Users trust assessment based on their past behavior in large scale collaboration

Claudia-Lavinia Ignat
Université de Lorraine, CNRS, Inria
LORIA F-54000 Nancy, France
claudia.ignat@inria.fr

Quang-Vinh Dang
Industrial University of Ho Chi Minh City
Ho Chi Minh City, Vietnam
dangquangvinh@iuh.edu.vn

***Abstract*—In a large scale peer-to-peer collaboration where control over data is given to users who can decide with whom to share their data, a main challenge is how to compute trust in the collaborators. In this paper we show how to automatically compute users trust according to their past behavior during the collaboration in order to be able to predict their future behavior. We focus on two use cases: contract-based multi-synchronous collaboration and trust game from game theory. We show that computing trust from a single interaction between two users depends on the application domain, but that a general methodology for aggregating trust during the successive interactions between two users can be applied for both use cases.**

***Index Terms*—large scale peer-to-peer collaboration, trust, contracts, trust game**

I. Introduction

Most commonly used collaborative systems are those provided by large service providers such as Facebook and Google. While these collaborative services offer very interesting functionalities, they feature certain limitations. Most of the platforms hosting these collaboration services rely on a central authority and place personal information in the hands of a single large corporation which is a perceived privacy threat. Users must provide and store their data to vendors of these services and have to trust that they will preserve privacy of their data, but they have little control over the usage of their data after sharing it with other users. These systems do not scale well in terms of the number of users. Furthermore, user communities cannot deploy these kind of service applications since they generally rely on costly infrastructures rather than allowing sharing infrastructure and administration costs.

Our vision is to move away from centralized authority-based collaboration towards a large scale peer-to-peer (P2P) collaboration where control over data is given to users who can decide with whom to share their data. The risk of privacy breaches is decreased in this P2P collaboration as only part of the protected data is exposed at any time. The main strengths of P2P systems [24] are their independence of a centralized control and of a dedicated infrastructure. Participating nodes are owned and operated by independent individuals and therefore administration costs of the system are shared. Distinctive characteristics of P2P systems are high scalability, resilience to faults and attacks and a low deployment barrier for new services.

In this large scale peer-to-peer collaboration a main question is how to choose your collaborators that you can trust in order to share them your data. A rational decision would be based on the evaluation of previous collaborative behavior of your collaborators. In a large scale collaboration we cannot remember the interactions of all users with whom we collaborated and sometimes we have a false impression on the contribution of our collaborators. Some of them are modest and do not know to put themselves forward, while some others know to exaggerate their contribution. A trust value that is computed automatically based on previous collaboration behavior would be of great help to users. A main challenge in this task is how to compute this trust value according to past collaboration in order to be able to predict future user behavior.

In this paper we show how to assess users trust according to their behavior during collaboration in a large scale environment. We focus on two use cases: contract-based multi-synchronous collaboration and trust game from game theory. In what follows we explain our choice for these two use cases.

The multi-synchronous mode of collaboration [12] is the most general collaboration model, allowing to maintain multiple, simultaneous streams corresponding to multi-user activities, and then looking to manage divergence between these streams. In this collaboration mode users can work independently with different streams of activity on the shared data. These streams can diverge and hence users have different views of the shared data. Divergence can arise due to delays in the propagation of operations on the shared data or execution of conflicting operations. These divergent streams synchronise at a later time and hence re-establish a common view of the shared data. In a contract-based multi-synchronous model, contracts are specified by data owners when they share the data and the adherence to or violation of contracts can be checked after users gained access to data.

As an example of the contract-based multi-synchronous collaboration we can consider implicit contracts in distributed version control systems (DVCS) widely used for source code development. In open source projects, usage restrictions are expressed in the license of the code, while in closed source code projects, these restrictions are expressed in the contracts developers sign when accepting their job. In both cases, usage restrictions are checked *a posteriori* outside the collaborative environment with social control or plagiarism detection.

Audit of user compliance to the given contracts in the contract-based multi-synchronous collaboration would allow the computation of trust scores associated to users. This trust scores would help users collaborate with other users they trust. In the example of DVCS systems, as a result of observations concerning usage violation, trustworthiness on the users who misbehaved is implicitly decreased and collaboration with those users risks to be ceased.

Our second use case is the trust game [4], a money exchange game that has been widely used in behavioral economics for studying trust and collaboration between humans. In this game, exchange of money is entirely attributable to the existence of trust between users. In the context of the trust game we are interested in evaluating user trust that reflects user behavior during the game in terms of the sum of money exchanged.

In [28]–[31] we presented the model underlying a contract-based multi-synchronous collaboration and an auditing mechanism that can check user compliance to the given contracts. In this paper we propose a trust metric for the contract-based mult-synchronous collaboration that computes user trust levels according to the auditing results. In [10] we proposed a trust metric that computes user trust levels according to their past behavior in trust game. In this paper we study the similarities and differences in computing user trust in the two use cases, i.e. contract-based multi-synchronous collaboration and trust game. We show that computing trust from a single interaction between two users depends on the application domain, but that a general methodology for aggregating trust during the successive interactions between two users can be applied for both use cases.

The paper is organised as follows. In sections II and III we present our solution for assessing trust in a contract-based multi-synchronous collaboration and trust game respectively. Section IV discusses the general methodology for trust assessement in a collaboration involving a sequence of interactions among users and presents some principles for designing a trust function describing a single interaction among two users. Section V presents an overview of existing approaches for trust assessment in multi-synchronous collaboration and trust game. Finally, section VI presents some concluding remarks.

II. Trust Assessment in contract-based multi-synchronous collaboration

In a contract-based model each user maintains a local workspace that contains shared data and contracts for the usage restriction of that data as well as changes on data. Changes done locally on the data together with specified contracts are shared with other users.

Modifications that users do on the different parts of shared documents and contracts specified by users while they exchange different versions of the document with other users are kept in logs of operations maintaining information about who did the operation and when the operation was performed. These logs are updated at user sites when operations are generated locally or when they are received from other users. A replica log therefore contains all operations that have been generated locally or received from other users.

Events and log structures are defined as follows:

Definition 2.1 (Event): Let $\mathbb{P}$ be a set of operations $\{$*insert, delete, update, share*$\}$ that users can generate; and let $\mathbb{T}$ be a set of event types $\{$*write, communication, contract*$\}$. An event e is defined as a triplet of $\langle$*evt* $\in \mathbb{T}$, *op* $\in \mathbb{P}$, *attr*$\rangle$, in which *attr* includes attributes which are in form of $\{$*attr_name*, *attr_value*$\}$ to present additional information for each event.

Definition 2.2 (Log): A document log L is defined as an append-only ordered list of events in the form $[e_1, e_2, \ldots, e_n]$.

Events of type *write* can be operations of *insert*, *delete* and *update* on the document. The event corresponding to a *share* operation of type *communication* is issued when a user pushes his changes and it is logged at the site of receiver when this one performs a pull. This *share* event can be followed in the log by an event of type *contract* representing usage policies for the shared data.

A contract is a special type of log event being composed of a set of contract primitives. In what follows we provide the definition of contract primitive and of contract.

Definition 2.3 (Contract primitive): A contract primitive is composed of a deontic operator (P - the permitted, O - the obligatory, F - the forbidden) followed by a write or a communication operation in $\mathbb{P} = \{op_1, op_2, \ldots, op_n\}$. If op is an operation in $\mathbb{P}$ then the contract primitive c_{op} based on op is denoted as: F_{op} (doing op is forbidden), O_{op} (doing op is obligatory), and P_{op} (doing op is permitted).

Definition 2.4 (Contract): A contract C is a set of contract primitives which are built on these operations of $\mathbb{P}$. It is denoted as $C_{\mathbb{P}} = \{c_{op_1}, c_{op_2}, \ldots, c_{op_n}\}$. Alternatively, we can use the notation $C = \{c_{op_1}, c_{op_2}, \ldots, c_{op_n}\}$ for a contract.

Through a mechanism of log auditing we check whether user actions comply with contracts. Log auditing is an approach that adopts *a posteriori* enforcement of controlling compliance of users after the fact. The main principles of our auditing mechanism for the multi-synchronous collaboration are:

1) Users can automatically audit the log in order to make misbehaving users accountable for their actions without the need of any central authority. In this way the dependence on an online entity that provides auditing logs is overcome. However, the disadvantage of the mechanism is that users have no knowledge about global actions done by all other users in order to completely assess if a particular user behaved well or not. Our auditing mechanism is therefore based on incompleteness evidence. However, this assumption is suitable to human society where a person is assessed only based on some of her noticed behavior.
2) Logs that reflect actions done by users and that are input to the auditing mechanism must be maintained correctly. A solution based on authenticators for detecting log tampering can be adopted, log tampering being detected at time of synchronization before the log is accepted by receivers [32].

3) By means of an automatic log auditing users can discover other users that misbehaved. Their trust levels can be accordingly automatically updated.

We identified three types of attacks that might lead to contract violation.

- Malicious users tamper logs to eliminate or modify contracts or other events in the log. We consider that a user u is *malicious* if she re-orders, inserts or deletes events in the log. For instance, u removes some obligations that she does not want to fulfill. This kind of attack can be detected by a log authentication mechanism.
- Malicious users perform actions that are forbidden by the specified contracts. These action events are labelled as *bad*.
- Users neglect obligations that need to be fulfilled. For instance, a user receives an obligation *"insert is obligatory"* but she never fulfills this obligation. If at a given moment a log auditing mechanism is performed and no event that fulfills the obligation is found, we cannot claim that the user misbehaved. The user might fulfill the obligation at a later time. The given obligation is labelled as *unknown* meaning that the obligation has not yet been fulfilled. Once the obligation is fulfilled, the *unknown* label is removed.

Users are expected to respect given contracts. If a user respects all given contracts, then she will get a good trust value assessed by others. Ideally, if a user misbehaves in one of the three ways mentioned above, his misbehavior should be detected by other users. The auditing mechanism returns a trust value that is computed from the number of events labelled with *good*, *unknown* and *bad*.

Our auditing mechanism aims at finding contract violations and making users accountable for their irresponsible actions by adjusting their trust levels following a trust metric. The general idea of the auditing algorithm (Algorithm 1) automatically executed locally by each user is to browse their log and check each event appearing in the log whether it conforms to the given contracts. For each violation of a particular user found, we increase the number of *bad* events counted for the user. Similarly for each obligation that is not yet fulfilled, we increase the number of *unknown* events. The number of contract violations of user v over all the total audited events done by v is used to compute the trust level of v.

Algorithm 1 *audit* takes as input the local log L of user u and the position in the log *lastCheckedPos* identifying the last event checked by the auditing mechanism. L is browsed to check whether log events respect or not the given contracts. This corresponds to the case where the user u audits actions of all other users v who performed events in the log. Trust values of all audited users v in V are recomputed based on auditing results.

$contracts[v]$ and $obligations[v]$ are used to keep a set of contracts and a set of obligations which user v holds, respectively ($obligations[v] \in contracts[v]$).

For each event e in the log L, we check its event type, contract or write event. If e is a contract given to user v then it is added to $contracts[v]$. Moreover, if e is an obligation, it

Algorithm 1: audit($L, lastCheckedPos$)

```
1  for i = lastCheckedPos + 1 to length(L) do
2      e ← i^th event in L;
3      if (e.evt =' contract') then
4          v ← e.to;
5          if (v ∈ contracts) then
6              contracts[v] ← contracts[v] ∪ {e};
7          else
8              contracts[v] ← {e};
9          if e overrides c in contracts[v] then
10             contracts[v] ← contracts[v] \ {c};
11         if (e.attr.modal =' O') then
12             if (v ∈ obligations) then
13                 obligations[v] ← obligations[v] ∪ {e};
14             else
15                 obligations[v] ← {e};
16             if (v ∈ numberOfUnknownEvents) then
17                 numberOfUnknownEvents[v] ←
                    numberOfUnknownEvents[v] + 1;
18             else
19                 numberOfUnknownEvents[v] ← 1;
20     else
21         v ← e.by;
22         if e violates contracts[v] then
23             if (v ∈ numberOfBadEvents) then
24                 numberOfBadEvents[v] ←
                    numberOfBadEvents[v] + 1;
25             else
26                 numberOfBadEvents[v] ← 1;
27         if e fulfills c in obligations[v] then
28             obligations[v] ← obligations[v] \ {c};
29             numberOfUnknownEvents[v] − −;
30     V ← V ∪ {v};
31     if (v ∈ numberOfAuditedEvents) then
32         numberOfAuditedEvents[v] ←
            numberOfAuditedEvents[v] + 1;
33     else
34         numberOfAuditedEvents[v] ← 1;
35 foreach v in V do
36     current_trust[v] ←
        exp(−λ × (numberOfUnknownEvents[v] +
        k × numberOfBadEvents[v])/((1 + k) ×
        numberOfAuditedEvents[v]));
37     if (v ∈ trust) then
38         trust[v] ←
            α × current_trust[v] + (1 − α) × trust[v];
39     else
40         trust[v] ← current_trust[v];
```

is counted as *unknown* event until an event that fulfills it will be found. If e is a write or a communication event performed by user v, it is checked if it complies with or violates contracts in $contracts[v]$. For each user v, *numberOfBadEvents[v]* and *numberOfUnknownEvents[v]* are used to count the number of *bad* and *unknown* events that are audited, respectively (remaining events are considered *good*). $auditedEvents[v]$ is used to count the total number of audited events. All users v audited by u are inserted in set V.

Before the auditing mechanism is applied, several variables are initialised as shown in the procedure initialization. The set of audited users V is initialised to the empty set. *contracts* and *obligations* are represented as a *map* between users and a set containing log events (contracts and obligations respectively). *numberOfUnknownEvents*, *numberOfBadEvents*, *numberOfAuditedEvents*, *current_trust*, *trust* are represented as a *map* between users and integers.

Procedure initialization

1 $V \leftarrow new\ set()$;
2 $contracts \leftarrow new\ map()$;
3 $obligations \leftarrow new\ map()$;
4 $numberOfUnknownEvents \leftarrow new\ map()$;
5 $numberOfBadEvents \leftarrow new\ map()$;
6 $numberOfAuditedEvents \leftarrow new\ map()$;
7 $current\_trust \leftarrow new\ map()$;
8 $trust \leftarrow new\ map()$;

A user u can perform log auditing at any time at local site. As a result of log auditing, user u updates the trust values of users in the system. Log analysis has a time complexity $O(n)$ where n is the number of events that are audited. In case auditing creates significant overhead, users might skip auditing some parts of log with events performed by highly trusted users. However, in case these users behave badly, they are discovered only in a next auditing phase.

In order to manage trust levels, we need a decentralized trust model. The trust level of a user assessed by one another could be aggregated from log-based trust, reputation trust and recommendation trust. We propose an example of a trust metric, where the trust value is computed directly by users based on auditing of their local log and not from an indirect source such as a recommendation.

The current trust value for a user is computed based on auditing results consisting of the number of writing events that violate contracts defined as *bad*, the number of contract events that are not fulfilled defined as *unknown* since it is unknown whether they will be fulfilled in the future and the number of remaining events that are neither bad nor unknown, defined as *good*.

We consider trust values range in the interval [0..1]. If all audited events are good, then the trust value equals 1. Otherwise, the *unknown* and *bad* events decrease the trust value. We denote by x_1, x_2 and x_3 the number of *good*, *unknown* and *bad* events, respectively and by $y = x_1+x_2+x_3$ the total number of audited events. We assume the number of *bad* events, x_3, is k times stronger than x_2 in decreasing trust. Applying weighted mean of x_2 and x_3 to compute the decrement they cause on trust values over y audited events, we have:

$$malicious\_rate = \frac{\frac{x_2 + k \times x_3}{1+k}}{y} = \frac{x_2 + k \times x_3}{y + k \times y}$$

We wanted to define a trust function that varies according to the number of malicious events. In a system where violations are assumed happening rarely trust should be decreased quickly if violations are found, while in a system where violations are assumed easily happening trust should not be decreased strongly. In the multi-synchronous collaboration model based on contracts we assume that users perform actions mostly according to their given contracts, so violations will occur rarely. We considered that when bad events occur, trust value should decrease quickly. This assumption is realistic as in social life where one bad action might strongly decrease the reputation of a trusted person. Derived from this hypothesis, we designed the following trust function that decreases exponentially with the amount of bad events or unknown events found, λ being the decreasing coefficient, u being the auditor user and v the auditee user:

$$current\_trust(u,v) = e^{-\lambda \times \frac{x_2 + k \times x_3}{y + k \times y}}$$

Fig. 1 depicts an example of trust values computed from the following parameters. The total number of events is $y = 100$. The *bad* events are three times more dangerous than unknown event, $k = 3$. The decreasing rate for trust in case of malicious events is $\lambda = 5$. The graph plots function $e^{-5 \times \frac{x+3\times y}{400}}$. We can see that trust has the highest value 1 when no malicious events (bad and unknown) are found. Otherwise, trust decreases exponentially, however, it decreases more slowly on the dimension of *unknown* than the dimension of *bad*.

Trust value in user v assessed by user u is updated at each step n when an auditing is performed by u. Given the last trust value in user v assessed by user u, $trust_{n-1}(u,v)$, the new trust value in user v is computed by u as an aggregation between the current trust $current\_trust(u,v)$ and the last trust value $trust_{n-1}(u,v)$ where $n \geq 2$.

$$\begin{aligned} trust_n(u,v) =& \alpha \times current\_trust(u,v) \\ &+ (1-\alpha) \times trust_{n-1}(u,v) \end{aligned}$$

α stands for the importance of the current trust against the old trust value computed from the last auditing phase, where $0 < \alpha < 1$. $trust_1(u,v) = current\_trust(u,v)$ is the trust between u and v computed after the first auditing phase.

III. Trust Assessment in Trust Game

We reviewed game theory literature in the fields of cognitive science, psychology and economics and investigated whether

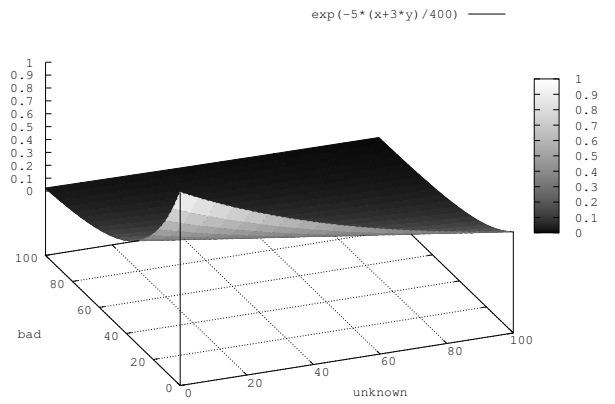

Fig. 1. Current trust computed from the variations of number of bad and unknown events that are found after auditing, $0 \leq x_2 \leq 100$, $0 \leq x_3 \leq 100$.

there is a game theory model that could reflect collaborative document sharing and that deals with user reputation and trust. The most appropriate game theory model is the trust game, also known as the investment game developed by Berg in 1995 [4], a money exchange game that is widely used in economics to study trust between users [20], [22]. In the traditional trust game, an investor (also called "sender" or "trustor") can invest a fraction of his money, and the broker (also called "receiver" or "trustee") can return only part of his gains. If both players follow their economics-based best interest, the investor should never invest and the broker should never re-pay anything. The observed money exchange is entirely attributable to the existence of trust. We illustrate next an example of the exchanges between the "sender" and the "receiver". Initially the sender sends an integer amount between 0 and 10 units to the receiver. The receiver gains three times the amount sent. For instance, if the sender sent 7 money units, the receiver will gain $3 \times 7 = 21$ units. Subsequently, the receiver can select an amount between 0 and the gained amount (in this case, 21) to return to the sender. However, the returned amount is not further multiplied. Suppose the receiver returned 11. The final payoff to the sender is 11 units, and the payoff to the receiver is $21 - 11 = 10$ units.

The trust game can be one-shot, i.e. the game ends after one round of money exchange, or repeated, i.e. it lasts several rounds [2], [7], [14]. The pairs of users could be fixed [9] or re-assigned before each round [13]. These games provide different kinds of partner information to players, such as their gender, age and income [26], or their past interaction history [5], [13].

We aimed to design a trust metric that computes partner trust as a basis for the prediction of partner behavior in the repeated trust game. Game theory predicts that, in trust game, sender will send 0 and receiver will send back 0 [8]. However, in experimental game theory we usually do not observe this user behavior as there is a trust built between the sender and receiver and if the sender sends some money, the receiver will return accordingly and so on.

The trust score function needs to satisfy the following requirements:

1) The trust value is higher if the sending amount is higher.
2) The trust value can distinguish between different types of users: honest and malicious ones.
3) The trust value considers user behavior over time.
4) The trust value encourages a stable behavior rather than a fluctuating one.
5) The trust value is robust against attacks.

A. Current trust

Separate trust scores are calculated for each player for each round, i.e. for each interaction between two players. The round number is denoted as t.

In each round, two users interact by sending a non-negative amount. For senders, the maximum amount they can send is set to 10, and for receivers, the maximum amount they can send is the amount they received from the sender (i.e. three times of what the sender sent). For both roles, we normalize the sending amount of both roles to $send\_proportion_t$ as the sending proportion of a user at round t, with $t \geq 1$:

$$send\_proportion_t = \frac{sending\_amount_t}{maximum\_sending\_amount_t}$$

It is obvious that $\forall t, 0 \leq send\_proportion_t \leq 1$.

We defined the trust metric for a single interaction between users as $current\_trust$. $current\_trust_t$ is a function of $send\_proportion_t$, meaning that the trustworthiness of a user in a single interaction depends on how much she sends to her partner in round t. $current\_trust_t$ should have a value between 0 and 1 inclusive. This function should satisfy the following properties (for convenience, we use the notation $f(x), f : [0,1] \rightarrow [0,1]$ for the function of $current\_trust_t$, with x being $send\_proportion_t$):

- $f(x)$ is continuous in $[0,1]$.
- $f(0) = 0$, meaning that $current\_trust$ is 0 if the user sends nothing.
- $f(1) = 1$, meaning that $current\_trust$ is 1 if the user sends the maximum possible amount.
- $f'(x) > 0$ with $x \in [0,1]$, meaning that $current\_trust$ is strictly increasing when $send\_proportion$ increases from 0 to 1. $f'(x)$ denotes the derivative of function $f(x)$.
- $f''(x) \leq 0$ with $x \in [0,1]$ meaning that the function is concave, i.e. the closer to 1 the value of $current\_trust$ is, the harder is to increment it.
- $f'(x^-) = f'(x^+), \forall x \in [0,1]$, meaning that the function is smooth, i.e. there is no reason that at some point the current trust increases roughly less than previously.

We proposed the following function that satisfies the above mentioned conditions:

$$current\_trust_t = \log(send\_proportion_t \times (e-1) + 1)$$

where $current\_trust_t$ is the $current\_trust$ function at round t and $send\_proportion_t$ is the value of $send\_proportion$ at round t.

B. Aggregated trust

We needed to calculate the aggregate trust score, which is the cumulative trust score over multiple rounds. Similar to trust computation in contract-based multi-synchronous collaboration, we defined the aggregate trust score function as an exponential averaging update function over the previous interactions between the two users that gives more weighting to the current computed trust than the simple average does.

$$aggregate\_trust_t = \alpha_t \times current\_trust_t + (1 - \alpha_t) \times aggregate\_trust_{t-1}$$

The requirement for our aggregate function was that it has to encourage a stable behavior rather than a fluctuating one. Our aggregate function was inspired by the trust model SecuredTrust [11] for computing trust and reputation of interacting agents in a multi-agent system in the presence of highly oscillating malicious behavior. The weight α_t in the aggregate trust score function has to change based on the accumulated deviation β_t.

$$\alpha_t = threshold + \frac{c \times \delta_t}{1 + \beta_t}$$
$$\delta_t = |current\_trust_t - current\_trust_{t-1}|$$
$$\beta_t = c \times \delta_t + (1 - c) \times \beta_{t-1}$$

The δ_t is the change of current trust value by two sequential interactions t and $t-1$ between two users, where $current\_trust_0 = 0$. We calculated δ_t to see how much a person changes her behavior since her last interaction.

$\beta_0 = 0$. c is a constant that controls to what extent we react to the recent change of the current trust. A higher value of c gives more significance to the recent change of the current trust (δ_t) than to the accumulated deviation β_{t-1}. We set $c = 0.9$.

It is easy to prove that α_t is bigger if δ_t is bigger, and vice versa. It means that, if the trust computed from the current interaction is much different from accumulated trust of all previous interactions, the current interaction will play a more important role in the final trust value.

The $threshold$ is used to prevent α_t from saturating to a fixed value. $threshold$ was set to 0.25.

We showed that our trust metric can predict the future behavior of users by analyzing its performance on several data sets including [13], [6] and [19] where the data is provided under the form of a behavior log of participants. The total data set comprised behavior of 174 participants in repeated trust games. Based on the behavior log we applied our trust metric on users behavior at a certain round, then used the output trust score as the independent variable to predict the users behavior in the next round. For all rounds starting with round five, we found a high correlation between the output trust scores and user behavior in the next round. The results of the linear regression between our trust metric and future users behavior at rounds five and ten are presented in Table I. For the dataset in [6], because of the design of the experiment, we could only test the relationship between our trust metric and user behavior at round five. We considered the independent variable as being the trust value our metric assigned to each user before a particular round, and the dependent variable as the behavior of this user in this round. We can notice that the slopes of all regressions are significant, meaning that our trust metric predicts well users behavior.

TABLE I
REGRESSION BETWEEN TRUST METRIC AND FUTURE USERS BEHAVIOR.

	Intercept	**Slope**	**Adj.R^2**
Dataset [19] (round 5)	0.071	0.701***	0.319
Dataset [19] (round 10)	-0.022	0.913***	0.542
Dataset [6] (round 5)	-0.006	0.715***	0.362
Dataset [13] (round 5)	0.072	0.848***	0.356
Dataset [13] (round 10)	0.027	0.855***	0.357

We denote '***' as significant level of 99.9%.

IV. DISCUSSION

A collaborative system can automatically assess trust of a user u in another user v according to the past behavior of v during its interactions with u. We proposed a general method on how to assess the trust in user v according to his/her behavior during n successive collaborations with u. We proposed that the trust value in user v assessed by user u is updated after each interaction between the two users according to the current interaction, but also taking into account previous interactions between the two users. The values of trust computed during previous interactions between two users are aggregated with the trust value of the current interaction as an exponential averaging update function as follows:

$$trust_n(u, v) = \alpha \times current\_trust(u, v) + (1 - \alpha) \times trust_{n-1}(u, v)$$

α ($0 < \alpha < 1$) is the parameter standing for the importance of the current trust against the old trust value computed from the previous interactions. α can be a fixed constant or can be further refined in order to deal with fluctuations of behavior of user v during his/her successive interactions with user u as we proposed for the trust game collaboration use case.

The trust value $current\_trust(u, v)$ from the current interaction between u and v has to be defined according to the specific application domain. In our first use case, the contract-based multi-synchronous collaboration, the assumption was that the majority of people behaves correctly so that the trust value decreases each time a user misbehaves, i.e. when *unknown* or *bad* events are audited. The trust function $current\_trust$ is therefore a decreasing function in terms of the number of malicious events. In our second use case, the trust-based game, we wanted that the trust value in a user should depend on the amount that user sends to the game partner. The trust function $current\_trust$ is therefore depending on $send\_proportion$, i.e. the ratio over the amount

a user sends to the partner and the maximum amount that user could have sent to the partner.

V. Related Work

In this section we provide a short overview of trust management in the fields of multi-synchronous collaboration and trust game.

In a distributed collaboration model where access is given first to data without control but with restrictions that are verified a posteriori, trust management is an important aspect. The concept of trust in different communities varies according to how it is computed and used. We rely on the concept of trust which is based on past user behaviors [23]. Trust is not immutable and it changes over time. Thus trust should be managed by using a trust model. A trust model includes three basic components [25] that are gathering behavioral information, scoring and ranking peers and rewarding or punishing peers. Most of existing P2P trust models (e.g. EigenTrust model [21]) propose mechanisms to update trust values based on direct interactions between peers. In a collaboration model where restrictions are verified a posteriori, log auditing helps one user evaluate others either through direct or indirect interactions. No existing trust model considers log auditing result for trust assessment.

Several approaches on predicting behavior of participants in one-shot trust game were proposed using additional information related to players, such as their personal information (e.g. age, gender, income) [16], [34] or evaluation collected through some tests or questionnaires users needed to fill in before the experiment [3], [15], [18], [33]. Sociometric information of users is usually not available [27] and when available it is not always reliable as users can declare false personal information. Our method to model and predict users' behavior does not require any additional personal data and relies uniquely on past user behavior during the game.

User trust in repeated trust game was measured as an average value of previous sending amount [1], [13], [17], [20]. However, this average trust metric can not deal with malicious user fluctuating behavior where users may strategically oscillate between collaboration periods when they aim gaining the trust of the other users and sudden betrayal. Our metric for repeated trust game punishes user fluctuating behavior.

VI. Conclusions

In this paper we showed how to automatically assess trust in users according to their interactions during the collaboration. We showed how trust can be computed in: (1) a contract-based multi-synchronous collaboration where contracts are specified by data owners when they share the data and the adherence to or violation of contracts can be checked after users gained access to data and (2) trust game where trust between users can be computed according to the exchange of money. A general methodology was proposed for both use cases for aggregating trust during the successive interactions between two users. However, computing trust from a single interaction between two users depends on the application domain and the associated semantics and we proposed trust functions for each of the use cases. In future work we plan to apply the same methodology for assessing trust in Wikipedia contributors according to the quality of their contributions to various articles.

VII. Acknowledgements

We thank Hien Thi Thu Truong for her preliminary work on the trust metric in contract-based multi-synchronous collaboration.

References

[1] Steffen Altmann, Thomas Dohmen, and Matthias Wibral. Do the reciprocal trust less? *Economics Letters*, 99(3):454–457, 2008.

[2] Vital Anderhub, Dirk Engelmann, and Werner Güth. An experimental study of the repeated trust game with incomplete information. *Journal of Economic Behavior & Organization*, 48(2):197–216, 2002.

[3] Nava Ashraf, Iris Bohnet, and Nikita Piankov. Decomposing trust and trustworthiness. *Experimental Economics*, 9(3):193–208, 2006.

[4] Joyce Berg, John Dickhaut, and Kevin McCabe. Trust, reciprocity, and social history. *Games and economic behavior*, 10(1):122–142, 1995.

[5] Gary E. Bolton, Elena Katok, and Axel Ockenfels. Cooperation among strangers with limited information about reputation. *Journal of Public Economics*, 89(8):1457–1468, 2005.

[6] Giangiacomo Bravo, Flaminio Squazzoni, and Riccardo Boero. Trust and partner selection in social networks: An experimentally grounded model. *Social Networks*, 34(4):481–492, 2012.

[7] Stephen V Burks, Jeffrey P Carpenter, and Eric Verhoogen. Playing both roles in the trust game. *Journal of Economic Behavior & Organization*, 51(2):195–216, 2003.

[8] Colin Camerer. *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press, 2003.

[9] Francois Cochard, Phu Nguyen Van, and Marc Willinger. Trusting behavior in a repeated investment game. *Journal of Economic Behavior & Organization*, 55(1):31–44, 2004.

[10] Quang Vinh Dang and Claudia-Lavinia Ignat. Computational Trust Model for Repeated Trust Games. In *Proceedings of the 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications - TrustCom 2016*, pages 34–41, Tianjin, China, August 2016.

[11] Anupam Das and Mohammad Mahfuzul Islam. SecuredTrust: A dynamic trust computation model for secured communication in multiagent systems. *IEEE Transactions on Dependable and Secure Computing*, 9(2):261–274, 2012.

[12] Paul Dourish. The parting of the ways: divergence, data management and collaborative work. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'95, pages 215–230, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[13] Dimitri Dubois, Marc Willinger, and Thierry Blayac. Does players' identification affect trust and reciprocity in the lab? *Journal of Economic Psychology*, 33(1):303–317, 2012.

[14] Jim Engle-Warnick and Robert L. Slonim. Learning to trust in indefinitely repeated games. *Games and Economic Behavior*, 54(1):95–114, 2006.

[15] Anthony M Evans and William Revelle. Survey and behavioral measurements of interpersonal trust. *Journal of Research in Personality*, 42(6):1585–1593, 2008.

[16] Armin Falk, Stephan Meier, and Christian Zehnder. Did we overestimate the role of social preferences? The case of self-selected student samples. *CESifo Working Paper No. 3177*, 2010.

[17] Edward L. Glaeser, David I. Laibson, Jose A. Scheinkman, and Christine L. Soutter. Measuring trust. *Quarterly Journal of Economics*, pages 811–846, 2000.

[18] Anna Gunnthorsdottir, Kevin McCabe, and Vernon Smith. Using the machiavellianism instrument to predict trustworthiness in a bargaining game. *Journal of Economic Psychology*, 23(1):49–66, 2002.

[19] Claudia-Lavinia Ignat, Quang-Vinh Dang, and Valerie L. Shalin. The influence of trust score on cooperative behavior. *ACM Transactions on Internet Technology*, 19(4):46:1–46:22, September 2019.

[20] Noel D Johnson and Alexandra A Mislin. Trust games: A meta-analysis. *Journal of Economic Psychology*, 32(5):865–889, 2011.

[21] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web, WWW 2003*, pages 640–651, Budapest, Hungary, May 2003. ACM Press.

[22] Roy J. Lewicki and Chad Brinsfield. Trust research: measuring trust beliefs and behaviours. In *Handbook of research methods on trust*, pages 46–64. Edward Elgar Publishing, 2015.

[23] Mui Lik, Mohtashemi Mojdeh, and Ari Halberstadt. A Computational Model of Trust and Reputation. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, HICSS 2002*, pages 2431–2439, Waikoloa, Big Island, Hawaii, January 2002. IEEE Computer Society.

[24] Rodrigo Rodrigues and Peter Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, October 2010.

[25] Marti Sergio and Garcia-Molina Hector. Taxonomy of Trust: Categorizing P2P Reputation Systems. *Computer Networks*, 50:472–484, March 2006.

[26] Robert Slonim and Ellen Garbarino. Increases in trust and altruism from partner selection: Experimental evidence. *Experimental Economics*, 11(2):134–153, 2008.

[27] Jiliang Tang, Huiji Gao, Huan Liu, and Atish Das Sarma. eTrust: understanding trust evolution in an online world. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 253–261. ACM, 2012.

[28] Hien Thi Thu Truong. *A Contract-based and Trust-aware Collaboration Model. (Un modèle de collaboration basé sur les contrats et la confiance)*. PhD thesis, University of Lorraine, Nancy, France, 2012.

[29] Hien Thi Thu Truong and Claudia-Lavinia Ignat. Log Auditing for Trust Assessment in Peer-to-Peer Collaboration. In *Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC 2011)*, pages 207–214, Cluj-Napoca, Romania, July 2011.

[30] Hien Thi Thu Truong, Claudia-Lavinia Ignat, Mohamed-Rafik Bouguelia, and Pascal Molli. A contract-extended push-pull-clone model. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2011)*, pages 211–220, Orlando, Florida, USA, October 2011.

[31] Hien Thi Thu Truong, Claudia-Lavinia Ignat, and Pascal Molli. A contract-extended push-pull-clone model for multi-synchronous collaboration. *Journal of Cooperative Information Systems*, 21(03):221–262, September 2012.

[32] Hien Thi Thu Truong, Claudia-Lavinia Ignat, and Pascal Molli. Authenticating Operation-based History in Collaborative Systems. In *Proceedings of the ACM International Conference on Supporting Group Work (Group 2012)*, pages 131–140, Sanibel Island, Florida, USA, October 2012. ACM.

[33] Toshio Yamagishi, Satoshi Akutsu, Kisuk Cho, Yumi Inoue, Yang Li, and Yoshie Matsumoto. Two-component model of general trust: Predicting behavioral trust from attitudinal trust. *Social Cognition*, 33(5):436, 2015.

[34] Steven T Yen. An econometric analysis of household donations in the USA. *Applied Economics Letters*, 9(13):837–841, 2002.