Users trust assessment based on their past behavior in large scale collaboration
Claudia-Lavinia Ignat, Quang-Vinh Dang

To cite this version:
Claudia-Lavinia Ignat, Quang-Vinh Dang. Users trust assessment based on their past behavior in large scale collaboration. ICCP 2021 - IEEE 17th International Conference on Intelligent Computer Communication and Processing, Oct 2021, Cluj-Napoca/Online, Romania. hal-03469344

HAL Id: hal-03469344
https://inria.hal.science/hal-03469344
Submitted on 7 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Users trust assessment based on their past behavior in large scale collaboration

Claudia-Lavinia Ignat  
Univrsit 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@iu.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 multi-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 assessment 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 \( \mathbb{E} \) be a set of operations \{insert, delete, update, share\} that users can generate, and let \( \mathbb{P} \) be a set of event types \{write, communication, contract\}. An event \( e \) is defined as a triplet of \( (\text{evt} \in \mathbb{E}, \text{op} \in \mathbb{P}, \text{attr}) \), in which \( \text{attr} \) includes attributes which are in form of \( \langle \text{attr\_name}, \text{attr\_value} \rangle \) 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 \( \mathbb{P} = \{\text{op}_1, \text{op}_2, \ldots, \text{op}_n\} \). If \( \text{op} \) is an operation in \( \mathbb{P} \) then the contract primitive \( c_{\text{op}} \) based on \( \text{op} \) is denoted as: \( F_{\text{op}} \) (doing \( \text{op} \) is forbidden), \( O_{\text{op}} \) (doing \( \text{op} \) is obligatory), and \( P_{\text{op}} \) (doing \( \text{op} \) 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 \( \mathbb{C}_P = \{c_{\text{op}_1}, c_{\text{op}_2}, \ldots, c_{\text{op}_n}\} \). Alternatively, we can use the notation \( \mathbb{C} = \{c_{\text{op}_1}, c_{\text{op}_2}, \ldots, c_{\text{op}_n}\} \) 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.
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.

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

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

![Algorithm 1: audit(L, lastCheckedPos)](image-url)

it is added to \(\text{contracts}[v] \). Moreover, if \( e \) is an obligation, it
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. \( V \leftarrow \text{new set}(); \)
2. contracts \( \leftarrow \text{new map}(); \)
3. obligations \( \leftarrow \text{new map}(); \)
4. \( numberOfUnknownEvents \leftarrow \text{new map}(); \)
5. \( numberOfBadEvents \leftarrow \text{new map}(); \)
6. \( numberOfAuditedEvents \leftarrow \text{new map}(); \)
7. current trust \( \leftarrow \text{new map}(); \)
8. trust \( \leftarrow \text{new map}(); \)

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:

\[
\text{malicious rate} = \frac{x_2 + k \times x_3}{y} = \frac{x_2 + k \times x_3}{y+k \times y}
\]

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, \( \lambda \) being the decreasing coefficient, \( u \) being the auditor user and \( v \) the auditee user:

\[
\text{current trust}(u,v) = e^{-\lambda \times \frac{x_2 + k \times x_3}{y+k \times y}}
\]

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 \( e^{-5 \times \frac{x_2 + 3x_3}{100+3x}} \). 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 \( u \), \( \text{trust}_{n-1}(u,v) \), the new trust value in user \( v \) is computed by \( u \) as an aggregation between the current trust \( \text{current trust}(u,v) \) and the last trust value \( \text{trust}_{n-1}(u,v) \) where \( n \geq 2 \).

\[
\text{trust}_n(u,v) = \alpha \times \text{current trust}(u,v) + (1 - \alpha) \times \text{trust}_{n-1}(u,v)
\]

\( \alpha \) stands for the importance of the current trust against the old trust value computed from the last auditing phase, where \( 0 < \alpha < 1 \). \text{trust}_1(u,v) = \text{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
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 × 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 $\text{send}_{\text{proportion}_{t}}$ as the
sending proportion of a user at round $t$, with $t \geq 1$:

$$\text{send}_{\text{proportion}_{t}} = \frac{\text{sending}_{\text{amount}}_{t}}{\text{maximum}_{\text{sending}_{\text{amount}}_{t}}}$$

It is obvious that $\forall t, 0 \leq \text{send}_{\text{proportion}_{t}} \leq 1$.

We defined the trust metric for a single interaction be-
 tween users as current\_trust, current\_trust$_t$ is a function of
$\text{send}_{\text{proportion}_{t}}$, 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 $\text{send}_{\text{proportion}_{t}}$):

- $f(x)$ 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 $\text{send}_{\text{proportion}_{t}}$ 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:

$$\text{current\_trust}_{t} = \log(\text{send}_{\text{proportion}_{t}} \times (e - 1) + 1)$$

where current\_trust$_t$ is the current\_trust function at round $t$
and $\text{send}_{\text{proportion}_{t}}$ is the value of $\text{send}_{\text{proportion}_{t}}$ at
round $t$. 

![Fig. 1. Current trust computed from the variations of number of bad and unknown events that are found after auditing, 0 ≤ x_2 ≤ 100, 0 ≤ x_3 ≤ 100.](image-url)
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.

\[
    \text{aggregate\_trust}_t = \alpha_t \times \text{current\_trust}_t + (1 - \alpha_t) \times \text{aggregate\_trust}_{t-1}
\]

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 \( \alpha_t \) in the aggregate trust score function has to change based on the accumulated deviation \( \beta_t \).

\[
    \alpha_t = \text{threshold} + \frac{c \times \delta_t}{1 + \beta_t}
\]

\[
    \delta_t = |\text{current\_trust}_t - \text{current\_trust}_{t-1}|
\]

\[
    \beta_t = c \times \delta_t + (1 - c) \times \beta_{t-1}
\]

The \( \delta_t \) is the change of current trust value by two sequential interactions between two users, where \( \text{current\_trust}_0 = 0 \). We calculated \( \delta_t \) 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 (\( \delta_t \)) than to the accumulated deviation \( \beta_{t-1} \). We set \( c = 0.9 \).

It is easy to prove that \( \alpha_t \) is bigger if \( \delta_t \) 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 \text{threshold} is used to prevent \( \alpha_t \) from saturating to a fixed value. \text{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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(round 5)</th>
<th>(round 10)</th>
<th>(round 5)</th>
<th>(round 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[19]</td>
<td>0.071</td>
<td>0.701***</td>
<td>0.357</td>
<td>0.319</td>
</tr>
<tr>
<td>[19]</td>
<td>-0.022</td>
<td>0.913***</td>
<td>0.542</td>
<td>0.542</td>
</tr>
<tr>
<td>[16]</td>
<td>0.006</td>
<td>0.715***</td>
<td>0.362</td>
<td>0.362</td>
</tr>
<tr>
<td>[13]</td>
<td>0.072</td>
<td>0.848***</td>
<td>0.356</td>
<td>0.356</td>
</tr>
<tr>
<td>[13]</td>
<td>0.027</td>
<td>0.855***</td>
<td>0.357</td>
<td>0.357</td>
</tr>
</tbody>
</table>

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

IV. DISCUSSION

A collaborative system can automatically assess trust of a user in another user’s behavior according to the past behavior or during its interactions with this user. We proposed a general method on how to assess the trust in user’s 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:

\[
    \text{trust}_n(u, v) = \alpha \times \text{current\_trust}(u, v) + (1 - \alpha) \times \text{trust}_{n-1}(u, v)
\]

\( \alpha \) (0 < \( \alpha \) < 1) is the parameter standing for the importance of the current trust against the old trust value computed from the previous interactions. \( \alpha \) 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 \( \text{current\_trust}(u, v) \) 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 \( \text{current\_trust} \) is therefore a decreasing function in terms of the number of malicious events. In our second use case, the trust-based games, we wanted that the trust value in a user should depend on the amount that user sends to the game partner. The trust function \( \text{current\_trust} \) is therefore depending on \( \text{send\_proportion} \), 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 [2] that are gathering behavioral information, scoring and ranking peers and reward 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 [8], [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


