CloudArmor: Supporting Reputation-based Trust Management for Cloud Services

Talal H. Noor, Quan Z. Sheng, *Member, IEEE,* Lina Yao, Schahram Dustdar, *Senior Member, IEEE,* and Anne H.H. Ngu

Abstract—Trust management is one of the most challenging issues for the adoption and growth of cloud computing. The highly dynamic, distributed, and non-transparent nature of cloud services introduces several challenging issues such as privacy, security, and availability. Preserving consumers' privacy is not an easy task 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 CloudArmor, 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—Cloud computing, trust management, reputation, credibility, credentials, security, privacy, availability.

1 Introduction

The highly dynamic, distributed, and non-transparent nature of cloud services make the trust management in cloud environments a significant challenge [1], [2], [3], [4]. According to researchers at Berkeley [5], trust and security are ranked one of the top 10 obstacles for the adoption of cloud computing. Indeed, Service-Level Agreements (SLAs) alone are inadequate to establish trust between cloud consumers and providers because of its unclear and inconsistent clauses [6].

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 [7], [6], [8], [9]. In reality, it is not unusual that a cloud service experiences malicious behaviors (e.g., collusion or Sybil attacks) from its users [6], [10]. 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 [11]. 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 [12].
- *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 [13]. Finally, it is difficult to predict when malicious behaviors occur (i.e., strategic VS. occasional behaviors) [14].
- *Trust Management Service's Availability.* A trust management service (TMS) provides an interface be-

- *Talal H. Noor, is with the College of Computer Science and Engineering, Taibah University, Yanbu, Medinah 46421-7143, Saudi Arabia. E-mail: tnoor@taibahu.edu.sa*
- *Quan Z. Sheng and Lina Yao are with the School of Computer Science, The University of Adelaide, Adelaide SA 5005, Australia.*
- *Schahram Dustdar is with the Distributed Systems Group, Vienna University of Technology, Austria. Anne H.H. Ngu is with the Department of Computer Science, Texas State University, USA.*

tween users and cloud services for effective trust management. However, guaranteeing the availability of TMS is a difficult problem due to the unpredictable number of users and the highly dynamic nature of the cloud environment [7], [6], [10]. Approaches that require understanding of users' interests and capabilities through similarity measurements [15] or operational availability measurements [16] (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 CloudArmor (CLOud consUmers creDibility Assessment & tRust manageMent of clOud seRvices): a framework for reputation-based trust management in cloud environments. In CloudArmor, trust is delivered as a service (TaaS) where TMS spans several distributed nodes to manage feedbacks in a decentralized way. CloudArmor exploits techniques to identify credible feedbacks from malicious ones. In a nutshell, the salient features of CloudArmor 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 CloudArmor 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 of CloudArmor and the results of experimental evaluations. Finally, Section 7 overviews the related work and Section 8 provides some concluding remarks.

2 The CloudArmor Framework

The CloudArmor 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 [17], [18]. 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 (more details about cloud services models and designs can be found in [19]). 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2408613, IEEE Transactions on Parallel and Distributed Systems

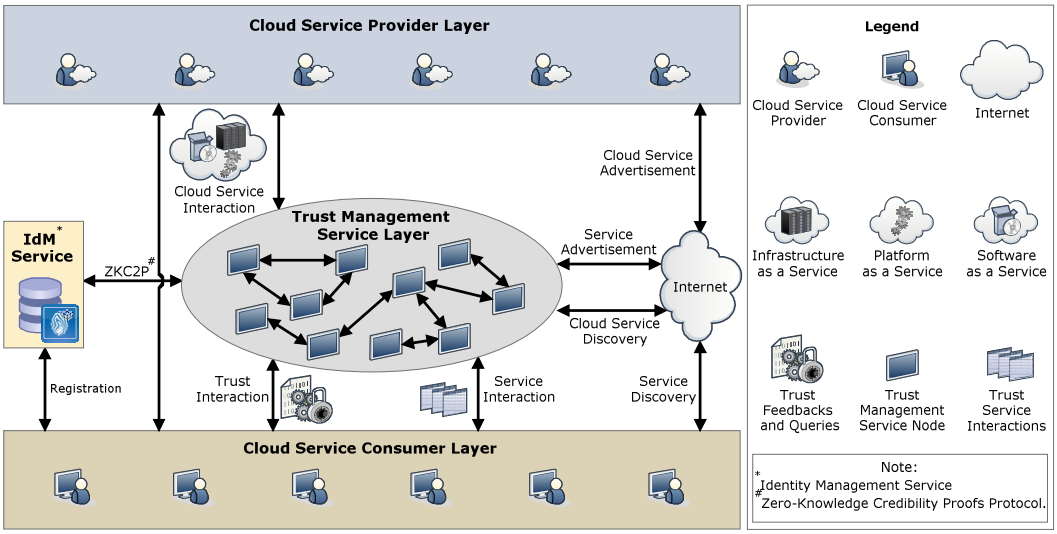


Fig. 1. Architecture of the CloudArmor Trust Management Framework

prove the credibility of a particular consumer's feedback (details in Section 3).

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 [20], 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 [11]. 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 [21]).

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 (see details in Section 4.2). 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 $\mathcal{I} = (\mathcal{C}, \mathcal{C}_a, \mathcal{T}_i)$ for each user. $\mathcal{C}$ is the user's primary identity (e.g., user name). $\mathcal{C}_a$ represents a set of credentials' attributes (e.g., passwords, postal address, and IP address) and $\mathcal{T}_i$ represents the user's registration time in TMS. More details on how IdM facilitates TMS in the detection of Sybil attacks can be found in Section 4.2.

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 service[1]. From users' feedback, the trust behavior of a cloud service is actually a collection of invocation history records, represented by a tuple $\mathcal{H} = (\mathcal{C}, \mathcal{S}, \mathcal{F}, \mathcal{T}_f)$, where $\mathcal{C}$ is the user's primary identity, $\mathcal{S}$ is the cloud service's identity, and $\mathcal{F}$ 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 $\mathcal{F}$ is represented in numerical form with the range of [0, 1], where 0, 1, and 0.5 means *negative*, *positive*, and *neutral* feedback respectively. $\mathcal{T}_f$ 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 $\mathcal{T}_r(s)$, 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 $\mathcal{V}(s)$ denotes the trust feedbacks given to cloud service s and $|\mathcal{V}(s)|$ represents the total number of trust feedbacks. $\mathcal{F}(c,s)$ are trust feedbacks from the c^{th} user weighted by the credibility aggregated weights $\mathcal{C}_r(c,s,t_0,t)$ to allow TMS to dilute the influence of those misleading feedbacks from attacks. $\mathcal{F}(c,s)$ is held in the invocation history record h and updated in the corresponding TMS. $\mathcal{C}_t(s,t_0,t)$ 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. More details on how to calculate $\mathcal{C}_r(c,s,t_0,t)$ and $\mathcal{C}_t(s,t_0,t)$ are described in Section 4.

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 communications 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 [13], [22] 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 [24]).

1. We assume a transaction-based feedback where all feedbacks are held in TMS

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., *Self-promoting* 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 [25]. However, the number of feedbacks is not enough in determining the credibility of trust feedbacks. For instance, suppose there are two different cloud services s_x and s_y and the aggregated trust feedbacks of both cloud services are high (i.e., s_x has 89% positive feedbacks from 150 feedbacks, s_y has 92% positive feedbacks from 150 feedbacks). Intuitively, users should proceed with the cloud service that has the higher aggregated trust feedbacks (e.g., s_y in our case). However, a *Self-promoting* attack might have been performed on cloud service s_y, which means s_x 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, $\mathcal{D}(s)$, is calculated as follows:

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

where $\mathcal{M}(s)$ denotes the total number of users who give feedback to cloud service s (i.e., the feedback mass).

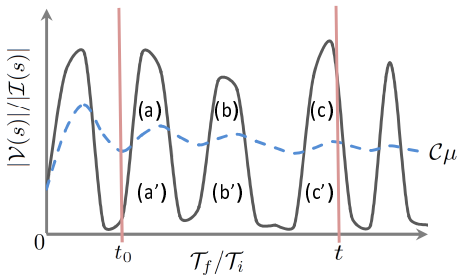


Fig. 2. Occasional Attacks Detection

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

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

This factor is calculated as the ratio of the number of feedback given by users $|\mathcal{V}_c(c,s)|$ who give feedbacks more than the specified volume collusion threshold $e_v(s)$ over the total number of trust feedbacks received by the cloud service $|\mathcal{V}(s)|$. 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, s_x and s_y where s_x has 89% and s_y has 92% positive feedbacks, from 150 feedbacks. Assume that the *Feedback Mass* of s_x is higher than s_y (e.g., $\mathcal{M}(x) = 20$ and $\mathcal{M}(y) = 5$) and the total number of trust feedbacks of the two services is $|\mathcal{V}_c(c,x)| = 60$ and $|\mathcal{V}_c(c,y)| = 136$ feedbacks respectively. We further assume that the volume collusion threshold e_v is set to 10 feedbacks. According to Equation 2, the *Feedback Density* of s_x is higher than s_y (i.e., $\mathcal{D}(x) = 0.0953$ and $\mathcal{D}(y) = 0.0175$). In other words, the higher the *Feedback Density*, the more credible are the aggregated feedbacks.

4.1.2 Occasional Feedback Collusion

Since collusion attacks against cloud services occur sporadically [14], we consider *time* as an important factor in detecting occasional and periodic collusion attacks (i.e., periodicity). In other words, we consider the total number of trust feedbacks $|\mathcal{V}(s)|$ given to cloud service s during a period of time $[t_0, t]$. A sudden change in the feedback behavior indicates likely an occasional feedback collusion because the change of the number of trust feedbacks given to a cloud service happen abruptly in a short period of time.

To detect such behavior, we measure the percentage of occasional change in the total number of feedbacks among the whole feedback behavior (i.e., users' behavior in giving feedbacks for a certain cloud service). The occasional feedback collusion factor $\mathcal{O}_f(s,t_0,t)$ of cloud service s in a period of time $[t_0, t]$, is calculated as follows:

$$\mathcal{O}_f(s,t_0,t) = 1 - \left(\frac{\left(\int_{t_0}^{t} |\mathcal{V}(s,t)|\,\mathrm{d}t\right) - \left(\int_{t_0}^{t} \Delta_f(s,t)\mathrm{d}t\right)}{\int_{t_0}^{t} |\mathcal{V}(s,t)|\,\mathrm{d}t} \right)$$

$$where \Delta_f(s,t) = \begin{cases} \mathcal{C}\mu\left(|\mathcal{V}(s,t)|\right) & if\ |\mathcal{V}(s,t)| \geq \mathcal{C}\mu\left(|\mathcal{V}(s,t)|\right) \\ |\mathcal{V}(s,t)| & otherwise \end{cases} \quad (4)$$

where the first part of the numerator represents the whole area under the curve which represents the feedback behavior for the cloud service s (i.e., $a \bigcup a'$, $b \bigcup b'$ and $c \bigcup c'$ in Figure 2). The second part of the numerator represents the intersection between the area under the curve and the area under the cumulative mean of the total number of trust feedbacks (i.e., the area $a' \bigcup b' \bigcup c'$ in Figure 2). $\mathcal{C}\mu\left(|\mathcal{V}(s,t)|\right)$ represents the mean of all points in the total number of trust feedbacks and up to the last element because the mean is dynamic and changes from time to time. The denominator represents the whole area under the curve. As a result, the occasional collusion attacks detection is based on measuring the occasional change in the total number of trust feedbacks in a period of time. The higher the occasional change in the total number of trust feedbacks, the more likely that the cloud service has been affected by an occasional collusion attack.

4.2 Sybil Attacks Detection

4.2.1 Multi-Identity Recognition

Since users have to register their credentials at the *Trust Identity Registry*, we believe that *Multi-Identity Recognition* is applicable by comparing the values of users' credential attributes from the identity records $\mathcal{I}$. The main goal of this factor is to protect cloud services from malicious users who use multiple identities (i.e., *Sybil* attacks) to manipulate the trust results. In a typical *Trust Identity Registry*, the entire identity records $\mathcal{I}$ are represented as a list of m users' primary identities $\mathcal{C}_p = \{p_1, p_2, ..., p_m\}$ (e.g., user name) and a list of n credentials' attributes $\mathcal{C}_a = \{a_1, a_2, ..., a_n\}$ (e.g., passwords, postal address, IP address, computer name). In other words, the entire $\mathcal{C}_p \times \mathcal{C}_a$ (Consumer's Primary Identity-Credentials' Attributes) Matrix, denoted as IM, covers all users who registered their credentials in TMS. The credential attribute value for a particular consumer $v_{c,t}$ is stored in TMS without including credentials with sensitive information using the ZKC2P (see Section 3).

We argue that TMS can identify patterns in users' anonymous credentials. Malicious users can use similar credentials in different identity records $\mathcal{I}$. Thus, we translate IM to the *Multi-Identity Recognition Matrix*, denoted as $MIRM$, which similarly covers the entire identity records $\mathcal{I}$ represented as the entire $\mathcal{C}_p \times \mathcal{C}_a$ matrix. However, the value for a particular consumer $q_{c,t}$ in the new matrix represents the frequency of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2408613, IEEE Transactions on Parallel and Distributed Systems

credential attribute value for the same particular consumer $v_{c,t}$ in the same credential attribute (i.e., attribute a_t). The frequency of a particular credential attribute value $v_{c,t}$, denoted as $q_{c,t}$, is calculated as the number of times of appearance (denoted as $\mathcal{A}_p$) that the credential value appears in the t^{th} credential attribute normalized by the total number of identity records (i.e., the length of a_t) as follows:

$$q_{c,t} = \frac{\sum_{c=1}^{c=m} (\mathcal{A}_p(v_{c,t}))}{|a_t|} \tag{5}$$

Then, the *Multi-Identity Recognition* factor $\mathcal{M}_{id}$ is calculated as the sum of frequencies of each credential attribute value for a particular consumer normalized by the total number of identity record as follows:

$$\mathcal{M}_{id}(c) = 1 - \left(\sum_{t=1}^{t=n} q_{c,t} \right) \tag{6}$$

where the sum of $q_{c,t}$ represents the similar credentials distributed over different identity records $\mathcal{I}$ and $\mathcal{M}_{id}(c)$ represents the opposite (i.e., at least that the consumer has fairly unique credentials).

4.2.2 Occasional Sybil Attacks

Malicious users may manipulate trust results to disadvantage particular cloud services by creating multiple accounts and giving misleading feedbacks in a short period of time (i.e., Sybil attacks). To overcome the occasional Sybil attacks, we consider the total number of established identities $|\mathcal{I}(s)|$ for users who give feedbacks to cloud service s during a period of time $[t_0, t]$. The sudden changes in the total number of established identities indicates a possible occasional Sybil attack. To detect such behavior, we measure the percentage of occasional change in the total number of established identities among the whole identity behavior (i.e., all established identities for users who gave feedback to a particular cloud service). Similarly, the occasional Sybil attacks factor $\mathcal{O}_i(s, t_0, t)$ of cloud service s in a period of time $[t_0, t]$, is calculated as follows:

$$\mathcal{O}_i(s,t_0,t) = 1 - \left(\frac{\left(\int_{t_0}^{t} |\mathcal{I}(s,t)| \, \mathrm{d}t \right) - \left(\int_{t_0}^{t} \Delta_i(s,t) \mathrm{d}t \right)}{\int_{t_0}^{t} |\mathcal{I}(s,t)| \, \mathrm{d}t} \right)$$

$$where \Delta_i(s,t) = \begin{cases} \mathcal{C}\mu\left(|\mathcal{I}(s,t)|\right) & if\ |\mathcal{I}(s,t)| \geq \\ & \mathcal{C}\mu\left(|\mathcal{I}(s,t)|\right) \\ |\mathcal{I}(s,t)| & otherwise \end{cases} \tag{7}$$

4.3 Feedback Credibility

Based on the proposed credibility metrics, TMS dilutes the influence of those misleading feedbacks by assigning the credibility aggregated weights $\mathcal{C}_r(c, s, t_0, t)$ to each trust feedback as shown in Equation 1. $\mathcal{C}_r(c, s, t_0, t)$ is calculated as follows:

$$C_r(c,s,t_0,t) = \frac{1}{\lambda} * (\rho * \mathcal{D}(s) + \phi * \mathcal{O}_f(s,t_0,t) + \Omega * \mathcal{M}_{id}(c) + \iota * \mathcal{O}_i(s,t_0,t)) \tag{8}$$

where ρ and $\mathcal{D}(s)$ denote the *Feedback Density* factor's normalized weight and the factor's value respectively. ϕ and $\mathcal{O}_f(s, t_0, t)$ denote the parameter of the occasional feedback collusion factor and the factor's value respectively. Ω denotes the *Multi-identity Recognition* normalized weight and $\mathcal{M}_{id}(c)$ denotes the factor's value. ι denotes the occasional Sybil attacks' normalized weight and $\mathcal{O}_i(s, t_0, t)$ denotes the factor's value. λ represents the number of factors used to calculate $\mathcal{C}_r(c, s, t_0, t)$. If only feedback density is considered, λ will be 1. If all credibility factors are considered, λ will be 4. All the metrics of the credibility model complement each other in detecting malicious behaviors and their influence can be adjusted using the above mentioned parameters.

4.4 Change Rate of Trust Results

To allow TMS to adjust trust results for cloud services that have been affected by malicious behaviors, we introduce an additional factor called the *change rate of trust results*. The idea behind this factor is to compensate the affected cloud services by the same percentage of damage in the trust results. Given $\mathcal{C}on(s, t_0)$ the conventional model (i.e., calculating the trust results without considering the proposed approach) for cloud service s in a previous time instance, $\mathcal{C}on(s, t)$ the conventional model for the same cloud service calculated in a more recent time instance, the credibility aggregated weights $\mathcal{C}_r(c, s, t_0, t)$, and $e_t(s)$ the attacks percentage threshold. The change rate of trust results factor $\mathcal{C}_t(s, t_0, t)$ is calculated as follows:

$$\mathcal{C}_t(s,t_0,t) = \begin{cases} \frac{\mathcal{C}on(s,t_0)}{\mathcal{C}on(s,t)} + 1 & if\ \mathcal{C}on(s,t) < \mathcal{C}on(s,t_0) \\ & and \\ & 1 - \mathcal{C}_r(c,s,t_0,t) \geq e_t(s) \\ 0 & otherwise \end{cases} \tag{9}$$

where $\frac{\mathcal{C}on(s,t_0)}{\mathcal{C}on(s,t)}$ represents the change rate of trust results for cloud service s during a period of time $[t_0, t]$. The idea behind adding 1 to this ratio is to increase the trust result for the affected cloud services. The change rate of trust results will only be used if the conventional model in the more recent time instance is less than the conventional model in the previous time instance and the attacks percentage during the same period of time $[t_0, t]$ (i.e., $1 - \mathcal{C}_r(c, s, t_0, t)$) is larger or equal to the attacks percentage threshold. For instance, if the conventional model in the current time for cloud service a is less than the conventional model 10 days ago, a will not be rewarded because the attacks percentage is less than the attacks percentage threshold (e.g., $1 - \mathcal{C}_r(c, a, t_0, t) = 20\%$ and $e_t(a) = 30\%$).

The change rate of trust results is designed to limit the rewards to cloud services that are affected by slandering attacks (i.e., cloud services that have decreased trust results) because TMS can dilute the increased trust results from self-promoting attacks using the credibility

factors (i.e., $\mathcal{C}_r(c, a, t_0, t)$). The adaptive change rate of trust results factor can be used to assign different weights using χ the normalized weight factor as shown in Equation 1.

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 CloudArmor, 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, $\mathcal{O}_p(s_{tms})$, 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:

$$\mathcal{O}_p(s_{tms}) = \frac{1}{2} * \left(\sqrt{\left(\frac{\mathcal{V}(s_{tms})}{\mathcal{V}(all_{tms})} - \frac{\mathcal{V}(mean_{tms})}{\mathcal{V}(all_{tms})} \right)^2} + \frac{\mathcal{V}(s_{tms})}{\mathcal{V}(all_{tms})} \right) \tag{10}$$

where the first part of the equation represents the *Euclidean distance* between the workload of node s_{tms} and the average workload of all nodes where $\mathcal{V}(mean_{tms})$ 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 $\mathcal{V}(s_{tms})$ over the total number of feedbacks handled by all nodes $\mathcal{V}(all_{tms})$.

Based on the operational power factor, TMS uses the workload threshold $e_w(s_{tms})$ to automatically adjust the number of nodes $\mathcal{N}_{tms}$ that host TMS instances by creating extra instances to maintain a desired workload for each TMS node. The number of nodes $\mathcal{N}_{tms}$ is adjusted as follows:

$$\mathcal{N}_{tms} = \begin{cases} \mathcal{N}_{tms} + 1 & if \quad \mathcal{O}_p(s_{tms}) \geq e_w(s_{tms}) \\ & or \quad \mathcal{N}_{tms} < 1 \\ \mathcal{N}_{tms} & otherwise \end{cases} \tag{11}$$

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 [26], then the particle filtering technique is used to estimate the availability. The point availability probability is denoted as:

$$\mathcal{A}(s_{tms}, t) = 1 - F(t) + \int_0^t m(x)(1 - F(t - x))dx \tag{12}$$

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, $f(t) = \lambda e^{-\lambda t}$, and $m(t) = \mu e^{-\mu t}$. It is not easy to observe the pattern of $\mathcal{A}(s_{tms}, t)$. We therefore conduct the Laplace transform of Equation 12 as below:

$$\mathcal{A}(s_{tms}, s) = \frac{1 - f(s)}{s(1 - f(s)m(s))} = \frac{s + \mu}{s(s + \mu + \lambda)} \tag{13}$$

where $f(s)$ and $m(s)$ are the Laplace transforms of the failure-free and renewal density functions. Equation 13 in time domain can be obtained using:

$$\mathcal{A}(s_{tms}, t) = 1 - \frac{\lambda}{\mu}(1 - e^{-\mu t}) \tag{14}$$

For more technicial details, interested readers can refer to [26].

To this point, we can model the TMS instance's availability prediction problem via defining the state

function and measurement function respectively by using:

$$\begin{aligned} z(t+1) &= \mathcal{A}(s_{tms}, t) + \epsilon_z \\ y(t+1) &= z(t+1) + \epsilon_y \\ where \quad \epsilon_z &\sim \mathcal{N}(0, \sigma_z^2), \epsilon_y \sim \mathcal{N}(0, \sigma_y^2) \end{aligned} \tag{15}$$

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 $p(y|z)$. 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 [27]. It has less time complexity and minimizes the Monte-Carlo variation. The overall particle filtering based estimation methodology is summarized in Algorithm 1.

Based on the predicted availability of the TMS instance $\mathcal{A}(s_{tms}, t)$, the availability threshold denoted as e_a that ranges from 0 to 1 and the total number of s_{tms} 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:

$$e_a(s_{tms}) < \mathcal{A}(s_{tms}, t)^{r(s_{tms})} \tag{16}$$

where $\mathcal{A}(s_{tms}, t)^{r(s_{tms})}$ 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:

$$r(s_{tms}) > log_{\mathcal{A}(s_{tms},t)}(e_a(s_{tms})) \tag{17}$$

5.3 Trust Result Caching

Due to the fact that several credibility factors are considered in CloudArmor 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 $e_{Cache}(c)$ and one for cloud services $e_{Cache}(s)$. 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 $|\mathcal{V}c(c,s)|_{Cache}$ and the number of new trust feedbacks given to a particular cloud service $|\mathcal{V}(s)|_{Cache}$. The caching process is briefly shown in Algorithm 2.

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, $\mathcal{P}_0 = (p_{0,0}, p_{0,1}, ...p_{0,\mathcal{N}-1})$ 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, $w_{t,k}$= $\prod(\mathcal{D}_w(\mathcal{A}(s_{tms})_{t,k}))(\frac{1}{\sqrt{2\pi}\sigma_y})exp(-\frac{\delta\mathcal{A}(s_{tms})_{t,k}^2}{2\sigma_y^2})$, 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

5.4 Instances Management

In CloudArmor, 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 [8] where all invocation history records for a certain client is mapped to a particular TMS instance (e.g., all feedback given to a

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

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

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 $e_w(s_{tms})$ is set to 50%. TMS instance $tms_{id}(1)$ triggers the threshold, therefore according to Algorithm 3, the trust feedbacks for the cloud service (2) are reallocated to $tms_{id}(2)$, which has the lowest feedbacks.

Example 1: Reallocation ($e_w(s_{tms})$ = 50%)

Frequency Table Before Reallocation (Step 1)
($tms_{id}(1)$, $|\mathcal{V}(1)|$: 200, $|\mathcal{V}(2)|$: 150, $|\mathcal{V}(3)|$: 195)
($tms_{id}(2)$, $|\mathcal{V}(4)|$: 30, $|\mathcal{V}(5)|$: 20, $|\mathcal{V}(6)|$: 45)
($tms_{id}(3)$, $|\mathcal{V}(7)|$: 90, $|\mathcal{V}(8)|$: 35, $|\mathcal{V}(9)|$: 95)

Check Workload (Step 2)
($tms_{id}(1)$, $\mathcal{O}_p(1_{tms})$: 0.617)
($tms_{id}(2)$, $\mathcal{O}_p(2_{tms})$: 0.278)
($tms_{id}(3)$, $\mathcal{O}_p(3_{tms})$: 0.205)

Frequency Table After Reallocation (Step 3)
($tms_{id}(1)$, $|\mathcal{V}(1)|$: 200, $|\mathcal{V}(3)|$: 195)
($tms_{id}(2)$, $|\mathcal{V}(2)|$: 150, $|\mathcal{V}(4)|$: 30, $|\mathcal{V}(5)|$: 20, $|\mathcal{V}(6)|$: 45)
($tms_{id}(3)$, $|\mathcal{V}(7)|$: 90, $|\mathcal{V}(8)|$: 35, $|\mathcal{V}(9)|$: 95)

6 Implementation and Experimental Evaluation

In this section, we report the implementation and experimental results in validating the proposed approach. Our implementation and experiments were developed to validate and study the performance of both the credibility model and the availability model.

6.1 System Implementation

The trust management service's implementation is part of our large research project, named CloudArmor[2], which offers a platform for reputation-based trust management of cloud services [28], [29], [30], [9]. The platform provides an environment where users can give feedback and request trust assessment for a particular cloud service. Specifically, the trust management service (TMS) consists of two main components: the *Trust Data Provisioning* and the *Trust Assessment Function*.

The Trust Data Provisioning. This component is responsible for collecting cloud services and trust information. We developed the *Cloud Services Crawler* module based on the Open Source Web Crawler for Java (crawler4j[3]) and extended it to allow the platform to automatically discover cloud services on the Internet. We implemented a set of functionalities to simplify the crawling process and made the crawled data more comprehensive (e.g., `addSeeds()`, `selectCrawlingDomain()`, `addCrawlingTime()`). In addition, we developed the *Trust Feedbacks Collector* module to collect feedbacks directly from users in the form of history records and stored them in the *Trust Feedbacks Database*. Indeed, users typically have to establish their identities for the first time they attempt to use the platform through registering their credentials at the *Identity Management Service* (IdM) which stores the credentials in the *Trust*

2. http://cs.adelaide.edu.au/~cloudarmor
3. http://code.google.com/p/crawler4j/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2408613, IEEE Transactions on Parallel and Distributed Systems

Identity Registry. Moreover, we developed the *Identity Info Collector* module to collect the total number of established identities among the whole identity behavior (i.e., all established identities for users who gave feedbacks to a particular cloud service).

The Trust Assessment Function. This function is responsible for handling trust assessment requests from users where the trustworthiness of cloud services are compared and the factors of trust feedbacks are calculated (i.e., the credibility factors). We developed the Factors Calculator for attacks detection based on a set of factors (more details on how the credibility factors are calculated can be found in Section 4). Moreover, we developed the Trust Assessor to compare the trustworthiness of cloud services through requesting the aggregated factors weights from the Factors Calculator to weigh feedbacks and then calculate the mean of all feedbacks given to each cloud service. The trust results for each cloud service and the factors' weights for trust feedbacks are stored in the Trust Results and Factors Weights Storage.

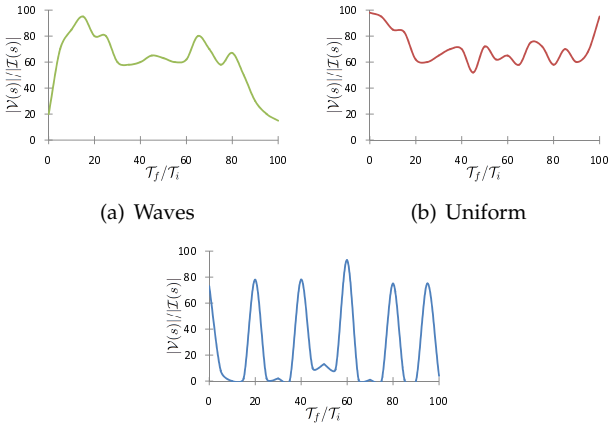
6.2 Experimental Evaluation

We particularly focused on validating and studying the robustness of the proposed credibility model against different malicious behaviors, namely collusion and Sybil attacks under several behaviors, as well as the performance of our availability model.

6.3 Credibility Model Experiments

We tested our credibility model using real-world trust feedbacks on cloud services. In particular, we crawled several review websites such as `cloud-computing.findthebest.com`, `cloudstorageprovidersreviews.com`, and `CloudHostingReviewer.com`, and where users give their feedbacks on cloud services that they have used. The collected data is represented in a tuple $\mathcal{H}$ where the feedback represents several QoS parameters as mentioned earlier in Section 3.2 and augmented with a set of credentials for each corresponding consumer. We managed to collect 10,076 feedbacks given by 6,982 users to 113 real-world cloud services. The collected dataset has been released to the research community via the project website.

For experimental purposes, the collected data was divided into six groups of cloud services, three of which were used to validate the credibility model against collusion attacks, and the other three groups were used to validate the model against Sybil attacks where each group consists of 100 users. Each cloud service group was used to represent a different attacking behavior model, namely: *Waves*, *Uniform* and *Peaks* as shown in Figure 3. The behavior models represent the total number of malicious feedbacks introduced in a particular time instance (e.g., $|\mathcal{V}(s)| = 60$ malicious feedbacks



(a) Waves (b) Uniform

(c) Peaks

Fig. 3. Attacking Behavior Models

when $\mathcal{T}_f = 40$, Figure 3(a)) when experimenting against collusion attacks. The behavior models also represent the total number of identities established by attackers in a period of time (e.g., $|\mathcal{I}(s)| = 78$ malicious identities when $\mathcal{T}_i = 20$, Figure 3(c)) where one malicious feedback is introduced per identity when experimenting against Sybil attacks. In collusion attacks, we simulated malicious feedback to increase trust results of cloud services (i.e., self-promoting attack) while in Sybil attacks we simulated malicious feedback to decrease trust results (i.e., slandering attack). To evaluate the robustness of our credibility model with respect to malicious behaviors (i.e., collusion and Sybil attacks), we used two experimental settings: I) measuring the robustness of the credibility model with a conventional model $\mathcal{C}on(s, t_0, t)$ (i.e., turning $\mathcal{C}_r(c, s, t_0, t)$ to 1 for all trust feedbacks), and II) measuring the performance of our model using two measures namely *precision* (i.e., how well TMS did in detecting attacks) and *recall* (i.e., how many detected attacks are actual attacks). In our experiments, TMS started rewarding cloud services that had been affected by malicious behaviors when the attacks percentage reached 25% (i.e., $e_t(s) = 25\%$), so the rewarding process would occur only when there was a significant damage in the trust result.

We conducted 12 experiments where six of which were conducted to evaluate the robustness of our credibility model against collusion attacks and the rest for Sybil attacks. Each experiment is denoted by a letter from A to F, as shown in Table 1.

TABLE 1
Behavior Experimental Design

Malicious Behaviors	Experimental Setting	Waves	Uniform	Peaks
Collusion	I	A	B	C
Attacks	II	A'	B'	C'
Sybil	I	D	E	F
Attacks	II	D'	E'	F'

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2408613, IEEE Transactions on Parallel and Distributed Systems

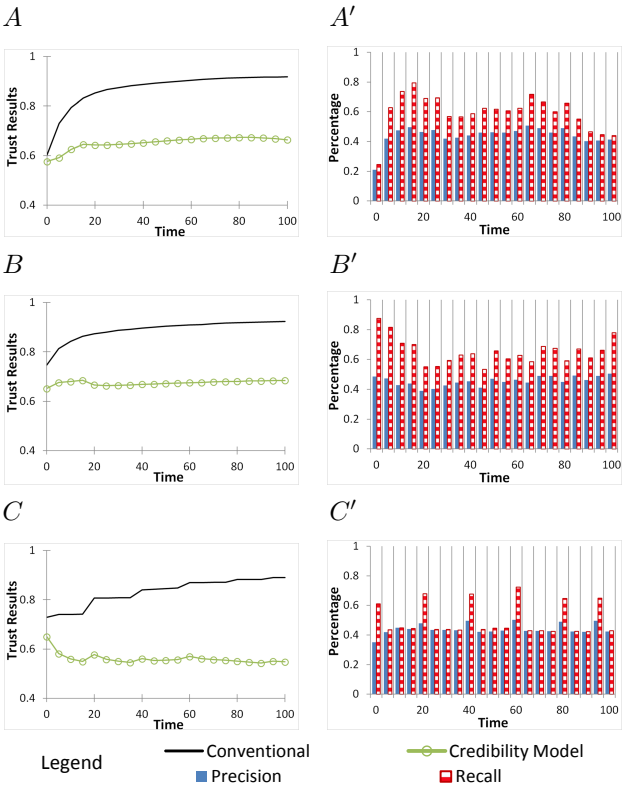


Fig. 4. Robustness Against Collusion Attacks

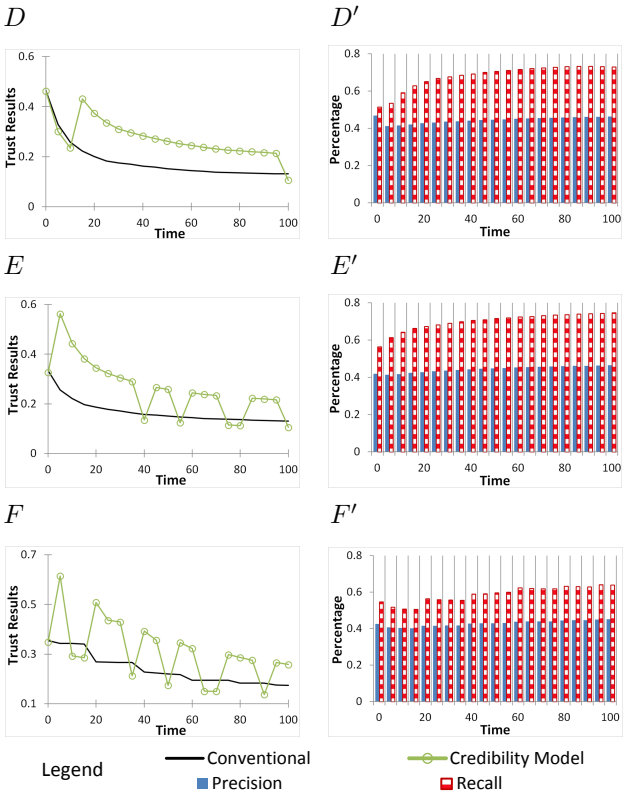


Fig. 5. Robustness Against Sybil Attacks

6.3.1 Robustness Against Collusion Attacks

For the collusion attacks, we simulated malicious users to increase trust results of cloud services (i.e., self-promoting attack) by giving feedback with the range of [0.8, 1.0]. Figure 4 depicts the analysis of six experiments which were conducted to evaluate the robustness of our model with respect to collusion attacks. In Figure 4, A, B, and C show the trust result for experimental setting I, while A', B', and C' depict the results for experimental setting II.

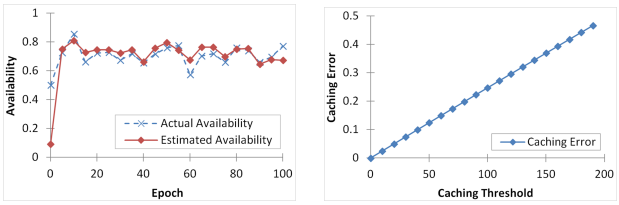
We note that the closer to 100 the time instance is, the higher the trust results are when when the trust is calculated using the conventional model. This happens because malicious users are giving misleading feedback to increase the trust result for the cloud service. On the other hand, the trust results show nearly no change when calculated using the proposed credibility model (Figure 4 A, B and C). This demonstrates that our credibility model is sensitive to collusion attacks and is able to detect such malicious behaviors. In addition, we can make an interesting observation that our credibility model gives the best results in precision when the $Uniform$ behavior model is used (i.e., 0.51, see Figure 4 B'), while the highest recall score is recorded when the $Waves$ behavior model is used (i.e., merely 0.9, see Figure 4 A'). Overall, recall scores are fairly high when all behavior models are used which indicate that most of the detected attacks are actual attacks. This means that our model can successfully detect collusion attacks (i.e., whether the attack is strategic such as in *Waves* and *Uniform* behavior models or occasional such as in the *Peaks* behavior model) and TMS is able to dilute the increased trust results from self-promoting attacks using the proposed credibility factors.

6.3.2 Robustness Against Sybil Attacks

For the Sybil attacks experiments, we simulated malicious users to decrease trust results of cloud services (i.e., slandering attack) by establishing multiple identities and giving one malicious feedback with the range of [0, 0.2] per identity. Figure 5 depicts the analysis of six experiments which were conducted to evaluate the robustness of our model with respect to Sybil attacks. In Figure 5, D, E, and F show the trust results for experimental setting I, while D', E', and F' depict the results for experimental setting II.

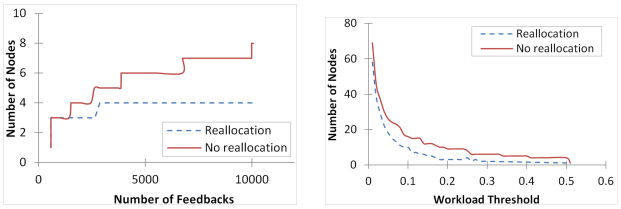
From Figure 5, we can observe that trust results obtained by using the conventional model decrease when the time instance becomes closer to 100. This is because of malicious users who are giving misleading feedback to decrease the trust result for the cloud service. On the other hand, trust results obtained by using our proposed credibility model are higher than the ones obtained by using the conventional model (Figure 5 D, E and F). This is because the cloud service was rewarded when the attacks occurred. We also can see some sharp drops in trust results obtained by considering our credibility model where the highest number of drops is recorded when the $Peaks$ behavior model is used (i.e., we can see 5 drops in Figure 5 F

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2408613, IEEE Transactions on Parallel and Distributed Systems



(a) Actual Availability VS. Estimated Availability (b) Trust Results Caching Error Rate

Fig. 6. Availability Prediction and Caching Accuracy



(a) Number of TMS Nodes VS. Feedbacks (b) Number of TMS Nodes VS. Workload Threshold

Fig. 7. Reallocation Performance

which actually matches the drops in the *Peaks* behavior model in Figure 3(c)). This happens because TMS will only reward the affected cloud services if the percentage of attacks during the same period of time has reached the threshold (i.e., which is set to 25% in this case). This means that TMS has rewarded the affected cloud service using the change rate of trust results factor. Moreover, from Figure 5 D', E' and F', we can see that our credibility model gives the best results in precision when the $Waves$ behavior model is used (i.e., 0.47, see Figure 4 D'), while the highest recall score is recorded when the $Uniform$ behavior model is used (i.e., 0.75, see Figure 4 A'). This indicates that our model can successfully detect Sybil attacks (i.e., either strategic attacks such as in *Waves* and *Uniform* behavior models or occasional attacks such as in the *Peaks* behavior model) and TMS is able to reward the affected cloud service using the change rate of trust results factor.

6.4 Availability Model Experiments

We tested our availability model using the same dataset we collected to validate the credibility model. However, for the availability experiments, we focused on validating the availability prediction accuracy, trust results caching accuracy, and reallocation performance of the availability model (i.e., to validate the three proposed algorithms including Particle Filtering based Algorithm, Trust Results & Credibility Weights Caching Algorithm, and Instances Management Algorithm).

6.4.1 Availability Prediction Accuracy

To measure the prediction accuracy of the availability model, we simulated 500 nodes hosting TMS instances and set the failure probability for the nodes as 3.5 percent, which complies with the findings in [31]. The motivation of this experiment is to study the estimation accuracy of our approach. We simulated TMS nodes' availability fluctuation and tracked their fluctuation of availability for 100 time steps (each time step counted as an *epoch*). The actual availability of TMS nodes and corresponding estimated availability using our particle filter approach were collected and compared. Figure 6(a) shows the result of one particular TMS node. From the figure, we can see that the estimated availability is very close to the actual availability of the TMS node. This means that our approach works well in tracing and predicting the availability of TMS nodes.

6.4.2 Trust Results Caching Accuracy

To measure thecaching accuracy of the availability model, we varied the caching threshold to identify the optimal number of new trust feedbacks that TMS received to recalculate the trust result for a particular cloud service without having a significant error in the trust results. The trust result caching accuracy is measured by estimating the root-mean-square error (RMSE) (denoted caching error) of the estimated trust result and the actual trust result of a particular cloud service. The lower the RMSE value means the higher accuracy in the trust result caching. Figure 6(b) shows the trust result caching accuracy of one particular cloud service. From the figure, we can see that the caching error increases almost linearly when the caching threshold increases. The results allow us to choose the optimal caching threshold based on an acceptable caching error rate. For example, if 10% is an acceptable error margin, the caching threshold can be set to 50 feedbacks. It is worth mentioning that the caching error was measured on real users' feedbacks on real-world cloud services.

6.4.3 Reallocation Performance

To validate the reallocation performance of the availability model, we used two experimental settings: I) comparing the number of TMS nodes when using the reallocation of trust feedbacks and without reallocation while increasing the number of feedbacks (i.e., when the workload threshold $e_w(s_{tms})$ = 25%); II) comparing the number of TMS nodes when using the reallocation of trust feedbacks and without reallocation while varying $e_w(s_{tms})$. The lower the number of TMS nodes, the more cost efficient TMS is. Figure 7(a) shows the results of experimental settings I. We can observe that the total number of TMS nodes when using the reallocation of trust feedbacks technique is fairly low and more stable than the total number of TMS nodes when reallocation is not used (i.e., even when the total number of feedbacks is high). Figure 7(b) shows the results of experimental settings II. From the figure, we can see that the higher the workload threshold the lower the number of TMS nodes. However, the number of TMS nodes when using the reallocation of trust feedbacks technique is lower than the number of TMS nodes when reallocation is not considered. This means that our approach has advantages in minimizing the bandwidth cost by reducing the total number of TMS nodes.

7 Related Work

Over the past few years, trust management has been one of the hot topics especially in the area of cloud computing [32], [14], [10]. Some of the research efforts use policy-based trust management techniques. For example, Ko et al. [33] propose TrustCloud framework for accountability and trust in cloud computing. In particular, TrustCloud consists of five layers including workflow, data, system, policies and laws, and regulations layers to address accountability in the cloud environment from all aspects. All of these layers maintain the cloud accountability life cycle which consists of seven phases including policy planning, sense and trace, logging, safe-keeping of logs, reporting and replaying, auditing, and optimizing and rectifying. Brandic et al. [7] propose a novel approach for compliance management in cloud environments to establish trust between different parties. The approach is developed using a centralized architecture and uses compliant management technique to establish trust between cloud service users and cloud service providers. Unlike previous works that use policy-based trust management techniques, we assess the trustworthiness of a cloud service using reputation-based trust management techniques. Reputation represents a high influence that cloud service users have over the trust management system [34], especially that the opinions of the various cloud service users can dramatically influence the reputation of a cloud service either positively or negatively.

Some research efforts also consider the reputation-based trust management techniques. For instance, Habib et al. [6] propose a multi-faceted Trust Management (TM) system architecture for cloud computing to help the cloud service users to identify trustworthy cloud service providers. In particular, the architecture models uncertainty of trust information collected from multiple sources using a set of Quality of Service (QoS) attributes such as security, latency, availability, and customer support. The architecture combines two different trust management techniques including reputation and recommendation where operators (e.g., AND, OR, NOT, FUSION, CONSENSUS, and DISCOUNTING) are used. Hwang et al. [4] propose a security aware cloud architecture that assesses the trust for both cloud service providers and cloud service users. To assess the trustworthiness of cloud service providers, the authors propose the trust negotiation approach and the data coloring (integration) using fuzzy logic techniques. To assess the trustworthiness of cloud service users, they develop the Distributed-Hash-Table (DHT)-based trust-overlay networks among several data centers to deploy a reputation-based trust management technique. Unlike previous works which do not consider the problem of unpredictable reputation attacks against cloud services, we present a credibility model that not only detects the misleading trust feedbacks from collusion and Sybil attacks, but also has the ability to adaptively adjust the trust results for cloud services that have been affected by malicious behaviors.

8 Conclusion

Given the highly dynamic, distributed, and non-transparent 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 reputation-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 reputation 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.

[9] T. H. Noor, Q. Z. Sheng, and A. Alfazi, "Reputation Attacks Detection for Effective Trust Assessment of Cloud Services," in *Proc. of TrustCom'13*, 2013.

[10] T. H. Noor, Q. Z. Sheng, S. Zeadally, and J. Yu, "Trust Management of Services in Cloud Environments: Obstacles and Solutions," *ACM Computing Surveys*, vol. 46, no. 1, pp. 12:1–12:30, 2013.

[11] S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising From Cloud Computing," in *Proc. CloudCom'10*, 2010.

[12] E. Bertino, F. Paci, R. Ferrini, and N. Shang, "Privacy-preserving Digital Identity Management for Cloud Computing," *IEEE Data Eng. Bull*, vol. 32, no. 1, pp. 21–27, 2009.

[13] E. Friedman, P. Resnick, and R. Sami, *Algorithmic Game Theory*. New York, USA: Cambridge University Press, 2007, ch. Manipulation-Resistant Reputation Systems, pp. 677–697.

[14] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

[15] F. Skopik, D. Schall, and S. Dustdar, "Start Trusting Strangers? Bootstrapping and Prediction of Trust," in *Proc. of WISE'09*, 2009.

[16] H. Guo, J. Huai, Y. Li, and T. Deng, "KAF: Kalman Filter Based Adaptive Maintenance for Dependability of Composite Services," in *Proc. of CAiSE'08*, 2008.

[17] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *Proc. of AINA'10*, 2010.

[18] Y. Wei and M. B. Blake, "Service-oriented Computing and Cloud Computing: Challenges and Opportunities," *Internet Computing, IEEE*, vol. 14, no. 6, pp. 72–75, 2010.

[19] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Sep 2011, accessed: 05/06/2012, Available at: http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition. pdf.

[20] O. David and C. Jaquet, "Trust and Identification in the Light of Virtual Persons," pp. 1–103, Jun 2009, accessed 10/3/2011, Available at: http://www.fidis.net/resources/deliverables/identity-of-identity/.

[21] B. Fung, K. Wang, R. Chen, and P. Yu, "Privacy-preserving Data Publishing: A Survey of Recent Developments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 1–53, 2010.

[22] J. R. Douceur, "The Sybil Attack," in *Proc. of IPTPS'02*, 2002.

[23] S. Ba and P. Pavlou, "Evidence of the Effect of Trust Building Technology in Electronic Markets: Price Premiums and Buyer Behavior," *MIS Quarterly*, vol. 26, no. 3, pp. 243–268, 2002.

[24] K. Lai, M. Feldman, I. Stoica, and J. Chuang, "Incentives for Cooperation in Peer-to-Peer Networks," in *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[25] L. Xiong and L. Liu, "Peertrust: Supporting Reputation-based Trust for Peer-to-Peer Electronic Communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.

[26] A. Birolini, *Reliability Engineering: Theory and Practice*. Springer, 2010.

[27] S. Maskell and N. Gordon, "A Tutorial on Particle Filters for On-line Nonlinear/Non-Gaussian Bayesian Tracking," in *Target Tracking: Algorithms and Applications (Ref. No. 2001/174), IEEE*. IET, 2001, pp. 2–1.

[28] T. H. Noor and Q. Z. Sheng, "Trust as a Service: A Framework for Trust Management in Cloud Environments," in *Proc. of WISE'11*, 2011.

[29] T. H. Noor, Q. Z. Sheng, A. H. Ngu, A. Alfazi, and J. Law, "CloudArmor: A Platform for Credibility-based Trust Management of Cloud Services," in *Proc. of CIKM'13*, 2013.

[30] T. Noor and Q. Z. Sheng, "Credibility-Based Trust Management for Services in Cloud Environments," in *Proc. of ICSOC'11*, 2011.

[31] S. M. Kim and M.-C. Rosu, "A Survey of Public Web Services," in *Proc. of WWW04*, 2004.

[32] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A Survey of Attack and Defense Techniques for Reputation Systems," *ACM Computing Surveys*, vol. 42, no. 1, pp. 1–31, 2009.

[33] R. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B. Lee, "TrustCloud: A Framework for Accountability and Trust in Cloud Computing," in *Proc. SERVICES'11*, 2011.

[34] C. Dellarocas, "The Digitization of Word of Mouth: Promise and Challenges of Online Feedback Mechanisms," *Management Science*, vol. 49, no. 10, pp. 1407–1424, 2003.

Talal H. Noor is an assistant professor in the College of Computer Science and Engineering at Taibah University in Yanbu, Saudi Arabia. He received his Ph.D. degree in Computer Science from the University of Adelaide in 2013. His research interests include cloud computing, service-oriented computing, security and privacy, and trust management.

Quan Z. Sheng is an associate professor at School of Computer Science, the University of Adelaide. He received the PhD degree in computer science from the University of New South Wales and B.E from Beihang University. His research interests include service-oriented computing, Web science, distributed computing, pervasive computing, and Web of Things. He is the recipient of ARC Future Fellowship in 2014, Chris Wallace Award in 2012, and Microsoft Research Fellowship in 2003. He is the author of more than 180 publications.

Lina Yao is currently a PostDoc and an associate lecturer at School of Computer Science, the University of Adelaide. She received PhD and M.Sc, both in Computer Science, from the University of Adelaide and B.E from Shandong University. Her research interests include Web mining, Internet of Things, Ubiquitous computing and Service Oriented Computing.

Schahram Dustdar is Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the Vienna University of Technology. He is a member of the Academia Europaea: The Academy of Europe, Informatics Section (since 2013), recipient of the ACM Distinguished Scientist award (2009), and the IBM Faculty Award (2012). He is an Associate Editor of IEEE Transactions on Services Computing, ACM Transactions on the Web, and ACM Transactions on Internet Technology and on the editorial board of IEEE Internet Computing. He is the Editor-in-Chief of Computing (an SCI-ranked journal of Springer).

Anne H.H. Ngu is currently a full Professor with the Department of Computer Science at Texas State University-San Marcos. From 1992-2000, she worked as a Senior Lecturer in the School of Computer Science and Engineering, University of New South Wales (UNSW), Australia. She has held research scientist positions with Telecordia Technologies and Microelectonics and Computer Technology (MCC). She was a summer faculty scholar at Lawrence Livermore National Laboratory from 2003-2006. Her main research interests are in large-scale discovery and integration of information services, scientific and business process automation, agent systems and Internet of Things.