, Friendz, and Revain), a binary system (such as, Like/Not Like as in [38]), or a numeric rating (such as [37]). In some cases, the rating is not required (such as LinkedIn) or authors do not specify whether it exists (i.e., [36]). In addition, some approaches do not specify the type of rating [12,39,40].

Similarly to [12,37,39], our approach stores on the blockchain only the numeric rating (0-N) of the reviews, as it is the minimal information structure needed for providing transparent review collection and item score calculation. Indeed, each review can be associated with a large amount of data (e.g., text, images, or videos), but storing all of these data on the blockchain is very expensive and it could negatively impact the overall system performance and scalability [41]. However, the InterPlanetary File System (IPFS) [42] could be exploited to store such data.

5.2. System architecture

In this section, we summarize for each approach of Table 3 the properties related to the system architecture by focusing on the control of the platform (which could be centralized or based on blockchain architecture), the resistance of the system to censorship, and the verifiability of the information.

The most popular RSs, such as Tripadvisor, Amazon, and LinkedIn are implemented by exploiting a centralized control architecture, where the system is controlled by a single authority, which collects the reviews submitted by users, manages the reputations, and performs recommendation for the other users, and are accessible only through the services offered by them. Such centralized control of the service provider introduces some issues and risks for the users of the services. For instance, the central authority controlling the service could alter or delete existing reviews to maliciously modify the rank of an item [13,43], increasing the risk of censorship. Moreover, the users are unaware of (or cannot verify) the method used by the centralized authority to compute the rank of the items. Summarizing, users need to trust those centralized control RSs, because the latters could alter the score of items in many ways without the former being aware of this.

From an architectural point of view, several works implement the whole service (or a part of it) by exploiting blockchain-based architectures, where there is no central authority that controls the system and any user can verify the data written on the blockchain. For instance, Friendz and [40] exploit the blockchain to manage the activities related to brands and campaigns, while the personal data stored in the users' profiles are not publicly available. Gastroadvisor exploits the blockchain to manage reward and payment, making the review process more transparent and verifiable by users using the hash of the corresponding payment transaction. In Revain and [38], the blockchain platform is used to increase auditability and resistance to censorship because each data is hashed and then written in the blockchain. Lina.Review utilizes a hybrid architecture based on the Ethereum blockchain and the Lina Core, a private high scalable blockchain that is used to store complete and verifiable information about transactions. Authors in [36] exploit blockchain to store records of achievement and degree certificates. Such information is added to the blockchain by the awarding institution while students can access and share them. The approaches proposed in [12,37] exploit a decentralized storage system (such as IPFS [42]) to minimizes the information that needs to be stored and shared in the blockchain. The architecture of [39] is designed to store on the blockchain encrypted personal data which are relevant for recommender systems.

From an architectural point of view, our approach relies on a blockchain-based solution (such as Lina.Review and [12,36,37,39]) but, unlike previous works, the proposed framework manages the entire review process through the blockchain, increasing transparency, auditability of data, and preventing the risk of censorship from third parties. Indeed, our solution is entirely built on top of a public blockchain where the running smart contracts implement the reward and payment mechanisms, the management of ratings, and the computation of the reputation score obtained from users' reviews.

5.3. Service features

In this section, we investigate the features of the provided services, focusing on payment support, reputation, and reward supports. Indeed, as shown in Table 3, the reward mechanism can be based on *reputation* (R), on *monetization* (M), or both (RM). The former improves the status of an individual while the latter provides an economic reward. Furthermore, we indicate for each approach the mechanisms used to implement the reward (i.e., token, discount, currency, ether, etc.) and the corresponding reward policy, which could be Global if the reward policy is unique among all the item owners, or Local, if each item owner is enabled to create his specific reward policy for each of his items.

Tripadvisor, LinkedIn, and Amazon do not provide any economic incentive for users to complete their tasks. Among these systems, only Amazon supports traditional payment methods because Tripadvisor and LinkedIn are mainly focused on recommendations. In Tripadvisor and Amazon, the reputation of the users is computed based on the number of reviews they submitted and the number of likes received. LinkedIn, instead, assigns to users values on their professional skills.

Gastroadvisor incentives its users with a reward mechanism based on reputation proportional to the number of reviews, likes, and reshares the users received, and monetization through the FORK tokens gained when booking, writing reviews, or publishing multimedia contents. The reward policy is global, and all item owners have to comply with it. Friendz motivates its users to participate in content creation and validation by using the FDZ token that can be spent on in-app services. The platform does not consider the reputation of users because it provides Online Social Network-oriented services. Revain provides its users a reward mechanism based on monetization with the internal RVN tokens, and with two types of reputation: the level based one, gained when writing or commenting reviews (with the 10th level as the highest), and the "karma" one, computed on the likes/dislike received. Moreover, a 9th level user can apply to become an "expert". In Lina.Review, each user can submit a review request or a new advertising campaign, exploiting an internal ERC20 compliant token (named lina) as a reward and payment method (e.g., an advertising campaign can reward the first 1000 users with 1 lina for each click). Furthermore, also in Lina.Review users with a high reputation can apply and be promoted with the role of expert and are paid by the platform.

The platform proposed in [36] exploits a reputation-based rewards mechanism to incentivize students for small educational services. The cryptocurrency provided by the underlying blockchain is used as a reward, and the amount of currency owned by an organization is proportional to its educational reputation. Each recognized institution, organization, and an intellectual worker is paired with an educational reputation currency, named kudos, which depends on the Rankings for Universities, H-index for academics, and author rank for published authors. The approach proposed in [37] is mainly focused on the auditability of the information and it does not take into account any reward. Authors in [38] proposed a monetization-based reward mechanism to discourage users from behaving dishonestly when sharing a file. When users submit some data on the blockchain they include in each transaction the user who requested the files and the hash of the file. The receiver answers with a transaction with "1" if they are satisfied, "0" otherwise: the reputation of a user is the sum of such values. However, the reputation is not computed on the blockchain but by the peer's client, and it avoids collusion attacks averaging the scores received by a single peer. The platform proposed in [39] exploits a monetization-based reward mechanism whenever users share data with a company, gaining cryptocurrency or obtaining discounts on the company's products: as a result, this is a local reward policy. The reputation of the users is not considered. Authors in [40] proposed an approach to increase the auditability and reputation of users when reducing carbon emission as a function of past emission rates, but it does not take into account any reward mechanisms or payment methods. The reward mechanism proposed in [12] is based on monetization because it expects the item owners to provide some money as an incentive for the users through a smart contract: the item owner funds the smart contract, they assign access tokens to users, and in turn, the smart contract will pay Ether to the EOAs of the authorized reviewers.

Compared to our previous work [16], the proposed framework introduces several improvements because of the integration of: (i) a local reputation and monetization-based reward mechanism that models the experience of a user based on personal skills and it is implemented through a standard token, (ii) a component that integrates blockchain cryptocurrency and token-based payments. Compared to existing approaches, a novelty introduced by our framework is the full support of both a local reward policy, helping the item owner, and a reward system helping the user. When cryptocurrency or tokens are used for payment, our framework automatically gives to the user the permissions to rate the item. An external payment, instead, needs a manual intervention from the item owner. Finally, to evaluate the score of an item, a user can choose among a number different criteria, the rating functions, that can be dynamically added to the framework. As shown in Table 3, the proposed approach is the first method in the literature using smart contracts for providing a local reward mechanism based on reputation and monetization. The most similar works to our proposal are [12,38,39], which also implement a local reward policy. However, these approaches have two major limitations: the reward is based only on either reputation or monetization, and they are not designed to support efficient token-based rewards.

5.4. Experimental comparison

In order to experimentally compare the performances of the different methods listed in Table 3 we took into account common evaluation measures that are relevant for performance evaluation. Since our prototype implementation is based on Ethereum, we compared the proposed approach against other similar solutions that have been designed for Ethereum blockchain, and we used the gas consumed by the smart contracts as a common evaluation measure for the comparison. For this reason, the approaches we selected for the experimental comparison have to meet the following requirements: (i) they must be designed to support Ethereum blockchain, and (ii) they must make available the source code of the smart contracts implementing the approach. Centralized approaches, i.e., Tripadvisor, Amazon and LinkedIn, as well as the approaches proposed by [38–40] do not meet any of the above requirements. Therefore, they cannot be experimentally compared with our solution. The approaches proposed in [34,36,37] are based on Ethereum but they do not provide the source code. Instead, the approaches proposed by [12, 16], Gastroadvisor, Revain, and Lina.review are suitable to be directly compared with our proposal because they meet both the above requirements. Consequently, the available source code has been downloaded, compiled, and deployed by using the Remix IDE to measure the gas cost.

The smart contracts implemented by Lina.Review, Gastroadvisor, and Revain focus only on the definition of a custom ERC-compliant token, which is globally used to manage the reward and the reputation of users. The cost of the deployment of Lina.Review and Gastroadvisor is quite low: 1,456,919 and 2,066,430 units of gas respectively. Instead, in the case of Revain, the implementation of the token includes also a large set of general-purpose contracts that implement a number of utility functions, such as, a threshold multi-signature scheme and a contract for generating unique identifiers, a proxy, and a storage service for ERC20 token. The system provides few specifics on how such smart contracts interact with each other and we were unable to deploy the implementation of the token because the contract's gas requirement is larger than Remix block gas limit. Unfortunately, Lina.Review, Gastroadvisor, and Revain do not provide further details on how the management of the items, skills, reviews, and rating functions would work. Furthermore, their repositories do not expose the code of any function that is semantically similar to our operations (i.e., the `grantPermission` and the `addRate`).

The method proposed in [12] is the least expensive to deploy in terms of gas (999,145 units of gas), and its functionalities are more similar to our `Item` contract (whose creation costs 2,313,822 units of gas) rather than to the entire framework. As shown in Table 3, the approach in [12] provides few advantages in comparison with conventional methods. In particular, it provides users a local reward policy based only on monetization, which does not support reputation and token standard. The item owners have complete control over the generated permissions and the fairness of the payment process may not be guaranteed. Indeed, the item owner can arbitrarily decide to grant or not grant permissions to a customer, even if they pay with the native cryptocurrency. Also, the code used to compute the score of the items cannot be directly executed on the blockchain because data are stored on IPFS.

In order to compare in more details our approach with [12], we replicated the transactions experiments described in Section 4.2. In particular, we compared the function called `IssueToken` exposed by the smart contract in [12] against the `grantPermission` function exposed by our approach because they are semantically similar, i.e., they both set the permission to a user to leave a rating. Fig. 12 compares our approach with [12], showing the average and the standard deviation resulting from our experiment by considering the performance metrics explained in Section 4.2.2. In particular, the gas price used for all the invocations is 10Gwei, and we used the results showed in Fig. 10 for the `grantPermission` function. The costs of the two functions are comparable, i.e., 50,676 units of gas for the `IssueToken` function and 79,693 units or the `grantPermission` function. For this reason, we note that all the proposed measurements in Fig. 12 have similar performance results for the two tested functions. As for instance, for the execution of a batch of 100 transactions, both functions require about 3 blocks on average (see Fig. 12(a)), the average number of transactions in each block is between 30 and 40 (see Fig. 12(c)), while the average time required for appending the transactions to the blockchain is less than 55 s for both approaches.

Finally, the rating framework proposed in this paper is obviously experimentally comparable with our previous approach [16]. In particular, our previous version of the framework takes about 4,347,987 units of gas to be deployed. Hence, the additional features introduced in the version presented in this manuscript increase the deployment gas cost by 76% as shown in Table 1 (column *Var %*). The cost of the other operations increase as well. For instance, the cost of the `addRate` operation in the new version is twice the cost we had in the previous version.

6. Framework discussion

In this section, we focus on a critical analysis of the proposed framework discussing weaknesses and main threats.

Architectural limitations. A RS built on top of a permissionless blockchain technology benefits of transparency and immutability of the data but inherits some flaws of the permissionless blockchain as well. An important one is the limited amount of data that can be processed and stored over time. However, a number of solutions for increasing blockchain scalability are under investigation [44]. The fee is another blockchain-related aspect to be discussed. Section 4.1.1 shows that computing the rating of items does not consume gas, while the cost of uploading a new review on our rating framework is low, and mitigated by the reward. Creating a new user, instead, is more expensive, but this operation is executed only once for each user.

Unauthorized reviews attack. A relevant threat consists of a massive submission of unauthorized reviews, i.e., reviews from users who did not really exploit the service provided by the item. This event is known as a review bomb [45], and it can have the goal of destroying or increasing the score of an item. These are called, respectively, *nuke* and *push* attacks in [46]. A simple authorization mechanism, like the one adopted in our framework, could be used to prevent the insertion of reviews from users who did not really exploit the service.

Sybil attack. A second threat is the Sybil attack, which consists of impersonating multiple identities in order to insert more than one review to the same item, thus altering its score. This attack is easy to accomplish when obtaining a new identity is cheap, and it is popular in recommender systems [38,46,47]. It is possible to mitigate the effectiveness of reviews left by ad-hoc generated users with the concept of rating functions. For instance, if the result returned by the rating function *f4*, which considers the users' reputations, is considerably lower than the result returned by the function *f2*, which does not consider users' reputations, then it is reasonable to think that the item owner attempted to get positive ratings from new generated accounts.

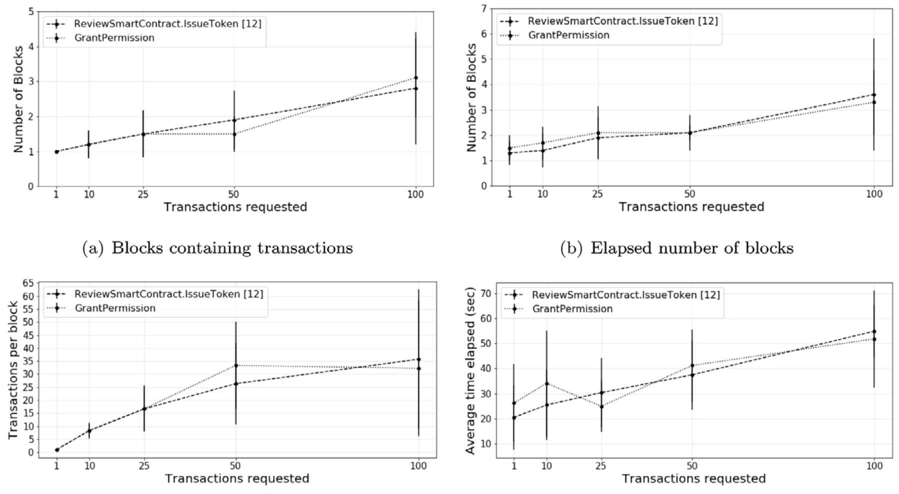
Collusion with high reputation users. A way for item owners to increase the score of their items is to bribe high reputation users and convince them to insert good reviews their items. We can mitigate this threat by restricting the system to accept only payments in cryptocurrency, so that the payment can be traced on the blockchain, thus attesting the exploitation of an item by a user. However, this countermeasure does not really solve the problem because the item owner could falsify service exploitation with arbitrarily small payments.

Fairness of the payment process. An important aspect is the fairness of the process described in Fig. 6, i.e., Bob should be paid by Carl, and Carl should receive permission to rate Bob's item. Since the services/products provided by items to users are not blockchain-related, e.g., the glass of wine Bob sold to Carl, our framework does not ensure that the item owner will be paid for it. However, in case Carl pays with cryptocurrency, our framework grants to Carl the review permissions only if he pays the amount specified by Bob. Then, it is up to Carl to decide whether he wishes to leave a rating. On the other hand, there are no guarantees when Carl pays in any other way (such as via fiat currency). In this case, Bob is responsible both to check that the payment has been performed, and to issue the `grantPermission` operation (b) to give Carl the permission to review his item. Several advanced payment techniques have been proposed in the literature to solve this issue, e.g., [48,49].

Malicious reviews. Finally, like any review platform, the proposed framework is vulnerable to ill-natured ratings, i.e., ratings submitted by authorized users of the system that do not reflect the real opinions of the users (*fake review*), or contain provocative contents (*trolling*), or mention off-topic contents. Lina.Review and [50] try to limit this problem by exploiting users with additional powers, like opinion leaders, to moderate or accept the reviews.

7. Conclusions and future works

This paper proposes a general decentralized rating framework based on blockchain, supporting recommender systems and rewarding its users for their reviews to compensate them for the cost they incurred due to the permissionless blockchain. In our approach, the ratings of items, the reputations, the tokens of users, and the algorithms exploited to compute the score of the items are stored on the blockchain, thus being publicly visible



(a) Blocks containing transactions

(b) Elapsed number of blocks

(c) Transactions per block

(d) Elapsed time

Fig. 12. Testing the resolution of multiple `grantPermission` and `IssueToken` transactions.

and not alterable. Users are incentivized to review items because: (i) their reputations concerning items' skills improve through the update mechanism defined in the system; (ii) reviews made by high reputation users impact most; (iii) higher reputations lead to a higher amount of tokens earned by reviewers for their reviews. The proposed system provides direct support to payments with cryptocurrency, but it also allows the item owner to accept other (off-chain) payment methods by manually invoking a smart contract to confirm the payment execution. Finally, the manuscript provides a comparison with the other systems proposed in the literature.

An interesting future work could be aimed at mitigating the misuse of the system that could be perpetrated by malicious high reputation users, for instance through the integration of advanced authorization systems [51,52], or by designing a sophisticated trust network [53,54] so that a user can label a set of known user as "trusted reviewer" even if they do not have a high reputation.

The problem of recommending the most appropriate item for each user is not trivial and the techniques proposed in the literature [28,55] are typically too expensive to be implemented by smart contracts. Hence, a future research direction could concern the definition of an efficient strategy that can run on top of such constrained resources. Finally, to overcome the limitations of public permissionless blockchains, the components building the proposed framework can be split between different blockchains, permissionless and permissioned, to find a tread-off between performance, immutability, and transparency [56]. Therefore, it is important to design an effective interoperation protocol: a survey of those protocols can be found in [56–58]. We find suitable for our case techniques based on Atomic Swaps [49] and Outsourcing Service Payment [48] since our framework supports the exchange of assets (a review permission for a payment, and a reward for a review) and cryptocurrency payments between untrusted parties.

CRediT authorship contribution statement

Andrea Lisi: Conceptualization, Validation, Investigation, Writing - original draft. **Andrea De Salve:** Conceptualization, Methodology, Writing - original draft, Supervision. **Paolo Mori:** Conceptualization, Supervision, Writing - review & editing. **Laura Ricci:** Conceptualization, Supervision, Writing - review & editing. **Samuel Fabrizi:** Conceptualization, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender systems: introduction and challenges, in: Recommender Systems Handbook, Springer, 2015, pp. 1–34.
[2] TripAdvisorInsights, Tripadvisor global report, 2017-2018, https://www.tripadvisor.com/TripAdvisorInsights/tripbarometer.
[3] A. De Salve, B. Guidi, L. Ricci, P. Mori, Discovering homophily in online social networks, Mob. Netw. Appl. 23 (6) (2018) 1715–1726.
[4] Y. Zhang, M. Chen, D. Huang, D. Wu, Y. Li, iDoctor: Personalized and professionalized medical recommendations based on hybrid matrix factorization, Future Gener. Comput. Syst. 66 (2017) 30–35.
[5] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al., The youtube video recommendation system, in: Proceedings of the Fourth ACM Conference on Recommender Systems, ACM, 2010, pp. 293–296.
[6] H. Zamani, M. Schedl, P. Lamere, C.-W. Chen, An analysis of approaches taken in the acm recsys challenge 2018 for automatic music playlist continuation, ACM Trans. Intell. Syst. Technol. (TIST) 10 (5) (2019) 1–21.
[7] A. Bogliolo, P. Polidori, A. Aldini, W. Moreira, P. Mendes, M. Yildiz, C. Ballester, J.-M. Seigneur, Virtual currency and reputation-based cooperation incentives in user-centric networks, in: 2012 8th International Wireless Communications and Mobile Computing Conference, IWCMC, IEEE, 2012, pp. 895–900.
[8] C.H. Declerck, C. Boone, G. Emonds, When do people cooperate? The neuroeconomics of prosocial decision making, Brain Cogn. 81 (1) (2013) 95–117.
[9] D. Jang, A.S. Mattila, An examination of restaurant loyalty programs: what kinds of rewards do customers prefer?, Int. J. Contemp. Hosp. Manag. 17 (5) (2005) 402–408.
[10] P. Seung-Bae, N. Chung, S.-C. Woo, Do reward programs build loyalty to restaurants? The moderating effect of long-term orientation on the timing and types of rewards, J. Serv. Theory Pract. 23 (3) (2013) 225–244.
[11] M. De Veirman, V. Cauberghe, L. Hudders, Marketing through instagram influencers: the impact of number of followers and product divergence on brand attitude, Int. J. Advertising 36 (2017) 798–828.
[12] K. Salah, A. Alfalasi, M. Alfalasi, A blockchain-based system for online consumer reviews, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2019, pp. 853–858.
[13] D. Mayzlin, Y. Dover, J. Chevalier, Promotional reviews: An empirical investigation of online review manipulation, Amer. Econ. Rev. 104 (8) (2014) 2421–2455.
[14] P. Han, B. Xie, F. Yang, R. Shen, A scalable P2P recommender system based on distributed collaborative filtering, Expert Syst. Appl. 27 (2) (2004) 203–210.

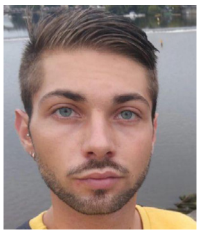
[15] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum Proj. Yellow Pap. 151 (2014) 1–32.
[16] A. Lisi, A. De Salve, P. Mori, L. Ricci, A smart contract based recommender system, in: International Conference on the Economics of Grids, Clouds, Systems, and Services, Springer, 2019, pp. 29–42.
[17] M.A. Khan, K. Salah, IoT security: Review, blockchain solutions, and open challenges, Future Gener. Comput. Syst. 82 (2018) 395–411.
[18] L. Chen, W.-K. Lee, C.-C. Chang, K.-K.R. Choo, N. Zhang, Blockchain based searchable encryption for electronic health record sharing, Future Gener. Comput. Syst. 95 (2019) 420–429.
[19] K. Wüst, A. Gervais, Do you need a blockchain?, in: 2018 Crypto Valley Conference on Blockchain Technology, CVCBT, IEEE, 2018, pp. 45–54.
[20] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2009, https://bitcoin.org/bitcoin.pdf.
[21] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9) (1997).
[22] N.M. Dwork C., Pricing via processing or combatting junk mail, in: Advances in Cryptology — CRYPTO' 92, Vol. 740, Springer, Berlin, Heidelberg, 1993, pp. 139–147.
[23] R. Dolan, J. Conduit, J. Fahy, S. Goodman, Social media engagement behaviour: A uses and gratifications perspective, J. Strateg. Mark. 24 (3–4) (2016) 261–277.
[24] S.K. Bista, S. Nepal, C. Paris, Engagement and cooperation in social networks: do benefits and rewards help?, in: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2012, pp. 1405–1410.
[25] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decis. Support Syst. 43 (2) (2007) 618–644, Emerging Issues in Collaborative Commerce.
[26] F. Hendrikx, K. Bubendorfer, R. Chard, Reputation systems: A survey and taxonomy, J. Parallel Distrib. Comput. 75 (2015) 184–197.
[27] L. Oliveira, L. Zavolokina, I. Bauer, G. Schwabe, To token or not to token: Tools for understanding blockchain tokens, in: Proceedings of the International Conference on Information Systems - Bridging the Internet of People, Data, and Things, ICIS 2018, San Francisco, CA, USA, December 13-16, 2018, Association for Information Systems, 2018.
[28] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey, Knowl.-Based Syst. 46 (2013) 109–132.
[29] W. Tay, X. Zhang, S. Karimi, Beyond mean rating: Probabilistic aggregation of star ratings based on helpfulness, J. Assoc. Inf. Sci. Technol. 71 (7) (2020) 784–799.
[30] Revain, https://revain.org/.
[31] M. McGlohon, N. Glance, Z. Reiter, Star quality: Aggregating reviews to rank products and merchants, in: Proceedings of the International AAAI Conference on Web and Social Media, Vol. 4, (1) 2010, pp. 114–121.
[32] H. Xie, J.C. Lui, Mathematical modeling and analysis of product rating with partial information, ACM Trans. Knowl. Discovery Data (TKDD) 9 (4) (2015) 1–33.
[33] Gastroadvisor whitepaper, https://www.gastroadvisor.com/whitepaper.pdf.
[34] Friendz whitepaper, https://friendz.io/assets/files/whitepaper_icofriendz.pdf.
[35] Lina Review, https://lina.network/lina-review/.
[36] M. Sharples, J. Domingue, The blockchain and kudos: A distributed system for educational record, reputation and reward, in: European Conference on Technology Enhanced Learning, Springer, 2016, pp. 490–496.
[37] F. Casino, C. Patsakis, An efficient blockchain-based privacy-preserving collaborative filtering architecture, IEEE Trans. Eng. Manage. 67 (4) (2020) 1501–1513.
[38] R. Dennis, G. Owen, Rep on the block: A next generation reputation system based on the blockchain, in: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), 2015, pp. 131–138.
[39] R.M. Frey, D. Wörner, A. Ilic, Collaborative filtering on the blockchain: A secure recommender system for e-commerce, in: 22nd Americas Conference on Information Systems, AMCIS 2016, San Diego, CA, USA, August 11-14, 2016.
[40] K.N. Khaqqi, J.J. Sikorski, K. Hadinoto, M. Kraft, Incorporating seller/buyer reputation-based system in blockchain-enabled emission trading application, Appl. Energy 209 (2018) 8–19.
[41] J. Eberhardt, S. Tai, On or off the blockchain? insights on off-chaining computation and data, in: European Conference on Service-Oriented and Cloud Computing, Springer, 2017, pp. 3–15.
[42] J. Benet, Ipfs-content addressed, versioned, p2p file system, 2014, arXiv preprint arXiv:1407.3561.
[43] P. Kaghazgaran, J. Caverlee, M. Alfifi, Behavioral analysis of review fraud: Linking malicious crowdsourcing to amazon and beyond, in: Eleventh International AAAI Conference on Web and Social Media, 2017.
[44] The Raiden network, https://raiden.network/.
[45] E. Ferrara, The history of digital spam, Commun. ACM 62 (8) (2019) 82–91.
[46] G. Noh, Y.-m. Kang, H. Oh, C.-k. Kim, Robust sybil attack defense with information level in online recommender systems, Expert Syst. Appl. 41 (4) (2014) 1781–1791.
[47] I. Gunes, C. Kaleli, A. Bilge, H. Polat, Shilling attacks against recommender systems: A comprehensive survey, Artif. Intell. Rev. 42 (4) (2014) 767–799.
[48] Y. Zhang, R. Deng, X. Liu, D. Zheng, Outsourcing service fair payment based on blockchain and its applications in cloud computing, IEEE Trans. Serv. Comput. (2018) 1, http://dx.doi.org/10.1109/TSC.2018.2864191.
[49] M. Herlihy, Atomic cross-chain swaps, in: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, 2018, pp. 245–254.
[50] S. Okazaki, L. Andreu, S. Campo, Knowledge sharing among tourists via social media: A comparison between facebook and tripadvisor, Int. J. Tourism Res. 19 (1) (2017) 107–119.
[51] D. Di Francesco Maesa, P. Mori, L. Ricci, A blockchain based approach for the definition of auditable access control systems, Comput. Secur. 84 (2019) 93–119.
[52] G. Brambilla, M. Amoretti, F. Zanichelli, Using blockchain for peer-to-peer proof-of-location, 2016, arXiv preprint arXiv:1607.00174.
[53] A. Abdul-Rahman, S. Hailes, A distributed trust model, in: Proceedings of the 1997 Workshop on New Security Paradigms, 1998, pp. 48–60.
[54] G. Caronni, Walking the web of trust, in: Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2000, IEEE, 2000, pp. 153–158.
[55] A. Tuzhilin, Towards the next generation of recommender systems, in: Proceedings of the 1st International Conference on E-Business Intelligence, ICEBI2010, Atlantis Press, 2010.
[56] V.A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, G.C. Polyzos, Interledger approaches, IEEE Access 7 (2019) 89948–89966.
[57] S. Schulte, M. Sigwart, P. Frauenthaler, M. Borkowski, Towards blockchain interoperability, in: International Conference on Business Process Management, Springer, 2019, pp. 3–10.
[58] T. Koens, E. Poll, Assessing interoperability solutions for distributed ledgers, Pervasive Mob. Comput. 59 (2019).

Andrea Lisi received his B.Sc. and M.Sc. degrees in computer science at the University of Pisa, where is also doing his PhD. His current area of interest concern blockchain technology, investigating on new domains of application and interoperability between different networks.

Andrea De Salve is a researcher at the Institute of Applied Sciences and Intelligent Systems (ISASI) of the National Research Council (CNR), Italy. He received a Ph.D. in Computer Science from the University of Pisa and he has 3 years experience as post-doc research at the Institute of Informatics and Telematics (IIT) of the National Research Council (CNR) and as a junior researcher at the Department of Matematics and Computer Science of the University of Palermo. His research area is related to the Complex network analysis, Web Mining, Distributed Ledger Technology, Distributed System, aspects of Security and Privacy of the Online Social Networks data.

Paolo Mori received the M.Sc. in Computer Science from the University of Pisa in 1998, and the Ph.D. from the same University in 2003. He is currently a researcher at Istituto di Informatica e Telematica of Consiglio Nazionale delle Ricerche. His main research interests involve trust, security and privacy in distributed environments focusing on the design and implementation of mechanisms for access/usage control, data privacy aspects of Online Social Networks, and Blockchain technology and its applications. He usually serves in the organization and Program Committees of international conference/workshops. In particular, he was Program Co-Chair of the 2nd-6th editions of the International Conference of Information System Security and Privacy (ICISSP 2016-2020). He is (co-)author of 100+ papers published on international journals and conference/workshop proceedings, and of several chapters in books on research topics. He is usually actively involved in EU funded and Italian research projects on information and communication security.

Laura Ricci received her Ph.D. from the University of Pisa. Currently, she is an Associate Professor at the Department of Computer Science, University of Pisa, Italy. She has been member of the committee for the definition of the Italian national strategy on blockchain. She is the coordinator of the local research unit of the European Project "Helios, A Context-Aware Social Networking Framework". Her research interests include distributed systems, peer-to-peer networks, distributed ledgers and cryptocurrencies. In this field, she has co-authored over 150 papers in refereed scientific journals and conference proceedings. She has served as a program committee member of several international conferences.



Samuel Fabrizi is currently a M.Sc. student in Artificial Intelligence at the University of Pisa. He received his B.Sc. from the same university in 2019 with a thesis on the application of Ethereum blockchain for recommender systems.