Plausibility-based Trust Management Mechanism for Cloud Services

Saikumar.Divi[1], G.V.Manikanth[2]

[1]*Student, CSE Department, Prakasam Engineering College, AP, India,* ***sai.dv567@gmail.com***
[2]*Assistant Professor, CSE Department, Prakasam Engineering College, AP, India,* ***gmkmam@gmail.com***

Abstract

It is one of the most challenging issues for the trust and growth of cloud computing. Several challenging issues such as privacy, security, and availability etc are introduced. These are highly dynamic, distributed, and non-transparent nature of cloud services. It is not an easy task of preserving consumers' privacy due to the sensitive information involved in the interactions between consumers and the trust management service. Protecting cloud services against their malicious users (e.g., such users might give misleading feedback to disadvantage a particular cloud service) is a difficult problem. Guaranteeing the availability of the trust management service is another significant challenge because of the dynamic nature of cloud environments. In this article, we describe the design and implementation of a Mechanism a reputation-based trust management framework that provides a set of functionalities to deliver Trust as a Service (TaaS), which includes i) a novel protocol to prove the credibility of trust feedbacks and preserve users' privacy, ii) an adaptive and robust credibility model for measuring the credibility of trust feedbacks to protect cloud services from malicious users and to compare the trustworthiness of cloud services, and iii) an availability model to manage the availability of the decentralized implementation of the trust management service. The feasibility and benefits of our approach have been validated by a prototype and experimental studies using a collection of real-world trust feedbacks on cloud services.

Index Terms: *Plausibility, Cloud computing, credibility, service, security, privacy, availability.*

1. INTRODUCTION

Consumers' feedback is a good source to assess the overall trustworthiness of cloud services. Several researchers have recognized the significance of trust management and proposed solutions to assess and manage trust based on feedbacks collected from participants. In reality, it is not unusual that a cloud service experiences malicious behaviors (e.g., collusion or Sybil attacks) from its users. This paper focuses on improving trust management in cloud environments by proposing novel ways to ensure the credibility of trust feedbacks. In particular, we distinguish the following key issues of the trust management in cloud environments:

Consumers' Privacy. The adoption of cloud computing raise privacy concerns. Consumers can have dynamic interactions with cloud providers, which may involve sensitive information. There are several cases of privacy breaches such as leaks of sensitive information (e.g., date of birth and address) or behavioral information (e.g., with whom the consumer interacted, the kind of cloud services the consumer showed interest, etc.). Undoubtedly, services which involve consumers' data (e.g., interaction histories) should preserve their privacy.

Cloud Services Protection. It is not unusual that a cloud service experiences attacks from its users. Attackers can disadvantage a cloud service by giving multiple misleading feedbacks (i.e., collusion attacks) or by creating several accounts (i.e., Sybil attacks). Indeed, the detection of such malicious behaviors poses several challenges. Firstly, new users join the cloud environment and old users leave around the clock. This consumer dynamism makes the detection of malicious behaviors (e.g., feedback collusion) a significant challenge. Secondly, users may have multiple accounts for a particular cloud service, which makes it difficult to detect Sybil attacks. Finally, it is difficult to predict when malicious behaviors occur (i.e., strategic VS. occasional behaviors).

Trust Management Service's Availability. A trust management service (TMS) provides an interface between users and cloud services for effective trust management. However, guaranteeing the availability of TMS is a difficult problem due to the un-predictable number of users and the highly dynamic nature of the cloud environment. Approaches that require understanding of users' interests and capabilities through similarity measurements or operational availability measurements (i.e., uptime to the total time) are inappropriate in cloud environments. TMS should be adaptive and highly scalable to be functional in cloud environments.

In this paper, we overview the design and the implementation of mechanism : a framework for plausibility-based trust management in cloud environments. In this, trust is delivered as a service (TaaS) where TMS spans several distributed nodes to manage feedbacks in a decentralized way. This mechanism exploits techniques to identify credible feedbacks from malicious ones. In a nutshell, the salient features of the mechanism are:

- *Zero-Knowledge Credibility Proof Protocol (ZKC2P).* We introduce ZKC2P that not only preserves the consumers' privacy, but also enables the TMS to prove the credibility of a particular consumer's feedback. We propose that the Identity Management Service (IdM)

can help TMS in measuring the credibility of trust feedbacks without breaching consumers' privacy. Anonymization techniques are exploited to protect users from privacy breaches in users' identity or interactions.

- *A Credibility Model.* The credibility of feedbacks plays an important role in the trust management service's performance. Therefore, we propose several metrics for the feedback collusion detection including the *Feedback Density* and *Occasional Feedback Collusion*. These metrics distinguish misleading feedbacks from malicious users. It also has the ability to detect strategic and occasional behaviors of collusion attacks (i.e., attackers who intend to manipulate the trust results by giving multiple trust feedbacks to a certain cloud service in a long or short period of time). In addition, we propose several metrics for the Sybil attacks detection including the *Multi-Identity Recognition* and *Occasional Sybil Attacks*. These metrics allow TMS to identify misleading feedbacks from Sybil attacks.

- *An Availability Model.* High availability is an important requirement to the trust management service. Thus, we propose to spread several distributed nodes to manage feedbacks given by users in a decentralized way. Load balancing techniques are exploited to share the workload, thereby always maintaining a desired availability level. The number of TMS nodes is determined through an *operational power* metric. Replication techniques are exploited to minimize the impact of crashing TMS instances. The number of replicas for each node is determined through a *replication determination* metric that we introduce. This metric exploits particle filtering techniques to precisely predict the availability of each node.

The remainder of the paper is organized as follows. Section 2 briefly presents the design of the framework. Section 3 introduces the design of the Zero- Knowledge Credibility Proof Protocol, assumptions and attack models. Section 4 and Section 5 describe the details of our credibility model and availability model respectively. Section 6 reports the implementation and the results of experimental evaluations. Finally, Section 7 overviews the related work and Section 8 provides some concluding remarks.

2 THE FRAMEWORK

The framework is based on the service oriented architecture (SOA), which delivers trust as a service. SOA and Web services are one of the most important enabling technologies for cloud computing in the sense that resources (e.g., infrastructures, platforms, and software) are exposed in clouds as services. In particular, the trust management service spans several distributed nodes that expose interfaces so that users can give their feedbacks or inquire the trust results. Figure 1 depicts the framework, which consists of three different layers, namely the *Cloud Service Provider Layer*, the *Trust Management Service Layer*, and the *Cloud Service Consumer Layer*.

The Cloud Service Provider Layer. This layer consists of different cloud service providers who offer one or several cloud services, i.e., IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service), publicly on the Web. These cloud services are accessible through Web portals and indexed on Web search engines such as Google, Yahoo, and Baidu. Interactions for this layer are considered as *cloud service interaction* with users and TMS, and *cloud services advertisements* where providers are able to advertise their services on the Web.

The Trust Management Service Layer. This layer consists of several distributed TMS nodes which are hosted in multiple cloud environments in different geographical areas. These TMS nodes expose interfaces so that users can give their feedback or inquire the trust results in a decentralized way. Interactions for this layer include: i) *cloud service interaction* with cloud service providers, ii) *service advertisement* to advertise the trust as a service to users through the Internet, iii) *cloud service discovery* through the Internet to allow users to assess the trust of new cloud services, and iv) *Zero-Knowledge Credibility Proof Protocol* (ZKC2P) interactions enabling TMS to prove the credibility of a particular consumer's feed- back.

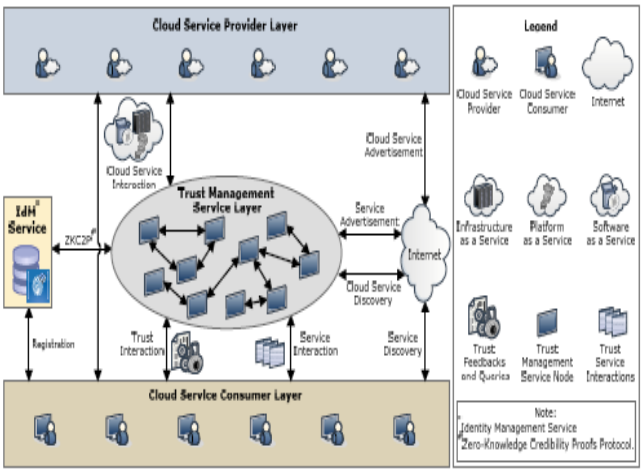

Fig. 1. Architecture of the Trust Management Framework

The Cloud Service Consumer Layer. Finally, this layer consists of different users who use cloud services. For example, a new startup that has limited funding can consume cloud services (e.g., hosting their services in Amazon S3). Interactions for this layer include: i) *service discovery* where users are able to discover new cloud services and other services through the Internet, ii) *trust* and *service* interactions where users are able to give their feedback or retrieve the trust results of a particular cloud service, and iii) *registration* where users establish their identity through registering their credentials in IdM before using TMS.

Our framework also exploits a Web crawling approach for automatic cloud services discovery, where cloud services are automatically discovered on the Internet and stored in a *cloud services repository*. Moreover, our framework contains an *Identity Management Service* (see Figure 1) which is responsible for the *registration* where users

register their credentials before using TMS and proving the credibility of a particular consumer's feedback through ZKC2P.

3 ZERO-KNOWLEDGE CREDIBILITY PROOF PROTOCOL (ZKC2P)

Since there is a strong relation between trust and identification as emphasized in, we propose to use the *Identity Management Service* (IdM) helping TMS in measuring the credibility of a consumer's feedback. However, processing the IdM information can breach the privacy of users. One way to preserve privacy is to use cryptographic encryption techniques. However, there is no efficient way to process encrypted data. Another way is to use anonymization techniques to process the IdM information without breaching the privacy of users. Clearly, there is a trade-off between high anonymity and utility. Full anonymization means better privacy, while full utility results in no privacy protection (e.g., using a de-identification anonymization technique can still leak sensitive information through linking attacks.

Thus, we propose a *Zero-Knowledge Credibility Proof Protocol* (ZKC2P) to allow TMS to process IdM's information (i.e., credentials) using the *Multi-Identity Recognition* factor. In other words, TMS will prove the users' feedback credibility without knowing the users' credentials. TMS processes credentials without including the sensitive information. Instead, anonymized information is used via consistent hashing (e.g., sha-256). The anonymization process covers all the credentials' attributes except the *Timestamps* attribute.

3.1. Identity Management Service (IdM)

Since trust and identification are closely related, as highlighted by David and Jaquet in [20], we believe that IdM can facilitate TMS in the detection of Sybil attacks against cloud services without breaching the privacy of users. When users attempt to use TMS for the first time, TMS requires them to register their credentials at the trust identity registry in IdM to establish their identities.

The trust identity registry stores an identity record represented by a tuple $I = (C, C_a, T_i)$ for each user. C is the user's primary identity (e.g., user name). C_a represents a set of credentials' attributes (e.g., passwords, postal address, and IP address) and T_i represents the user's registration time in TMS.

3.2. Trust Management Service (TMS)

In a typical interaction of the reputation-based TMS, a user either gives feedback regarding the trustworthiness of a particular cloud service or requests the trust assessment of the service1. From users' feedback, the trust behavior of a cloud service is actually a collection of invocation history records, represented by a tuple $H = (C, S, F, Tf)$, where C is the user's primary identity, S is the cloud service's identity, and F is a set of Quality of Service (QoS) feedbacks (i.e., the feedback represent several QoS parameters including availability, security, response time, accessibility, price). Each trust feedback in F is represented in numerical form with the range of [0, 1], where 0, 1, and 0.5 means *negative*, *positive*, and *neutral* feedback respectively. Tf is the timestamps when the trust feedbacks are given. Whenever a user c requests a trust assessment for cloud service s, TMS calculates the trust result, denoted as $Tr(s)$, from the collected trust feedbacks as follows:

$$\mathcal{T}_r(s) = \frac{\sum_{c=1}^{|\mathcal{V}(s)|} \mathcal{F}(c,s) * \mathcal{C}_r(c,s,t_0,t)}{|\mathcal{V}(s)|} * (\chi * \mathcal{C}_t(s,t_0,t)) \quad (1)$$

where $V(s)$ denotes the trust feedbacks given to cloud service s and $jV(s)j$ represents the total number of trust feedbacks. $F(c, s)$ are trust feedbacks from the cth user weighted by the credibility aggregated weights $Cr(c, s, t0, t)$ to allow TMS to dilute the influence of those misleading feedbacks from attacks. $F(c, s)$ is held in the invocation history record h and updated in the corresponding TMS. $Ct(s, t0, t)$ is the rate of trust result changes in a period of time that allows TMS to adjust trust results for cloud services that have been affected by malicious behaviors. χ is the normalized weight factor for the rate of changes of trust results which increase the adaptability of the model.

3.3. Assumptions and Attack Models

In this paper, we assume that TMS is handled by a trusted third party. We also assume that TMS commu- nications are secure because securing communications is not the focus of this paper. Attacks such as *Man-in- the-Middle* (MITM) is therefore beyond the scope of this work. We consider the following types of attacks:

- **Collusion Attacks.** Also known as *collusive* malicious feedback behaviors, such attacks occur when several vicious users collaborate together to give numerous misleading feedbacks to increase the trust result of cloud services (i.e., a self-promoting attack [22]) or to decrease the trust result of cloud services (i.e., a slandering attack [23]). This type of malicious behavior can occur in a *non-collusive* way where a particular malicious user gives multiple misleading feedbacks to conduct a self-promoting attack or a slandering attack.

- **Sybil Attacks.** Such an attack arises when malicious users exploit multiple identities to give numerous misleading feedbacks (e.g., producing a large number of transactions by creating multiple virtual machines for a short period of time to leave fake feedbacks) for a self-promoting or slandering attack. It is interesting to note that attackers can also use multiple identities to disguise their negative historical trust records (i.e.,whitewashing attacks

4 THE CREDIBILITY MODEL

Our proposed credibility model is designed for i) the *Feedback Collusion Detection* including the feedback density and occasional feedback collusion, and ii) the *Sybil Attacks Detection* including the multi-identity recognition and occasional Sybil attacks.

4.1 Feedback Collusion Detection

4.1.1 Feedback Density

Malicious users may give numerous fake feedbacks to manipulate trust results for cloud services (i.e., *Selfpromoting* and *Slandering* attacks). Some researchers suggest that the number of trusted feedbacks can help users to overcome such manipulation where the number of trusted feedbacks gives the evaluator a hint in determining the feedback credibility. However, the number of feedbacks is not enough in determining the credibility of trust feedbacks. For instance, suppose there are two different cloud services *sx* and *sy* and the aggregated trust feedbacks of both cloud services are high (i.e., *sx* has 89% positive feedbacks from 150 feedbacks, *sy* has 92% positive feedbacks from 150 feedbacks). Intuitively, users should proceed with the cloud service that has the higher aggregated trust feedbacks (e.g., *sy* in our case). However, a *Self-promoting* attack might have been performed on cloud service *sy*, which means *sx* should have been selected instead.

To overcome this problem, we introduce the concept of *feedback density* to support the determination of credible trust feedbacks. Specifically, we consider the total number of users who give trust feedbacks to a particular cloud service as the *feedback mass*, the total number of trust feedbacks given to the cloud service as the *feedback volume*. The feedback volume is influenced by the *feedback volume collusion* factor which is controlled by a specified volume collusion threshold. This factor regulates the multiple trust feedbacks extent that could collude the overall trusted feedback volume. For instance, if the volume collusion threshold is set to 15 feedbacks, any user *c* who gives more than 15 feedbacks is considered to be suspicious of involving in a feedback volume collusion. The feedback density of a certain cloud service *s*, *D*(*s*), is calculated as follows:

$$\mathcal{D}(s) = \frac{\mathcal{M}(s)}{|\mathcal{V}(s)| * \mathcal{L}(s)} \qquad (2)$$

where*M*(*s*) denotes the total number of users who give feedback to cloud service *s* (i.e., the feedback mass).

|*V*(*s*)| represents the total number of feedbacks given to cloud service *s* (i.e., the feedback volume). *L*(*s*) represents the *feedback volume collusion* factor, calculated as follows:

$$\mathcal{L}(s) = 1 + \left(\sum_{\forall c \in \mathcal{V}(s)} \left(\sum_{c=1}^{|\mathcal{V}_c(c,s)|} \frac{\sum_{|\mathcal{V}_c(c,s)| > \varepsilon_v(s)} |\mathcal{V}_c(c,s)|}{|\mathcal{V}(s)|} \right) \right) \qquad (3)$$

This factor is calculated as the ratio of the number of feedback given by users |*Vc*(*c*, *s*)| who give feedbacks more than the specified volume collusion threshold *ev*(*s*) over the total number of trust feedbacks received by the cloud service |*V*(*s*)|. The idea is to reduce the value of the multiple feedbacks which are given from the same user.

For instance, consider the two cloud services in the previous example, *sx* and *sy* where *sx* has 89% and *sy* has 92% positive feedbacks, from 150 feedbacks. Assume that the *Feedback Mass* of *sx* is higher than *sy* (e.g., *M*(*x*) = 20 and *M*(*y*) = 5) and the total number of trust feedbacks of the two services is |*Vc*(*c*, *x*)| = 60 and |*Vc*(*c*, *y*)| = 136 feedbacks respectively. We further assume that the volume collusion threshold *ev* is set to 10 feedbacks. According to Equation 2, the *Feedback Density* of *sx* is higher than *sy* (i.e., *D*(*x*) = 0.0953 and *D*(*y*) = 0.0175). In other words, the higher the *Feedback Density*, the more credible are the aggregated feedbacks.

5 THE AVAILABILITY MODEL

Guaranteeing the availability of the Trust Management Service (TMS) is a significant challenge due to the unpredictable number of invocation requests that TMS has to handle at a time, as well as the dynamic nature of the cloud environments. In this, we propose an *availability model*, which considers several factors including the *operational power* to allow TMS nodes to share the workload and *replication determination* to minimize the failure of a node hosting TMS instance. These factors are used to spread several distributed TMS nodes to manage trust feedbacks given by users in a decentralized way.

5.1 Operational Power

In our approach, we propose to spread TMS nodes over various clouds and dynamically direct requests to the appropriate TMS node (e.g., with lower workload), so that its desired availability level can be always maintained. It is crucial to develop a mechanism that helps determine the optimal number of TMS nodes because more nodes residing at various clouds means higher overhead (e.g., cost and resource consumption such as bandwidth and storage space) while lower number of nodes means less availability. To exploit the load balancing technique, we propose that each node hosting a TMS instance reports its operational power.

The operational power factor compares the workload for a particular TMS node with the average workload of all TMS nodes. The operational power for a particular TMS node, *Op*(*stms*), is calculated as the mean of the *Euclidean distance* (i.e., to measure the distance between a particular TMS node workload and the mean of the workload of all TMS nodes) and the TMS node workload (i.e., the percentage of trust feedbacks handled by this node) as follows:

$$\mathcal{O}_p(s_{tms}) = \frac{1}{2} * \left(\sqrt{\left(\frac{\mathcal{V}(s_{tms})}{\mathcal{V}(all_{tms})|} - \frac{\mathcal{V}(mean_{tms})}{\mathcal{V}(all_{tms})} \right)^2 + \frac{\mathcal{V}(s_{tms})}{\mathcal{V}(all_{tms})} \right)$$

where the first part of the equation represents the *Euclidean distance* between the workload of node *stms* and the average workload of all nodes where *V*(*meantms*) denotes the mean of feedbacks handled by all nodes. The second part of the equation represents the ratio of feedbacks handled by a particular node *V*(*stms*) over the total number of feedbacks handled by all nodes *V*(*alltms*).

Based on the operational power factor, TMS uses the workload threshold *ew*(*stms*) to automatically adjust the number of nodes *Ntms* that host TMS instances by creating extra instances to maintain a desired workload for each TMS node. The number of nodes *Ntms* is adjusted as follows:

$$\mathcal{N}_{tms} = \begin{cases} \mathcal{N}_{tms} + 1 & if \quad \mathcal{O}_p(s_{tms}) \geq e_w(s_{tms}) \\ & or \quad \mathcal{N}_{tms} < 1 \\ \mathcal{N}_{tms} & otherwise \end{cases}$$

5.2 Replication Determination

In CloudArmor, we propose to exploit replication techniques to minimize the possibility of the crashing of a node hosting a TMS instance (e.g., overload) to ensure that users can give trust feedbacks or request a trust assessment for cloud services. Replication allows TMS instance to recover any lost data during the down time from its replica. In particular, we propose a particle filtering approach to precisely predict the availability of each node hosting a TMS instance which then will be used to determine the optimal number of the TMS instance's replicas. To predict the availability of each node, we model the TMS instance as an instantaneous (or point) availability. To predict the availability of each node, TMS instance's availability is modeled using the point availability model, then the particle filtering technique is used to estimate the availability. The point availability probability is denoted as:

$$\mathcal{A}(s_{tms}, t) = 1 - F(t) + \int_0^t m(x)(1 - F(t - x))dx$$

where 1 - *F*(*t*) denotes the probability of no failure in (0, *t*], *m*(*x*)*dx* denotes the probability that any renewal points in interval (*x*, *x* + *dx*], and 1 - *F*(*t* □ *x*) represents the probability that no further failure occurs in (*x*, *t*]. This availability function is a function of the time parameter and can be estimated for different time points. In our work, the failure free density follows the exponential distribution and the renewal density function follows the exponential distribution in time domain, (*t*) = *λe*−_*t*, and *m*(*t*) = *μe*−_*t*. It is not easy to observe the pattern of *A*(*stms*, *t*). We therefore conduct the Laplace transform of Equation as below:

$$\mathcal{A}(s_{tms}, s) = \frac{1 - f(s)}{s(1 - f(s)m(s))} = \frac{s + \mu}{s(s + \mu + \lambda)}$$

where *f*(*s*) and *m*(*s*) are the Laplace transforms of the failure-free and renewal density functions.

We use the particle filtering technique to estimate and track the availability. A particle filter is a probabilistic approximation algorithm implementing a Bayes filter and a sequential Monte Carlo method. It maintains a probability distribution for the estimated availability at time *t*, representing the belief of the TMS instance's availability at that time. We initialize a uniformly distributed sample set representing TMS instance's availability state. We assign each sample a same weight *w*. When the availability changes, the particle filter will calculate next availability by adjusting and normalizing each sample's weight. These samples' weights are proportional to the observation likelihood *p*(*yjz*). The particle filters randomly draw samples from the current sample set whose probability can be given by the weights. Then we can apply the particle filters to estimate the possible next availability state for each new particle. The prediction and update steps will keep going until convergence. We calculate the weight distribution by considering the bias resulted from the routing information between users and TMS instances (e.g., routing-hops between the user and the instances or whether user and TMS instances are in the same IP address segment). The Sequential Importance Sampling (SIS) algorithm consists of recursive propagation of the weights and support points as each measurement is received sequentially. To tackle the degeneracy problem, we adopt a more advanced algorithm with resampling. It has less time complexity and minimizes the Monte-Carlo variation. The overall particle filtering based estimation methodology is summarized in Algorithm 1.

Algorithm 1 Particle Filtering based Algorithm

1. **Initialization:** compute the weight distribution $\mathcal{D}_w(\mathcal{A}(s_{tms}))$ according to prior knowledge on replicas, e.g., the IP address of server hosting replicas etc.
2. **Generation:** generate the particle set and assign the particle set containing $\mathcal{N}$ particles
   - generate initial particle set $\mathcal{P}_0$ which has $\mathcal{N}$ particles, $\mathcal{P}_0 = (p_{0,0}, p_{0,1}, \cdots p_{0,\mathcal{N}-1})$ and distribute them in a uniform distribution in the initial stage. Particle $p_{0,k} = (\mathcal{A}(s_{tms})_{0,k}, weight_{0,k})$
   - assign weight to the particles according to our weight distribution $\mathcal{D}_w(\mathcal{A}(s_{tms}))$.
3. **Resampling:**
   - Resample $\mathcal{N}$ particles from the particle set from a particle set $\mathcal{P}_t$ using weights of each particles.
   - generate new particle set $\mathcal{P}_{t+1}$ and assign weight according to $\mathcal{D}_w(\mathcal{A}(s_{tms}))$
4. **Estimation:** predict new availability of the particle set $\mathcal{P}_t$ based on availability function $\mathcal{A}(s_{tms}, t)$.
5. **Update:**
   - recalculate the weight of $\mathcal{P}_t$ based on measurement m, $w_{t,k} = \prod(\mathcal{D}_w(\mathcal{A}(s_{tms})_{t,k}))(\frac{1}{\sqrt{2\pi}\sigma_y})exp(-\frac{\delta\mathcal{A}(s_{tms})^2_{t,k}}{2\sigma_y^2})$, where $\delta\mathcal{A}(s_{tms})_k = m_{\mathcal{A}}(s_{tms}) - \mathcal{A}(s_{tms})_{t,k}$
   - calculate current availability by mean value of $p_t(\mathcal{A}(s_{tms})_t)$
6. Go to step 3 and iteration until convergence

Based on the predicted availability of the TMS instance *A*(*stms*, *t*), the availability threshold denoted as *ea* that ranges from 0 to 1 and the total number of *stms* replicas denoted *r* are calculated. The desired goal of the replication is to ensure that at least one replica is available, represented in the following formula:

$$e_a(s_{tms}) < \mathcal{A}(s_{tms}, t)^{r(s_{tms})}$$

where *A*(*stms*, *t*)*r*(*stms*) represents the probability of at least one TMS instance's replica is available. As a result, the optimal number of TMS instance's replicas can be calculated as follows:

$$r(s_{tms}) > log_{\mathcal{A}(s_{tms},t)}(e_a(s_{tms}))$$

5.3 Trust Result Caching

Due to the fact that several credibility factors are considered in this framework when computing the trust result for a particular cloud service, it would be odd if the TMS instance retrieves all trust feedbacks given to a particular cloud service and computes the trust result every time it receives a trust assessment request from a user. Instead we propose to cache the trust results and the credibility weights based on the number of new trust feedbacks to avoid unnecessary trust result computations. The caching process is controlled by two thresholds: one for users *eCache*(*c*) and one for cloud services *eCache*(*s*). If the TMS instance receives a trust assessment request from a user, it should use the trust result in the cache as much as possible, instead of computing the trust result from scratch. The TMS instance updates the cache based on the number of new trust feedbacks (i.e., since the last update) given by a particular consumer *jVc*(*c*, *s*)*jCache* and the number of new trust feedbacks given to a particular cloud service |*V*(*s*)| *Cache*. The caching process is briefly shown in Algorithm 2.

Algorithm 2 Trust Results & Credibility Weights Caching Algorithm

Input: s, **Output**: $\mathcal{T}r(s)$
Count $|\mathcal{V}c(c,s)|_{Cache}$ /*TMS instance counts the total number of new trust feedbacks given by a particular consumer*/
if $|\mathcal{V}c(c,s)|_{Cache} \geq e_{Cache}(c)$ **then** /*TMS determines whether a recalculation is required for credibility factors related to the consumer*/
    Compute $\mathcal{J}(c)$; Compute $\mathcal{B}(c)$
    Compute $\mathcal{M}_{id}(c)$; Compute $\mathcal{C}r(c,s)$
end if
Count $|\mathcal{V}(s)|_{Cache}$ /*TMS instance counts the total number of new trust feedbacks given to a particular cloud service*/
if $|\mathcal{V}(s)|_{Cache} \geq e_{Cache}(s)$ **then** /*TMS determines whether a recalculation is required for credibility factors related to the cloud service including the trust result*/
    Compute $\mathcal{D}(s)$; Compute $\mathcal{C}r(c,s)$
    Compute $\mathcal{T}r(s)$
end if

5.4 Instances Management

In the frame work, we propose that one TMS instance acts as the *main instance* while the rest instances act as *normal instances*. The main instance is responsible for the optimal number of nodes estimation, feedbacks reallocation, trust result caching (consumer side), availability of each node prediction, and TMS instance replication. Normal instances are responsible for trust assessment and feedback storage, the trust result caching (cloud service side), and frequency table update.

Algorithm 3 shows the brief process on how TMS instances are managed.

Unlike previous work such as where all invocation history records for a certain client is mapped to a particular TMS instance (e.g., all feedback given to a certain cloud service in our case), in our approach, each TMS instance is responsible for feedbacks given to a set of cloud services and updates the frequency table. The frequency table shows which TMS instance is responsible for which cloud service and how many

feedbacks it has handled. Example 1 illustrates how feedbacks can be reallocated from one TMS instance to a different instance. In this example, there are three TMS instances and the workload threshold *ew*(*stms*) is set to 50%. TMS instance *tmsid*(1) triggers the threshold, therefore according to Algorithm 3, the trust feedbacks for

the cloud service (2) are reallocated to *tmsid*(2), which has the lowest feedbacks.

Algorithm 3 Instances Management Algorithm

1. **Initialization:** $tms_{id}(0)$ computes $\mathcal{O}_p(s_{tms})$ for all trust management service nodes if any
2. **Generation:** $tms_{id}(0)$ estimates $\mathcal{N}_{tms}$ and generates additional trust management service nodes if required
3. **Prediction:** $tms_{id}(0)$ predicts new availability of all trust management service nodes $\mathcal{A}(s_{tms}, t)$ using Algorithm 1
4. **Replication:** $tms_{id}(0)$ determines $r(s_{tms})$, and generate replicas for each trust management service node
5. **Caching:** $tms_{id}(0)$ starts caching trust results (consumer side) and $tms_{id}(s)$ start caching trust results (cloud service side) using Algorithm 2
6. **Update:** All $tms_{id}(s)$ update the frequency table
7. **Check Workload 1:** $tms_{id}(0)$ checks whether $e_w(s_{tms})$ is triggered by any $tms_{id}(s)$ before reallocation

if $\mathcal{O}_p(s_{tms}) \geq e_w(s_{tms})$ and $\mathcal{V}(s_{tms}) \geq \mathcal{V}(mean_{tms})$ **then**
    go to next step
else
    go to step 3
end if

8. **Reallocation:**
   - $tms_{id}(0)$ asks $tms_{id}(s)$ which triggered $e_w(s_{tms})$ to reallocate all trust feedbacks of the cloud service that has the lowest $|\mathcal{V}(s)|$ to another $tms_{id}(s)$ that has the lowest $\mathcal{V}(s_{tms})$
   - perform step 6

9. **Check Workload 2:** $tms_{id}(0)$ computes $\mathcal{O}_p(s_{tms})$ for all trust management service nodes and checks whether $e_w(s_{tms})$ is triggered for any $tms_{id}(s)$ after reallocation

if $\mathcal{O}_p(s_{tms}) \geq e_w(s_{tms})$ and $\mathcal{V}(s_{tms}) \geq \mathcal{V}(mean_{tms})$ **then**
    go to step 2
else
    go to step 3
end if

Example 1: Reallocation ($e_w(S_{tms})$ = 50%)
Frequency Table Before Reallocation (Step 1)
($tms_{id}(1)$, $jV(1)j$: 200, $jV(2)j$: 150, $jV(3)j$: 195)
($tms_{id}(2)$, $jV(4)j$: 30, $jV(5)j$: 20, $jV(6)j$: 45)
($tms_{id}(3)$, $jV(7)j$: 90, $jV(8)j$: 35, $jV(9)j$: 95)
Check Workload (Step 2)
($tms_{id}(1)$, $O_p(1_{tms})$: 0.617)
($tms_{id}(2)$, $O_p(2_{tms})$: 0.278)
($tms_{id}(3)$, $O_p(3_{tms})$: 0.205)
Frequency Table After Reallocation (Step 3)
($tms_{id}(1)$, $jV(1)j$: 200, $jV(3)j$: 195)
($tms_{id}(2)$, $jV(2)j$: 150, $jV(4)j$: 30, $jV(5)j$: 20, $jV(6)j$: 45)
($tms_{id}(3)$, $jV(7)j$: 90, $jV(8)j$: 35, $jV(9)j$: 95)

6. CONCLUSION

Given the highly dynamic, distributed, and nontransparent nature of cloud services, managing and establishing trust between cloud service users and cloud services remains a significant challenge. Cloud service users' feedback is a good source to assess the overall trustworthiness of cloud services. However, malicious users may collaborate together to i) disadvantage a cloud service by giving multiple misleading trust feedbacks (i.e., collusion attacks) or ii) trick users into trusting cloud services that are not trustworthy by creating several accounts and giving misleading trust feedbacks (i.e., Sybil attacks). In this paper, we have presented novel techniques that help in detecting plausibility based attacks and allowing users to effectively identify trustworthy cloud services. In particular, we introduce a credibility model that not only identifies misleading trust feedbacks from collusion attacks but also detects Sybil attacks no matter these attacks take place in a long or short period of time (i.e., strategic or occasional attacks respectively). We also develop an availability model that maintains the trust management service at a desired level. We have collected a large number of consumer's trust feedbacks given on real-world cloud services (i.e., over 10,000 records) to evaluate our proposed techniques. The experimental results demonstrate the applicability of our approach and show the capability of detecting such malicious behaviors. There are a few directions for our future work. We plan to combine different trust management techniques such as plausibility and recommendation to increase the trust results accuracy. Performance optimization of the trust management service is another focus of our future research work.

REFERENCES

[1] S. M. Khan and K. W. Hamlen, "Hatman: Intra-Cloud Trust Management for Hadoop," in *Proc. CLOUD'12*, 2012.
[2] S. Pearson, "Privacy, Security and Trust in Cloud Computing," in *Privacy and Security for Cloud Computing*, ser. Computer Communications and Networks, 2013, pp. 3–42.
[3] J. Huang and D. M. Nicol, "Trust Mechanisms for Cloud Computing," *Journal of Cloud Computing*, vol. 2, no. 1, pp. 1–14, 2013.
[4] K. Hwang and D. Li, "Trusted Cloud Computing with Secure Resources and Data Coloring," *IEEE Internet Computing*, vol. 14, no. 5, pp. 14–22, 2010.
[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[6] S. Habib, S. Ries, and M. Muhlhauser, "Towards a Trust Management System for Cloud Computing," in *Proc. of TrustCom'11*, 2011.
[7] I. Brandic, S. Dustdar, T. Anstett, D. Schumm, F. Leymann, and R. Konrad, "Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds," in *Proc. of CLOUD'10*, 2010.

[8] W. Conner, A. Iyengar, T. Mikalsen, I. Rouvellou, and K. Nahrstedt, "A Trust Management Framework for Service-Oriented Environments," in *Proc. of WWW'09*, 2009