Integrating a Market-Based Model in Trust-Based Service Systems

Suronapee Phoomvuthisarn, Yan Liu*, Liming Zhu, and Ross Jeffery

Abstract: The reputation-based trust mechanism is a way to assess the trustworthiness of offered services, based on the feedback obtained from their users. In the absence of appropriate safeguards, service users can still manipulate this feedback. Auction mechanisms have already addressed the problem of manipulation by market-trading participants. When auction mechanisms are applied to trust systems, their interaction with the trust systems and associated overhead need to be quantitatively evaluated. This paper proposes two distributed architectures based on centralized and hybrid computing for integrating an auction mechanism with the trust systems. The empirical evaluation demonstrates how the architectures help to discourage users from giving untruthful feedback and reduce the overhead costs of the auction mechanisms.

Key words: trust systems; software architecture; economic mechanisms

1 Introduction

Service oriented applications are highly distributed and loosely coupled with less central authority over participating for services. These services have a high degree of autonomy, and hence most of them can arbitrarily claim service properties such as Quality of Service (QoS) in order to attract service users. This poses a number of risks to service users, who might end up selecting a poor-quality or even malicious service to cooperate. The Reputation-Based Trust mechanism (hereafter, the RBT) is one of the trust mechanisms commonly used for minimizing the risks of interactions between participating services[1, 2]. This mechanism assumes that individual service users (or “raters” — service users that rationally provide ratings to other services) can leave feedback such as ratings on services that they have previously consumed[1]. These ratings can be accumulated into a meaningful reputation to assess the trustworthiness of offered services.

There is, however, evidence that such feedback can be manipulated[1, 3-5]. Some service users lie in their feedback intentionally to gain benefits from the biased reputation that they establish, thus compromising the reputation produced by the RBT.

Economic mechanisms such as auctions are capable of addressing the problem of cheating behavior, even when the majority of market-trading participants lie. Unlike other approaches for preventing cheating, the auction mechanisms’ aim is to make sure lie does not gain, where the measure of gain for each bidder is independent of the majority of bids from others[6]. This property enforces each participant to report the truth no matter how others act, and discourages dishonest raters from gaining benefits even when they are in a majority. Recognizing the merits of auction mechanisms, this paper incorporates them into a trust negotiation protocol. The aim is to provide the extra capability for trust systems to discourage raters from cheating when they provide ratings to other services.

Supporting these additional auction mechanisms
within trust-based service systems necessitates a suitable distributed architecture for integration. This architecture ensures the following four trust systems’ quality attributes:

- **Extensibility.** The trust system can keep their normal functionalities without incurring unaffordable overhead when integrating with the auction mechanism;
- **Scalability.** The overall architecture is highly scalable when leveraging the auction mechanism that was originally designed for economics of a centralized nature;
- **Decentralization.** The system can capture the truth-telling property of the auction mechanism in distributed environments;
- **Performance.** The architecture ensures an affordable response time.

Therefore, an architecture design to address above challenges needs to be rigorously evaluated.

This paper presents two architecture implementations for integrating an auction mechanism with trust systems. The first architecture embeds the auction mechanism within a trust system that coordinates all the raters with a central coordinator. To make the architecture further scalable, a hybrid architecture is designed to place the auction mechanism at distributed services. Without a central coordinator, each service needs to calculate and share its trust ratings to others. The quality of each architecture design is evaluated according to above four quality attributes of extensibility, scalability, decentralization, and performance. The contribution of this paper is twofold. First, we provide an architectural approach to integrating auction mechanisms with trust systems to discourage raters from providing untruthful ratings to other services, even in scenarios where the majority of them are dishonest. Second, we evaluate this approach with two architecture designs and compare quality attributes with empirical experiments.

## 2 Background

### 2.1 Economic mechanisms

Economics is a social science that studies how humans make choices on allocating scarce resources to satisfy their unlimited wants\[^7\]. Its mechanisms designed for marketplaces have already addressed the issue of discouraging market-trading participants from cheating behavior\[^8\]. One economic mechanism that yields truth-telling properties of market-trading participants is the Vickrey Auction Mechanism (hereafter, the VAM)\[^9\]. The VAM’s rules are simple. Bidders place their bids for a particular item and hand them sealed to the auctioneer. The winner of the auction is the one who places the highest bid and pays the exact amount of the second highest bid. After the auction, the reward and punishment are conducted based on the utility gain (i.e., the monetary gain after the auction) computed for each bidder. The utility gain represented as money for the winner is calculated by the difference between his or her true valuation for the item and the second highest bid in the auction, while the utility for other bidders is zero as follows:

\[
    u_i = \begin{cases} 
    v_i - \text{Max}_j \neq_i (b_j), & \text{if } b_i > \text{Max}_j \neq_i (b_j); \\
    0, & \text{otherwise} 
    \end{cases}
\]

The VAM principle ensures that reporting the true valuation maximizes the utility\[^9\]. Bidders who submit untruthful bids will not gain, and are sometimes penalized. The property of the VAM encourages a bidder to reveal the truth no matter how others act\[^8\]. Also, the VAM’s costs of computation are minimal because the modeling of participants’ behavior is simple and easy to implement. Furthermore, it does not require complex knowledge of how other participants behave. This lightweight nature of the VAM is promising to satisfy the above four quality attributes required in the integration architecture.

In this paper, besides the VAM, we also incorporate the Reverse Vickrey Auction Mechanism (hereafter, the RVAM) as a solution to the unfairness nature of the VAM. This generally happens when some raters who submit a very low but truthful bid have less chance to win the auction. The measure of gain based on the RVAM is in a reverse order between the true valuation of the winning bidder who submits the lowest bid and the second lowest bid in the auction.

### 2.2 Architectural challenges

The VAM is feasible to discourage raters from cheating in trust-based domains; however, integrating the VAM within trust systems poses architectural challenges on balancing the quality attributes of extensibility, scalability, decentralization, and performance. From the architectural point of view, there are four concerns raised as follows.

#### 2.2.1 Extensibility

The first aspect is to achieve good separation of concerns. Auctioning messages incurred by the VAM
might introduce tight couplings between the trust components and the VAM components, thus degrading the extensibility of the trust system itself. Trust and the VAM components should reduce the overlap in functionality so that modifications to the trust components or replacement of the VAM components can be maintained without reengineering other parts of the architecture.

Besides, the VAM’s key characteristics need to be captured in the deployed trust-based scenario. Each rater has preferences to choose strategies that maximize its utilities. Hence, the architecture should handle varying preferences for all strategies to ensure that submitting unfair ratings cannot provide benefits to unfair raters. In addition, the resource each participant puts into the auction depends on each strategy the participant performs. This requires the relations between the strategies and their required resources captured by relevant trust components and the VAM components in the architecture.

2.2.2 Scalability
The original VAM auction mechanism assumes a centralized environment in which a central coordinator takes care of all auctioning computation. When integrated with the trust system, a centralized architecture provides a common interface by which raters can issue their bids through auctions. Such a system therefore has limitations to scale and handle the increased load of raters. This centralized infrastructure is not suitable for ultra-large scale service environment where the number of potential services is huge. This necessitates a distributed trust system that can handle bids from raters independently so that raters are not blocked while their bids are executed in the auctioning process.

2.2.3 Decentralization
To achieve better scalability, the VAM can be embedded with distributed services without a central coordinator. Consequently, each service needs to calculate and share their information to others. However, capturing the VAM’s truth-telling property in this environment is not straightforward. Traditional VAM is suitable for a class of problems where bidders’ budgets are unlimited\cite{10}. Without central authority, a distributed service environment lacks the overall knowledge of individual’s budgets. Hence, services may take an underbidding strategy to gain remote resources. As a result, the VAM’s property to prevent lying may be broken. Moreover, reporting the VAM outcome is not trivial since the results need to be distributed to dynamic raters, which is not a known priori, or may even change over time as a service joins or leaves. Obviously, explicit polling by the VAM for notification incurs communication overhead. All lead to the challenge of capturing the VAM’s truth-telling property in decentralized architecture.

2.2.4 Performance
The last aspect is to reduce overhead as discussed in the above challenge of decentralization. When the number of participating services involved is very high, the volume of messages exchanged between the VAM and trust components might incur significant overhead. This imposes a further research question in the architectural design: how to leverage the VAM’s benefits to prevent raters from cheating, while reducing the end-to-end delay incurred by the VAM messages. The resulting architecture should be optimized to reduce this traffic overhead significantly.

3 The Architecture
This section presents two VAM-based architectures to discourage raters from cheating, namely centralized and hybrid computing architecture. These VAM-based architectures are demonstrated through two case studies. Each case has its own emphasis on specific quality attributes. They are then compared to find optimal architectural solutions. The use of case studies provides insights on how these architectures can be extended with the VAM.

3.1 Centralized architecture
This centralized architecture relies on the system’s trust manager embedding with the VAM’s capabilities to help a requester find a set of potential providing services that have high reputations so as to assist the choice of their cooperation.

3.1.1 Case description
The case follows the supply chain scenario, where a customer service (acting as a requester) intends to find potential providing services in the line of supply chain including retailer, warehouse, and manufacturer services to negotiate. To complete this line of business, the customer has to submit an order consisting of line items to a retailer service. Each line item identifies
a product and the corresponding quantity to be ordered. To fulfill orders, the retailer goes through each line item and finds a warehouse service with sufficient stock to ship them. The warehouse then ships the line item to the customer.

To manage stock levels in the warehouse, each warehouse needs to restock from the inventory of a relevant manufacturer service whenever its inventory levels fall below the minimum of inventory levels for a particular product. To choose the best deal available among all the providing services (i.e., retailer, warehouse, and manufacturer services) that have similar functions, the requesting service (i.e., customer, chosen retailer, and chosen warehouse service) sends a trust system the request asking for the reputation of relevant providers on how well they can provide their QoS information as claimed. Since each service is independent of the services or resources it provides[11], the interactions between participating services have the risks of failure, such as time delay and not always accessible within the promised timeline. Therefore, the overall performance QoS attributes, including response time, waiting time, and availability, can be used in this scenario.

The VAM comes into play in this context to motivate each rater to reveal the rating faithfully. In the case of this supply chain, the customer service invokes the central trust system to evaluate each retailer’s trustworthiness before sending the list of line items. Once having taken a decision for choosing one retailer, the chosen retailer then invokes the trust system to evaluate each warehouse’s trustworthiness before ordering a shipment from one warehouse. If the chosen warehouse’s stock is needed to be refilled, the warehouse then invokes the trust system to evaluate each manufacturer’s trustworthiness before ordering a product from the manufacturer.

3.1.2 Architecture layers

Conceptually, the architecture of a trust system has three layers to compose the key components, namely service layer, trust layer, and service metadata layer as depicted in Fig. 1.

The service layer performs bootstrapping, which receives ratings from raters. The trust layer prevents accumulation of unfair ratings. This layer consists of the trust engine to calculate trust-related information, such as a trust level of certain services, and the reputation engine to provide a robust reputation. The service metadata layer contains the service registry to support the VAM’s reward and punishment process made to raters. The registry is extended to capture the reputation of registered services. In addition, a database is used to store trust-related information such as trust parameters.

3.1.3 Auction-based trust negotiation protocol

The steps of the VAM are embedded in the trust negotiation protocol, to guide the interactions between key components across three service layers.

At Stage one, interrogation, a requester at the service layer finds providing services that meet its functional requirements. The requester checks the trust level of the discovered providers through Stage two, negotiation. Once receiving the requests from the trust engine for calculating the trust level of a provider, the reasoning manager then instructs the auction engine to perform the auction-based calculation. The calculation steps are shown in Fig. 2.

1. Auction engine initiates a new auction round (one auction per each provider requested) by setting an auction time and aggregates ratings from raters. The auction engine encapsulates the auction logic in the VAM component, which computes the utility gain of participating raters based on Eq. (1) (Section 2.1). This utility gain is input to the reasoning manager to make the decision on a reward to fair raters or some punishment to unfair raters. To calculate this utility, in addition to the second highest bid captured in the auction, the true value of the rating submitted by the winning rater is required (see Eq. (1)).

2. Auction engine terminates the auction.

3. Reasoning manager sends all values of ratings already excluded to the auction engine.

4. Auction engine finds the winning rater and instructs the controller to monitor the winner’s true
value. In this paper the term “the winning rater” refers to the raters who submit the highest rating (in the VAM process) and the lowest rating (in the RVAM process) in the auction.

(5) **Controller** monitors the winning rater’s true value.

(6) **Auction engine** instructs the VAM component to perform utility computation.

(7) **Auction engine** then announces the winning rater and its utility through the reasoning manager.

(8) **Reasoning manager** performs the VAM’s reward and punishment process and updates the newly raters’ reputation to the service registry.

(9) **Reasoning manager** then calculates the provider’s reputation.

The **trust engine** then uses the provider’s reputation produced by the reasoning manager to calculate the provider’s trust level. Once the computation is completed, the calculated trust level is then returned to the requester to determine whether each provider’s trust level exceeds the requester’s minimal trust threshold to further negotiate.

At Stage three, **interaction**, the requester negotiates with a chosen provider to interact with. The requester and the chosen provider use the trust system as a mediator to establish the trust negotiation between them. At Stage four, **termination**, once completely consuming the chosen provider’s service, the requester ends trust negotiation with the chosen provider.

### 3.2 Hybrid architecture

In the previous case study, the VAM-based centralized architecture makes all the decisions in one place. This poses a **scalability** issue when concurrent raters are participating with the central trust system at the same time. Simply decentralizing the architectural components in the centralized architecture to each participating service cannot solve this scalability issue, since query messages incurred by the auction-based trust negotiation protocol might induce a large volume of traffic overhead due to the end-to-end delay. Our solution is to the combination of them into a hybrid computing architecture.

#### 3.2.1 Case description

The supply chain scenario is further extended to explore the hybrid computing architecture. In Fig. 3, each of registry services maintains the centralized architecture presented in Section 3.1. Each registry service has its service indexes and implements the VAM protocol in the trust manager.

Each service registry acts as a broker and communicates with each other in a decentralized manner to find potential providers (i.e., retailer, warehouse, and manufacturer services) for a requester. After receiving a list of relevant providers from registries’ peers, the registry invokes its trust manager to evaluate the trustworthiness of these providers.

#### 3.2.2 Architecture layers

Figure 4 shows the architecture of hybrid computing. It mainly consists of a set of service registries, each of which acts as a central server in resolving a query message for a requesting service. These service registries maintain a set of their own services registered to the trust system using indexing scheme in the
service description database, and communicate with each other in a P2P manner. In this nature, propagation of messages between individual services only occurs within a small population of service registries, which help to reduce the messages communicated by the VAM protocol.

Consequently, each service’s registry component communicates with other services across four service layers, namely application layer, queuing layer, trust layer, and service metadata layer.

The application layer handles an incoming request from requesting services as well as an outgoing request from a service registry. It interacts with the registry’s own registered services and other peers of the registry in resolving the query message of a requester. The layer also receives ratings submitted by raters in the auctioning process through the discovery manager.

The queuing layer aims to reduce a runtime overhead incurred by interchange messages passing between the application layer and the trust layer. It consists of (1) the service caching to previously store evaluated results of a requester’s incoming request, (2) the queue list to store the ratings of raters when the application cannot concurrently process them, and (3) the scheduler engine to schedule the execution of these ratings according to a specific set of constraints.

The trust layer incorporates components to prevent accumulation of unfair ratings. This layer consists of two key components: (1) the trust manager from the centralized architecture is reused to produce a trust level or reputation of providing services accumulated from truthful ratings in the VAM process; while (2) the matchmaking manager is a broker for a registry to interact with the discovery manager, the service description database, and the trust manager for evaluating the trustworthiness of a providing service.

The service metadata layer stores trust data in the system. The repositories, including trust repository and reputation repository are reused from the centralized architecture. Since each service registry interacts with
each other in a decentralized manner, the reputation repositories consist of both (1) the local reputation repository to store the reputation of raters in which the registry itself is responsible for, and (2) the global reputation repository to store the reputation of other raters organized by its set of registries’ trusted neighbors (registries’ peers). The layer also contains a service description database to store service information indexed when services register to the system.

To propagate changes among shared data (e.g., raters’ reputation credits), the proxy offers an observer-based change notification mechanism in which the global reputation repository is the subject and each registry’s trust manager as the observer. The proxy can trigger the trust manager of each registry to be notified about the changes of raters’ budget when they participate in multiple auctions through its own reasoning manager.

### 3.2.3 Auction-based trust negotiation protocol

The auction-based trust negotiation protocol consists of the following five stages: **initiation**, **matching**, **interrogation**, **negotiation**, and **interaction**. This protocol interacts between a requester and a service registry to choose a provider for the requester with which to cooperate as shown in Fig. 5.

At Stage one, **initiation**, the discovery manager receives a requester’s query message, and then instructs the matchmaking manager to resolve the request for a relevant provider.

At Stage two, **matching**, when the matchmaking manager receives the query message, it first checks its local cache for relevant providing services that can meet the requester’s requirements in the query message. If the number of matching providers is not enough for the requester, it then forwards the query message to other registries’ peers. Once it has received a sufficient number of relevant providers, the matchmaking manager then keeps the list of these providers for evaluating their trustworthiness.

At Stage three, **interrogation**, after receiving a list of relevant providers, if the service registry has enough experience with those providers, it can choose one or a set of the providers that it trusts most for the requester. Otherwise, the service registry requests raters’ opinions about the reputation of these providers. To compute these reputations, the reasoning manager instructs the auction engine to initialize auction services to gather all values of ratings from raters. The auction-based calculation follows the similar steps of the protocol in the centralized architecture. After calculating the reputations of the potential providers, the trust engine then uses these reputations to calculate their trust level, which is used to determine whether each provider’s trust level exceeds the requester’s trust threshold for further negotiation.

At Stage four, **negotiation**, the requester then establishes a trust negotiation with the chosen provider directly to get its service. At Stage five, **interaction**, after the trust negotiation is established, the requester then gets the chosen provider’s service.

![Fig. 5 Trust negotiation protocol in hybrid architecture.](image-url)
4 Architecture Deployment

In order to realize the VAM-based architectures, this paper uses JXTA (2003), an open source peer-to-peer protocol specification initiated by Sun Microsystems, as a platform for web service publication and discovery infrastructure. Compared to other distributed web service discovery, JXTA provides service group concurrent searching by aggregating a search result of each service group, thus its architecture is suitable for web service discovery and selection process that requires highly efficient searching[12]. The JXTA network (Sun Microsystems, 2003) is built out of five key abstractions—uniform peer ID addressing, peer groups, advertisements, resolver, and pipes—that provide a generic infrastructure to deploy P2P services and applications. The deployment of the VAM-based architectures is as follow.

4.1 Centralized architecture

The architecture deployment is shown in Fig. 6. All services including the all raters and all services are deployed as JXTA edge nodes hosted by the Apache AXIS 1.0 Web Server. All raters and retailer, warehouse, and manufacturer services interact with the trust-based service application via the Web service deployed in the IIS 5.0 Web Server that receives service requests and sends responses from the application.

The trust engine and the reputation engine are developed as Java EJBs and deployed as the single trust-based application hosted by the Tomcat Application Server. This Application Server processes requests from the Web Server and sends responses back to the Web Server.

The Application Server implements the auction-based trust negotiation protocol. It communicates with services and the service registry using SOAP messages and connects to the SQL Database Server using the JDBC driver. The service registry implements LDAP components to support service registration.

4.2 Hybrid architecture

The architecture deployment is shown in Fig. 7. All services including customers, retailers, warehouses, manufacturers are built using JXTA relay nodes with 3 friends per peer. They are randomly assigned to one of the service registry deployed as rendezvous nodes with each having its own trust manager. With a rendezvous capability, a service registry can maintain an index of advertisement published by providing services to discovery requests using a Shared Resource Distributed Index (SRDI) (Sun Microsystems, 2003). These service registries keep this random list of their registry’s peers ranging from 1 to 10 peers and maintain a maximum number of 50 registered services per registry. These registries interact with all raters through auctions processed by their associated trust managers that receive ratings or any service requests.

The components of the trust managers and the Application Server are implemented and deployed the same way as the centralized architecture.

5 Evaluation

The architectural framework presented in Section 3 is now applied to three test cases as follows:

- **The VAM property test** Unfair raters should get a penalty, especially when the majority of them lie about their ratings.

- **The performance test** evaluates overhead of the devised architecture integrated with the VAM. The test implements two prototypes with and without the
VAM deployed and observes the overhead cost for each deployment.

- **The scalability test** evaluates the extent to which the VAM-based architectures can scale in terms of interacting raters. The test is to compare the response time of two architectures (centralized vs hybrid) when the number of interacting raters increases.

### 5.1 Test setup

The test environment includes two identical Windows XP machines with 3 GHz Core 2 Duo processors and 3.25 GB of RAM. One is used for hosting all raters’ services, and the other hosts the rest of the application.

The auction process initially involves 50 raters for each case study. The raters are grouped into two groups: fair and unfair raters. A fair rater offers QoS ratings to a provider just as it is perceived while an unfair rater randomly offers QoS ratings above or below what it perceived. To simplify the problem of cheating by raters, the approach of this paper considers the case where all raters have their past experience with a certain provider with 70% probability. Each of them has to rate each provider with 50% probability.

At the end of each auction, the reputation of each participating rater is updated based on the utility gain/loss calculated by Eq. (1). To update the reputation of a rater with the calculated utility gain, the rater’s newly updated reputation is calculated by accumulating the rater’s utility gain with the rater’s current reputation. For example, if a rater, whose reputation is 20 credits, gets a positive 0.6 utility gain, the newly updated reputation is 20.6 credits. The reputation of unfair raters is initially set to any random numbers between 5 to 15 credits while for the fair raters it is between 10 to 20 credits based on 100 transactions previously conducted. The minimal credits of raters to participate in auctions are initially set as 5 credits.

In the case of centralized architecture, the newly updated reputation of a rater can be directly stored in the centralized service registry. However, to make these newly updated reputations publicly known to others in the hybrid architecture, a Distributed Hash Tables (DHT) technique is applied to store an index of raters’ reputation in each service registry’s peers. This DHT method uses multiple hash functions to map a single rater’s ID to corresponding trusted neighbors that calculate and store the reputation of raters individually. DHT also provides a basic operation for retrieving the reputation of raters. When services need the reputation of raters, they can retrieve them using DHT operation with their ID as a parameter.

In a service selection phase, after completing the calculations of relevant providers’ reputation, the trust system then ranks these providers based on the requester’s preferences. At the end, the requester will get a sorted list of potential providers with which to cooperate.

To select some of these providers for the requester, the trust system ranks them based on the overall score taking into account both (1) QoS score published by the providers in service registries, and (2) reputation score of the providers calculated by trust systems. Both parameters’ thresholds are specified in the service discovery request by the requester. The details of each parameter are in the following.

1. The QoS score is an average real number in [0,1] specified by a provider when it publishes its QoS information to a service registry. In this paper, service providers specify QoS information by using the concept of embedded tModel in WSDL files.

2. The reputation score is a measure of a provider’s trustworthiness. It can be computed by collecting ratings from other services that have previously interacted with the provider. The reputation of one provider can be calculated by aggregating its ratings submitted by raters into one percentage measure Reputation in [0,1], each of which is weighted by raters’ reputation as shown in Eq. (2).

   \[
   \text{Reputation} = \sum_{i=1}^{n} w_i \times \text{rating}_i \quad (2)
   \]

   where \( w_i \) represents a weighted reputation in [0,1] of a rater, \( \text{rating}_i \) represents a rating rated by a rater \( i \), and \( n \) represents a number of raters.

After taking into account both parameters, the overall score of a provider in the ranking process can be calculated as shown in Eq. (3) below.

\[
\text{Overall score} = \alpha \times \text{QoS score} + \beta \times \text{Reputation} \quad (3)
\]

where \( \alpha \) and \( \beta \) represent weights in [0,1], which is subjectively determined by the requesters depending on how important each source is (\( \alpha + \beta = 1.0 \)).

Based on the requesters’ preferences, the \( N \) numbers of providing services that satisfy the requesters’ threshold are then returned in descending order based on the overall score. Services whose overall score exceeds the trust level of 0.7, further proceed in negotiate.
5.2 VAM property test
Unfair raters should get a penalty, especially when the majority of them are dishonest. This can be measured by the average reputation of unfair raters when they constantly provide unfairly high or low ratings to potential providing services.

5.2.1 Centralized architecture
The number of unfair raters is varied from 10% to 90%, with 10% increment per experiment. Each experiment involves 50 raters, each of which rates QoS ratings of providers based on given probabilities in Section 4. A total of 10, 300, and 500 auction rounds (equal to the number of the targeted providing services being rated) have been executed to observe the reputation changes incurred by cheating behavior of raters.

Figures 8 and 9 show that the average reputation of unfair raters decreases when the number of raters who lie increases under two different strategies. The main difference is that when unfair raters perform an overbidding strategy, their average reputation is decreasing much more when compared to the random bidding strategy. This is because in the overbid case, the unfair raters who rate others bids very high have more chance to win the auction compared to the unfair raters in the random bid case. However, their gains of reputation are impaired by the VAM’s punishment process made to the dishonest raters who submits the highest QoS rating. In contrast, we can see that the average reputation of fair raters from both the overbid and random bid case increases as the result of the reward granted by the VAM process.

In the underbid case, an unfair rater issues its bid lower than it perceived. Before applying the RVAM’s utility functions with the VAM’s calculation logic, the average reputation of unfair raters are not getting much decreased when they issue their bids lower than they actually perceived as shown in Fig. 10.

This is because the unfair raters who rate others bids very low have less chance to win the auction in the case that they are in the minority. However, after applying the RVAM’s utility functions, their gains of reputations are degraded due to the VAM’s punishment process made to the dishonest raters who submits the lowest QoS rating as depicted in Fig. 11.

The results clearly demonstrate that the VAM helps preventing cheating behavior by ensuring that unfair raters get a penalty when they simply lie. This is evident by the significant decrease in the average reputation of unfair raters, especially when the majority of them are dishonest. Also, the approach promotes a direct
incentive for fair raters participating in an auction due to the increasing of the average reputation of fair raters when they give ratings truthfully in the auction.

5.2.2 Hybrid architecture
The number of unfair raters varies from 10% to 90%, with 10% increment per experiment. Each experiment involves 100,000 raters, each of which rates QoS ratings of providing services including retailers, warehouse, and manufacturers requested by requesting services based on given probabilities in Section 5.1. The same total of auction rounds (10, 300, and 500 equal to the number of the providers rated) have been executed to observe the effect of changes in the average reputation of raters. The providing services were randomly grouped into 10 service registries, each of which can contain the maximum of 50 providing services.

The results of the overbid case and the random bid case can discourage cheating by raters (the same as in the case of centralized computing). However, in the case of the underbid, unfair raters have slight gains of reputation as depicted in Fig. 12. After integrating with RVAM to detect the underbidding strategy of unfair raters, the results demonstrate that the average reputation of unfair raters decreases as depicted in Fig. 13.

The results demonstrate that integrating RVAM in the hybrid architecture discourages cheating behavior of raters by ensuring that unfair raters get a penalty when they simply lie in a distributed environment.

5.3 Performance and scalability test
In the case of centralized architecture, the computational overhead is measured in terms of CPU usage and memory usage as shown in Table 1. The result shows the resource usage of deploying VAM is comparable to the deployment without VAM.

We further measure the application’s response time (second) with and without the VAM deployed. By varying a number of raters from 50 to 1000, the experiments are performed with 10 concurrent auctions, each of which is conducted for one providing service being rated.

The results show in the centralized architecture, the response time is almost identical with/without VAM (see Fig. 14). The performance overhead with VAM is approximately 3.9% higher than without VAM, as the number of raters is up to 1000.

In the case of the hybrid architecture, the performance overhead with VAM is approximately 5.7% higher than without VAM for 10,000 raters (see Fig. 15).

<table>
<thead>
<tr>
<th>Table 1</th>
<th>CPU and Memory Usage with/without VAM for 1000 raters.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU usage (%)</td>
</tr>
<tr>
<td>With VAM</td>
<td>53</td>
</tr>
<tr>
<td>Without VAM</td>
<td>48</td>
</tr>
</tbody>
</table>

Fig. 12 Changes in the reputation of raters for the underbid case before integrating with RVAM.

Fig. 13 Changes in the reputation of raters for the underbid case after integrating with RVAM.

Fig. 14 Performance and scalability of centralized architecture.

Fig. 15 Performance and scalability of hybrid architecture.
5.4 Discussions

This paper aims to support trust-based services on a very large scale to prevent unfair raters from cheating, especially when they are in the majority, in the service selection process. Based on the evaluation, integrating VAM in the service architecture (centralized or hybrid) can prevent benefits to dishonest raters. Each has its own strengths and weaknesses that make it suitable for a certain service environment.

The centralized architecture is suitable for small-scale service systems where the number of raters is not very high. This architecture has a simple deployment since all of the processing is controlled in a central location. The hybrid architecture can scale to handle larger number of raters and service providers.

6 Related Work

There are two main problems for the trust systems in the use of the RBT. The first problem is that of unfair ratings. A number of research studies have proposed techniques to tackle the problem of unfair ratings flowing from raters’ cheating behavior. These include: (1) detective techniques\cite{2, 14-16}, which detect unfair ratings for predicting the trend of raters’ untruthful behavior with statistical measures (e.g., clustering); and (2) preventive techniques\cite{3-5}, such as the side payment schemes\cite{17}, which discourages raters from lying by giving them some kind of incentives (e.g., digital currency or credit), so that truthful reporting maximizes the raters’ expected revenue.

Our approach generally belongs to the category of preventive techniques. Preventive techniques whose effectiveness replies on the majority of the participants to be truthful are difficult to apply in the large scale service oriented computing. In service-oriented computing, participating services are loosely coupled and dynamic in nature\cite{18}, a large number of services are expected to grow very quickly in the future\cite{12, 18, 19}. Therefore the contribution of our approach is identifying a well-proven preventive mechanism that is suitable to integrate with the trust-based services even majority of participants are dishonest.

Most research work related to the RBT\cite{3-5, 14, 16, 17, 20, 21} has focused on devising a mechanism to solve the problems of raters’ cheating behavior. Only a few studies\cite{22-24} have investigated the overhead costs incurred by these proposed mechanisms when they are integrated with the original trust systems. One study\cite{23} evaluates the efficiency of its proposed mechanism in terms of the consumption of computing resources (i.e., CPU usage). Wang et al.\cite{22} also dealt with the overhead cost of integrated mechanisms within trust systems. The authors attempted to investigate the cost and efficiency of the trust systems where such mechanisms are applied. By generalizing individual small groups of networks, the authors quantitatively analyzed the behavior of complex networks based on the mathematical equations. Another study is from Zhao et al.\cite{24} that integrates the mechanism as a number of exchange messages between participants, realized in relevant layers in a message feedback protocol.

These approaches assume the existence of centralized party that maintains the digital currency or reputation scores of participants to enforce their mechanisms’ truth-telling properties through rewarding or charging. Therefore, the computation of such approaches is all centralized in the trust system\cite{25}. This imposes research questions concerning their extension to support service oriented computing, while achieving performance and scalability.

All of the above demonstrate the need for a proper architecture to integrate a preventive mechanism with existing trust systems. This architecture needs to maintain quality attributes of the trust system: extensibility, decentralization, scalability, and performance.

7 Conclusions

In this paper, we integrate an auction mechanism with trust systems to discourage raters from providing untruthful ratings to other services, even in scenarios where most of raters are dishonest. There are two main contributions in this paper. The first contribution is an extension to existing trust systems, adding the extra capability to prevent cheating behavior on the part of raters. The notion of the two architectures with the VAM-based auction mechanism induces an effective trust negotiation by preventing the trust systems from being exploited by raters, even if the majority of them lie about their ratings. Second, the two architectures developed support the integration of the VAM in term of optimized overhead costs regarding the quality attributes of: (1) extensibility in terms of reusing software components across two
architectural designs; (2) scalability in terms of large concurrent raters interacting with trust systems; (3) decentralization for capturing the VAM’s truth-telling property in distributed environments; and (4) no significant performance overhead due to the end-to-end delay incurred by a significant amount of messages in highly scalable trust systems.

The VAM can effectively discourage raters from submitting untruthful ratings. However, the VAM assumes that each bidder submits his or her bid independently. This assumption raises the issue of how far the VAM can be applied in situations where players can communicate with others outside of the game. Thus, a number of participating services (e.g., raters) can communicate with each other and thus manage their bids in an auction. Apart from bidding collusion, the VAM rule that bids are sealed might cause some difficulty in enforcing bid privacy in service oriented computing. In a common auction, some tricky sellers might adopt a strategy of tracing the bids of others before submitting their own bids in the auction. In this way, such a seller can manage his or her bid in order to gain benefits. Future work involves providing architectural solutions to solve these problems. Hence, in future work, the resulting architecture will be proposed to support protection against bidding collusion, as well as integrating some security mechanisms to protect bid privacy.

References


Suronapee Phoomvuthisarn is a lecturer in the Faculty of Information Sciences and Technology, Mahanakorn University of Technology, Bangkok, Thailand. He received his MS degree in information sciences from the University of Pittsburgh in 2007 and PhD in computer sciences and engineering from the University of New South Wales in 2011. His research interests include service economics, trust computing, and software architecture. He is currently the vice-dean for Public Relations and Student Affairs of the Faculty of Information Sciences and Technology at Mahanakorn University.

Yan Liu is an associate professor in the Faculty of Engineering and Computer Science, Concordia University, Canada. Her current research involves cloud computing, distributed systems, software architecture, and model driven development. She has extensive research experience leading projects in domains arranging from enterprise applications, to data intensive workflows, to energy systems and smart grids. She has published over sixty international journal and conference papers in these areas since 2004. She was a Senior Scientist at Pacific Northwest National Laboratory from 2009 to 2012. She received her PhD in computer science from University of Sydney, Australia in 2004. Until December 2009, she was a Senior Researcher and project leader at NICTA (National ICT Australia).

Liming Zhu is a principal researcher at NICTA. NICTA is Australia’s centre of excellence for Information and Communications Technology R&D. He also holds conjoint positions at University of New South Wales (UNSW) and University of Sydney. He received his PhD in software engineering from UNSW in 2007. His current research interests include dependable systems and operation, software architecture, and software engineering.

Ross Jeffery is Emeritus Professor of Software Engineering in the School of Computer Science and Engineering at UNSW and Research Leader at NICTA. He received his PhD in software engineering from the University of New South Wales. His current research interests are in software engineering process and product modeling and improvement, electronic process guides and software knowledge management, software quality, software metrics, software technical and management reviews, and software resource modeling and estimation. His research has involved many government and industry organizations over a period of 30 years and has been funded from industry, government, and universities. He has co-authored four books and over one hundred and thirty research papers. He has served on the editorial board of the IEEE Transactions on Software Engineering, and the Wiley International Series in Information Systems and the Journal of Empirical Software Engineering. He is a founding member of the International Software Engineering Research Network (ISERN). He was elected Fellow of the Australian Computer Society for his contribution to software engineering research.