TSLAM: A Trust-enabled Self-Learning Agent Model for Service Matching in the Cloud Market

WENJUAN LI, Hangzhou Normal University; Shanghai Jiao Tong University; University of Melbourne
JIAN CAO and SHIYOU QIAN, Shanghai Jiao Tong University
RAJKUMAR BUYYA, University of Melbourne

With the rapid development of cloud computing, various types of cloud services are available in the marketplace. However, it remains a significant challenge for cloud users to find suitable services for two major reasons: (1) Providers are unable to offer services in complete accordance with their declared Service Level Agreements, and (2) it is difficult for customers to describe their requirements accurately. To help users select cloud services efficiently, this article presents a Trust enabled Self-Learning Agent Model for service Matching (TSLAM). TSLAM is a multi-agent-based three-layered cloud service market model, in which different categories of agents represent the corresponding cloud entities to perform market behaviors. The unique feature of brokers is that they are not only the service recommenders but also the participants of market competition. We equip brokers with a learning module enabling them to capture implicit service demands and find user preferences. Moreover, a distributed and lightweight trust model is designed to help cloud entities make service decisions. Extensive experiments prove that TSLAM is able to optimize the cloud service matching process and compared to the state-of-the-art studies, TSLAM improves user satisfaction and the transaction success rate by at least 10%.

CCS Concepts: • **Information systems** → **Trust**; • **Theory of computation** → **Multi-agent learning**; • **Computer systems organization** → **Cloud computing**;

Additional Key Words and Phrases: Cloud market model, service preference learning mechanism, multi-agent platform, trust management

ACM Reference format:
Wenjuan Li, Jian Cao, Shiyou Qian, and Rajkumar Buyya. 2019. TSLAM: A Trust-enabled Self-Learning Agent Model for Service Matching in the Cloud Market. *ACM Trans. Auton. Adapt. Syst.* 13, 4, Article 16 (July 2019), 41 pages.
https://doi.org/10.1145/3317604

This work is supported by the National Natural Science Foundation of China under grant 61702151 and 61702320, National Key Research and Development Plan under grant 2018YFB1003800, Zhejiang Provincial Natural Science Foundation under grant LY17E070004 and the Research Project for Department of Education of Zhejiang Province under grant Y201635438.
Author's addresses: W. Li, Qianjiang College, Hangzhou Normal University, Hangzhou, China; email: liellie@163.com; J. Cao (corresponding author) and S. Qian, Computer Science and Technology, Shanghai Jiao Tong University, Shanghai, China; emails: {cao-jian, qshiyou}@sjtu.edu.cn; R. Buyya, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia; email: rbuyya@unimelb.edu.au.

1556-4665/2019/07-ART16 $15.00
https://doi.org/10.1145/3317604

1 INTRODUCTION

Cloud computing is an Internet-based platform for sharing all kinds of resources at different levels [1, 2]. The most significant features of cloud computing include full virtualization, elastic resource provision, and low-cost on-demand access to services on a subscription-basis model from anywhere and anytime. Since cloud computing offers services based on flexible pricing models, it has gained wide attention from both academia and industry.

IT giants such as Google, Amazon, IBM, and Microsoft have invested a lot of manpower and material resources into the cloud and developed relevant technologies and solutions. Many small- and medium-sized enterprises have also been involved in this field, trying to gain a foothold in the fiercely competitive market. Due to a large number of vendors entering the cloud market, a wide range of cloud services are published in succession. As of 2015, the number of Chinese cloud service enterprises exceeded 1,000, while still in the rapid development stage. It is estimated that by 2020, the total monetary size of global cloud computing services will reach 100 billion [3].

It seems that cloud users can easily find their preferred services from a number of options. However, they become confused by the difficulties of finding suitable and high-performance services. Taking cloud storage services as an example, a large number of providers offer services with similar functionalities but of different quality and price. Thus, for most ordinary users, it is difficult to select services that best suit their preferences.

To help users select a suitable service, a couple of approaches have been proposed [4–6], however these are all based on an assumption, i.e., users' requirements can be fully specified and captured. Unfortunately, this may not always be true in practice. First, users have implicit requirements that may be hard to express, or it may be that the user is not aware of particular requirements. Sometimes, due to security or privacy concerns, users may not fully specify their requirements. Therefore, current research on service matching cannot adapt to situations where service requests are not well described. However, in cloud service markets, there is fierce competition between providers. As for providers, improving their QoS and user satisfaction is critical. To achieve this goal, the first thing they have to do is understand user preferences and then provide services according to the preference. However, as mentioned, when users have implicit requirements, it is not easy to obtain the preference simply by the accumulation of transaction data.

Another challenge is that the actual performance of cloud services is different to that claimed by providers. Furthermore, performance may deteriorate even further if the service succumbs to a malicious attack. Obviously, service matching based on wrong performance information leads to wrong results. Therefore, many researchers have proposed trust-based service matching models in which service matching decisions are made by recording and analyzing the trust-included past performance of providers. References [7] and [8] proposed the use of trust to support QoS-aware service scheduling. X. Li proposed a trust-based broker model to provide professional service recommendations through a third-party intermediary [9]. Our previous work [10] proposed a recommendation transaction-based cloud trust model and trust-enabled cloud workflow scheduling model to achieve a relatively high transaction success rate. However, a common problem with these models is that they all adopt a centralized architecture. Service matching and trust management rely on a centralized broker, which can easily lead to a single point of failure and a situation where the problem cannot be solved because the broker is under attack or is controlled by some untrusted node.

The cloud service market is a self-organizing business system in which entities such as users and service providers make deals in a collaborative and autonomic way [11, 12]. Currently, agent technology is a popular way to construct cloud systems. Intelligent agents enable cloud entities to have self-adaptive, decision making, and learning abilities and multi-agent systems allow the cloud market to operate and evolve in its original way.

To meet these requirements, we propose a Trust-enabled Self-Learning Agent Model (TSLAM) for service matching in cloud computing environments. We use multi-agent technology to build the architecture of the cloud service market model. Different categories of agents represent the corresponding cloud entities to carry out market behaviors. Of these, the broker is a special type of entity who undertakes the task of service recommendation and also participates in market competition. A distributed and lightweight trust model is designed to help entities in service decisions. Moreover, to improve user satisfaction, brokers are equipped with a learning module that enables them to capture the implicit service demands and find the real service preferences of users.

To make the service matching process efficient, satisfactory and reliable, in the construction of our approach, the following key questions are addressed:

- What is a suitable service matching framework in the cloud market and how do cloud entities interact with each other?
- How to deal with trust issues in the service matching environment, including the definition of trust, how to compute trust, how to update trust and how to make decisions about trust?
- How to better analyze cloud users' service requests and learn their real service preferences, especially when customers are not totally open with the details of their requirements?

The key contributions of this article are as follows:

1. the design of a three-layer and agent-based cloud service matching framework and interaction protocols.
2. a proposed distributed and lightweight trust model with the addition of a trust mechanism in the service selection process to improve reliability.
3. the development of a learning mechanism for broker agents to learn their customers' service preferences when there are implicit service demands.

Furthermore, we carry out an extensive evaluation of the proposed algorithms on the multi-agent platform and compare them to state-of-the-art studies.

The rest of the article is organized as follows. Section 2 describes the related work in detail. TSLAM is proposed in Section 3. Section 4 introduces a distributed and lightweight trust management model. Section 5 provides details of the operational mechanisms in TSLAM. The experiment and analysis are presented in Section 6. Finally, conclusions are drawn and future work is suggested in Section 7.

2 RELATED WORK

2.1 Broker-based Service Selection Model for Market-oriented Cloud Computing

Cloud computing is a network-based business platform. To ensure cloud systems operate more efficiently and commercially, some researchers introduced the concept of market-oriented cloud computing [13, 14]. With the rapid increase in the number of available cloud services, service selection has become a core issue in a market-oriented cloud system. An efficient service scheduling mechanism can reduce the waiting time of users and improve their satisfaction with services, However, it can also maximum the benefits of service providers. In a study of service matching models, the broker-based model is the most popular. For example, R. Buyya et al. put forward a cloud computing market framework, including users, service brokers and resource providers, in which resource brokers assist in the collaboration between cloud users and cloud service providers [15]. B. Song et al. proposed a similar concept, called the cloud services market intermediary, which is responsible for matching users and service providers through the auction mechanism [16]. These mechanisms enable clouds to operate in a market-oriented manner. However, there are still many challenges in building an efficient cloud service market, especially in relation to satisfying the QoS demands of users.

S. Arifulina et al. proposed a situation-specific domain-specific language (DSL) for the service broker to make an efficient service composition and to perform market-specific service matching [17]. It is very practical research, but it only focuses on the matching language, the composition process, and is somewhat context limited.

E. Badidi proposed a broker-based service framework for SaaS provisioning [18]. The broker in the proposed framework can help consumers find suitable providers that can fulfill their functional and QoS requirements and uses a multi-attribute negotiation model to negotiate with the selected providers on behalf of the customers and monitors the behaviors of the providers. The negotiation mechanisms in this article are detailed and complete. However, it neglects the realization and application of the model.

T. Deng carried out a study on user behaviors (job arrival rate and job service time) to help cloud broker make scheduling decisions [19]. However, no methods are provided to improve the broker's behaviors as a result of their findings.

D. Rane and A. Srivastava proposed a service broker-based trading framework for cloud computing systems in which the broker is able to calculate users' aggregated requirements and use the newly defined service scheduling algorithm to find an optimized match between the aggregated requirements with the offerings [20]. The article compares the cost and execution time of the model. However, it doesn't compare it with other similar models and the number of providers is small.

M. Aazam and E. Huh proposed a resource estimate and pricing model for service broker in inter-cloud environments on the basis of the historical record of each customer [21]. Mainly from the perspective of service providers, it evaluates the performance of the model. The main problem with this article is that it only predicts users' resource requirements from a price perspective, without fully considering other factors.

S. Aldawood proposed a brokerage model specifically for multi-cloud resource spot markets, using a tuple space architecture to facilitate coordination and matched the lowest price resource for customers [22].

One of the common problems of these strategies is that they neglect the security and trust risks that possibly exist in business cloud markets. Therefore, some researchers have tried to add trust or security factors into the service matching mechanisms of brokers.

S. Wagle et al. used a similarity-matching algorithm to design an SLA-assured brokering framework matching the requirements of customers with the SLA offered by CSPs [23]. Here the broker is more like an inter-cloud gateway whose responsibility includes service packaging, matching, and monitoring. However, the authors do not give enough details about the implementation of the framework.

O. Wenge et al. introduced a security governance-driven cloud brokerage model that uses the security labeling of cloud products to ensure the security trading in ad hoc cloud collaboration [24]. However, they only considered the security needs of the users and there is no technical implementation or simulation of the model.

P. Pawar proposed a trust-enhanced OPTIMIS Cloud Broker (CBR) as a mediation layer to help cloud users find a trustworthy cloud provider and discuss in detail the different role and function of the broker [25]. The problem with the work is that it only fulfils the trust requirements of users and ignores the other factors that can also affect a users' selection.

W. Abderrahim and Z. Choukair proposed a cloud brokering architecture for dependability as a means for trust assurance in service matching [26]. Similarly to the former paper, it mainly takes into consideration the reliability factor in the service matching.

M. Gupta and B. Annappa used a lightweight method to calculate trust and help users to find a credible trading partner [27].

X. Li et al. proposed a trust-aware service brokering scheme named T-broker for the efficient matching of cloud services to users [9]. T-broker combines direct experiences and feedback to establish a hybrid and adaptive trust model to compute the overall trust degree of service resources. The authors conducted experiments to demonstrate the performance of the model.

T-broker, which also adopts the trust broker model, appears similar to our model; however, it is essentially different from ours.

(1) The architecture of cloud service markets is different. T-broker uses the centralized model in which there is only one broker in the market who is in charge of both service matching and trust management. However, in our model, there are many brokers who are also competitors in the market.
(2) The trust management method is different. T-broker adopts the centralized trust management approach. Cloud users rely entirely on a broker to make trust decisions, because there is no trust module on the user side. However, trust management in our model is distributed; that is, brokers manage their trust relationships while users manage theirs.
(3) The timing of service selection is different. Cloud users in T-broker have to use the services recommended by the only broker, but in our model, users can choose whether to adopt the services recommended by brokers or not.
(4) We add a learning ability to the brokers that helps them to understand user preferences, even in the presence of implicit user requirements.

The common problem with the aforementioned trust-based broker models is that they use a centralized broker, which may lead to a single-point failure. Once the centralized broker itself is attacked or controlled by a malicious entity, it may result in market confusion. Furthermore, cloud markets are self-organizing systems. The interaction between market entities is a dynamic and adaptive process. The centralized control model neglects the autonomy and intelligence of the market participants.

To ensure cloud services have autonomy and the ability of independent coordination, a natural choice is distributed management. In this regard, K. Sim and his collaborators undertook a large volume of work, including the multi-agent system for cloud resource management and a series of service scheduling methods [28–34]. Unfortunately, their models did not consider trust issues.

Table 1 summarizes the features of the aforementioned broker-based service scheduling models.

2.2 Trust Management Model

Trust is a mechanism by which to solve reputation and reliability problems in an open environment. In recent years, researchers have made great achievements in the research of trust.

X. Li et al. proposed a dynamic trust model that can more accurately quantify and predict a user's cognitive behavior [35]. The advantages of this model are as follows: (1) it combines the human cognitive model to optimize the weight of direct and feedback trust, and (2) it considers the impact of the time decay factor in the evaluation of trust. Y. Tan et al. proposed a combination-weighted approach based on relative entropy to evaluate user behavior [36]. The main contributions of this article include the following: (1) it uses the combination weighting method to optimize the weight of subjective and objective trust, and (2) it introduces a fuzzy comprehensive evaluation method to compute combined trust. Some researchers proposed evolutionary algorithms combined with trust mechanisms [37, 38]. But usually the time overhead of evolutionary algorithms affects the practical application of these models. S. Wang proposed the trust assessment method based on the cloud model [39]. The model is primarily suitable for the calculation of subjective trust. X. Xie

Table 1. The Comparison of Broker-based Service Scheduling Models

Method	Organization mode	Application scenario	Main function of broker	Learning ability	Security aware
Market-Specific Service Compositions: Specification and Matching [17]	Centralized	On-The-Fly Computing	Market-specific service matching and service composition	No	No
A Cloud Service Broker for SLA-based SaaS provisioning [18]	Centralized	Software-as-a-Service (SaaS) provision	Negotiation, matching, monitoring	No	No
Analysis of user behavior in cloud broker [19]	Centralized	Cloud computing environment	User behavior analysis	Yes	No
Cloud Brokering Architecture for Dynamic Placement of Virtual Machines [20]	Centralized	Cloud market	Calculate aggregated requirements, service scheduling	Yes	No
Broker as a Service (BaaS) Pricing and Resource Estimation Model [21]	Centralized	Inter-cloud computing environment	Service matching, resource estimate, pricing and billing	Yes	No
A Coordination-Based Brokerage Architecture for Multi-cloud Resource Markets [22]	Centralized	Multi-cloud resource spot markets	Lowest cost service matching	No	No
SLA Assured Brokering (SAB) and CSP Certification in Cloud Computing [23]	Centralized	Cloud computing environment	Service composition and matching language	No	Partly
Towards Establishing Security-Aware Cloud Markets [24]	Centralized	Ad hoc cloud collaborations	Service matching, security labelling and negotiation	No	Yes
Trust Assessment Using Cloud Broker [25]	Centralized	Cloud computing environment	Trust-based service selection	No	Yes
The Promethee Method for Cloud Brokering withTrust and Assurance Criteria [26]	Centralized	Cloud market	Service discover, Service composition, Service delivery	No	Yes
Trusted Partner Selection in Broker Based Cloud Federation [27]	Centralized	Cloud federation	Trust evaluation, service selection	No	Yes
T-Broker: A Trust-Aware Service Brokering Scheme for Multiple Cloud Collaborative Services [9]	Centralized	Multi-cloud environment	Trust-based service matching	No	Yes
Agent-based cloud resource management models [28–34]	Distributed	Agent-based cloud platform	Service scheduling	No	No

designed the double excitation and deception detection-based trust model [40]. The main contributions of this article are as follows: (1) it proposes a trust time decay model, and (2) it introduces a provider and user behavior incentive model. However, this model can only be applied to evaluate the trust of providers. K. Ahmadi proposed a trust-based decision-making model for multi-agent societies [41]. This model combines users' subjective experience and recommendation trust; however, it neglects the credibility of recommenders. We have also made some attempts in the research of cloud trust management, such as the development of trust management architecture for open cloud environments [42], the recommendation transaction-based lightweighting trust management model [43], portfolio trust strategy based on fuzzy clustering [44]. However, in the extremely complex and volatile cloud market environments, our trust models still need to solve many problems, such as how to reduce the trust decision cost and how to provide customized strategies for different trust requirements of different cloud entities.

In response to the question of trustworthiness in service selection, some researchers have also proposed trust-based technologies. For example, X. Li proposed a trust-based and multi-attribute service selection algorithm [45]. The main contribution of this article is that it combines trust with the general service attributes to comprehensively evaluate the quality of service. However, the problems with this work are as follows: (1) the test dataset is small, and (2) there is no detailed description of how to deploy the model. C. Hu et al. proposed a trust and spanning tree-based cloud service organization approach to help cloud users eliminate malicious and spurious services [46]. This model can be applied to the scenario of credible service composition. Y. Wang et al. put forward the community trust-driven service selection model [47]. The principle function of the model is to predict the evaluation of users of the unknown services. X. Meng et al. proposed a two-tier service selection approach for matching trustworthy and behavioral patterns [48]. The main contribution of this article is that it can optimize the evaluation of the direct trust experience and the credible recommendation group thus to obtain a relatively higher trust evaluation. However, the authors do not discuss how to test its performance though experiments. S. Yan et al. proposed a service-oriented, user-centric trust model [49]. However, the trust evaluation process seems a bit complex, which is not suitable for real-time cloud trading environments. C. Hang et al. propose two distributed trust-aware service selection approaches for service-oriented computing (SOC) environments [50]. The author did not introduce the implementation of the model. Some researchers proposed service selection algorithms based on trust and QoS preferences, such as in References [51–53]. We also proposed trust-based cloud workflow scheduling and service discovery models [10, 54–56].

The existing service selection strategies based on the trust mechanism have some common problems: (1) Most of them adopt a centralized management method that is not completely reliable and may cause a single point of failure, and (2) the measurement of trust is too complex to be well integrated with the service selection process and also incurs a high system overhead.

3 TSLAM: A TRUST-ENABLED SELF-LEARNING AGENT MODEL FOR SERVICE MATCHING

Despite the differences in the realization technology of the cloud for different service providers, the techniques have become more mature. However, as a kind of commercial solution, the underlying business architecture of cloud computing is still not perfect, especially due to a lack of built-in business management strategies.

Therefore, not surprisingly, the concept of the market-oriented cloud computing model has been put forward [15]. Under this concept, market mechanisms are developed and combined with service provisioning, service billing and virtual machine scheduling approaches. The cloud is becoming a thriving market.

To model the cloud market and to provide the system structure to support the cloud market, our TSLAM is based on a multi-agent framework in which intelligent agents represent cloud market entities to perform market behaviors.

As with any real market, some service providers may exaggerate the effects of service quality. Moreover, fraudulent behavior cannot be avoided in the development of the cloud market. Trust is a simple and safe alternative to a security strategy in the distributed network environment. Therefore, trust is also introduced into the TSLAM.

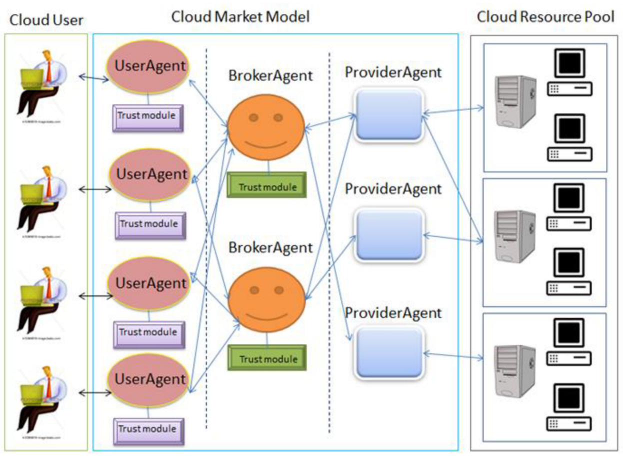

Fig. 1. Framework of TSLAM.

3.1 System Framework

The framework consists of three main components, i.e., user agents, service provider agents, and broker agents (see Figure 1). User agents operate on behalf of users to submit service requests according to their service requirements, select services, use services, and evaluate service quality. On behalf of service providers, provider agents organize resources and deliver services to users. Broker agents recommend cost-effective and high-credibility services to users.

There are many users and service providers in the cloud market, which leads to low efficiency in service matching. Therefore, we add brokers to the market who act as intermediaries and coordinators between users and service providers. Similarly to a broker in the real market, they can make profits by providing matching services between users and service providers.

Two main reasons require brokers to have a learning ability: (1) Users have both explicit and implicit requirements. Because of the multi-attributes and complexity of services that include many different types of requirements (function, non-function, QoS, etc.), users may find it difficult to accurately describe their complete requirements. Sometimes, for security or privacy reasons, users may not entirely describe all their requirements; (2) the management of service providers incurs costs, which means brokers should optimize the list of selected providers according to their customers' preferences and to increase the market success.

Obviously, the cost of service selection is an important user concern, so users tend to choose only a small number of intermediaries (brokers) to cooperate with those whom they believe are highly credible and can provide high quality recommendations. This means only those who provide referral recommendation services that match user preferences are able to survive in the long run. Therefore, to provide accurate recommendations, broker agents should be equipped with a certain learning capability to understand user preference and to optimize their behaviors.

In a commercial market like the cloud, there are various vague and uncertain factors. For example, providers sometimes cannot provide services in full accordance with the SLAs. Some malicious nodes profit from camouflage services. All of these require cloud entities to have sufficient resolutions. Therefore, we design special trust mechanisms for the participants involved in market transactions.

For cloud users, trust behaviors include the following: (1) recording, evaluating, and updating the trust of the corresponding brokers through the trading experience and (2) treating trust as an important factor when choosing trading partners and only cooperating with credible brokers. For brokers, trust behaviors include (1) recording, evaluating, and updating the trust of users and providers through transactions. The trust degree of users is used as a weighting to comprehensively evaluate their satisfaction with providers and their trust in providers, which is one of the most important factors for judging a recommendation, and (2) only selecting highly credible providers when considering a recommendation services.

3.2 Segmentations of Brokers

Similarly to the real market, because of the information asymmetry problem, in the initial phase, broker agents select provider agents to be managed randomly. However, with an increase in transaction numbers, especially when they collect feedback from a number of users, broker agents gradually learn about the main preferences of their customers to choose some new service providers to replace some old ones. This process happens in a random way, i.e., a broker agent who provides miscellaneous services in the initial stage will eventually provide services with some common features. For example, some broker agents provide high-quality and high-cost services, while some broker agents provide cheap services of lower quality. Although they provide services with different features, they can all serve their customers well. This market segmentation phenomenon can be observed in the real market. We realize the market segmentation of brokers by adding a self-learning capability into their models.

3.3 Agent Interaction Protocols

The main processes for service matching include (1) provider agent and broker agent relationship maintenance, (2) user agent and broker agent relationship maintenance, and (3) service matching.

(1) Provider Agent and Broker Agent Relationship Maintenance
Generally, a provider agent tries to register its service information with broker agents so that it can have a chance to provide services. However, a broker agent updates its provider agent list to optimize its business.
(2) User Agent and Broker Agent Relationship Maintenance
Initially, a user agent has to submit the request to broker agents in a random way. After several rounds of interactions, the user agent knows who can provide better services so it submits the requests to these agents in the future.
(3) Service Matching Process
Initially, a user agent sends a service request for a certain cloud service to its familiar broker agents. The broker agents then recommend providers from their provider agent lists. Then the user agent sends an "accept" message to the most suitable broker and a "reject" to the others after selection. The broker agent who receives the "accept" message forwards the service request to the corresponding provider. If the provider agrees to provide the service, then the broker will reply with a confirmation message to the user agent and thus a transaction channel is constructed between the user agent and the provider. After service invocation, the user agent sends evaluation feedback to the broker. The main service matching process are shown in Figure 2.

Four types of messages are used in TSLAM: CFP, PROPOSE, CONFIRM, and INFORM. CFP messages are used for service/cooperation requests, PROPOSE messages are used for recommendation, CONFIRM messages are used for confirmations, and INFORM messages are used for sending the result, either the service or the evaluation. Table 2 shows the function of each kind of message.

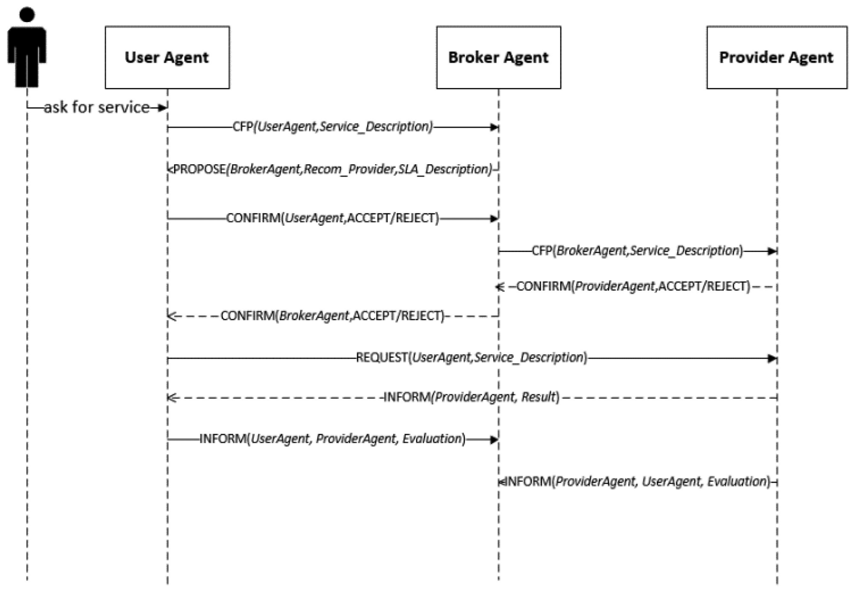
User Agent
Broker Agent
Provider Agent
ask for service
CFP(UserAgent,Service_Description)
PROPOSE(BrokerAgent,Recom_Provider,SLA_Description)
CONFIRM(UserAgent,ACCEPT/REJECT)
CFP(BrokerAgent,Service_Description)
CONFIRM(ProviderAgent,ACCEPT/REJECT)
CONFIRM(BrokerAgent,ACCEPT/REJECT)
REQUEST(UserAgent,Service_Description)
INFORM(ProviderAgent, Result)
INFORM(UserAgent, ProviderAgent, Evaluation)
INFORM(ProviderAgent, UserAgent, Evaluation)

Fig. 2. Service matching process.

Table 2. Four Types of Messages

Message	Representation Forms	Sender	Receiver	Function
CFP	CFP (*UserAgent, Service_Description*)	*UserAgent*	*BrokerAgent*	*UserAgent* asks *BrokerAgent* for service recommedation
	CFP (*BrokerAgent, Service_Description*)	*BrokerAgent*	*ProviderAgent*	*BrokerAgent* asks *ProviderAgent* to deal with a new service request
PROPOSE	PROPOSE (*BrokerAgent, Recom_Provider, SLA_Description*)	*BrokerAgent*	*UserAgent*	*BrokerAgent* recommends service to *UserAgent*
CONFIRM	CONFIRM (*UserAgent,* ACCEPT/REJECT)	*UserAgent*	*BrokerAgent*	*UserAgent* agrees with the recommendation or not
	CONFIRM (*ProviderAgent,* ACCEPT/REJECT)	*ProviderAgent*	*BrokerAgent*	*ProviderAgent* agrees to provide service or not
	CONFIRM (*BrokerAgent,* ACCEPT/REJECT)	*BrokerAgent*	*UserAgent*	*BrokerAgent* confirms the service to *UserAgent*
INFORM	INFORM (*ProviderAgent, Result*)	*ProviderAgent*	*UserAgent*	*ProviderAgent* tells *UserAgent* the result
	INFORM (*UserAgent, ProviderAgent, Satisfaction*)	*UserAgent*	*BrokerAgent*	*UserAgent* sends the satisfaction of provider to *BrokerAgent*

4 TRUST MANAGEMENT STRATEGY

4.1 The Definition of Trust

The concept of trust comes from sociology. In 1996, M. Blaze introduced trust into distributed systems to solve the security problems of the Internet services. However, due to differences in the context of different services, it is very difficult to give trust a universal definition. To better describe the subsequent problems, this article gives trust a relatively comprehensive definition under the deep analysis of former research.

Definition 4.1 (Trust). This refers to the trust of the trustor in the trustee in the recognition of the trustee's identity and the trust of the trustor that the trustee will complete some special task as expected over a specified period of time and in a particular context. Trust is the trustor's subjective evaluation of the trustee depending on their own experience and knowledge of the trustee. Indicators of trust include authenticity, honesty, reliability, and stability.

Definition 4.2 (Trust Degree). This refers to the trust value that represents the trust degree of the trustor in the trustee. In real applications, the trust degree may be jumpy or continuous.

Definition 4.3 (Direct Trust (DT)). DT is calculated on the basis of the direct transactions between the trustor and the trustee. In this article, DT (A, B, tc) is used to represent the direct trust degree of cloud entity A to B in a special transaction context (tc).

Definition 4.4 (Recommendation Trust (RT)). RT is calculated based on the recommendations of the other entities with which the trustor is familiar. In this article, RT (A, B, tc) is used to represent the recommendation trust of cloud entity A to B in a special context (tc).

Definition 4.5 (Function Trust (FT)). FT refers to the trust of the trustor in the trustee as to whether or to what extent the functions provided by the trustee can meet the requirements of the trustee.

Definition 4.6 (Experience Trust (ET)). ET is the trust of the trustor in the trustee as a result of the direct interactions between them or feedback from the familiar entities of the trustor who have direct interactions with the trustor. If ET is gained purely by the former, say, the direct transaction experience of the trustor, then ET is equal to DT, which means DT is a special part of ET.

Definition 4.7 (Integration Trust (IT)). IT is the final assessment of the trustor in relation to the trustee that is normally obtained by the integration of DT and RT, or by the combination of FT and ET. In this article, IT (A, B, tc) is used to represent the integration trust of cloud entity A to B in a special context (tc).

There are two types of computation models of behavior-based service trust: (1) DT plus RT computation model, and (2) FT plus ET computation model. Equation (1) shows the general method of the first model, and Equation (2) shows the second,

$$IT(A,B,tc) = \alpha * DT(A,B,tc) + \beta * RT(A,B,tc), \quad \alpha + \beta = 1, \tag{1}$$

$$IT(A,B,tc) = \alpha * FT(A,B,tc) + \beta * ET(A,B,tc), \quad \alpha + \beta = 1. \tag{2}$$

Since the distributed trust management method is adopted, this article uses the above two models flexibly and selectively according to the different trust requirements of the different entities, with the target of minimizing trust overhead.

4.2 Trust entities in TSLAM

Trust management in TSLAM is distributed and implemented on different nodes. There are primarily three kinds of cloud entities: users, brokers, and providers. However, since providers commonly act as resource managers and service providers, they are not treated as trust implementers, which means the QoS performance and the trust in the providers are evaluated, but they are not subjective entities of trust. The active and subjective trust entities in TSLAM are cloud users and brokers.

In addition, since services are recommended by brokers and many attributes of a service, such as geographic location and the provider, are totally transparent to users, users do not have to maintain a providers' trust list. The only thing users need to do before choosing a service is to calculate

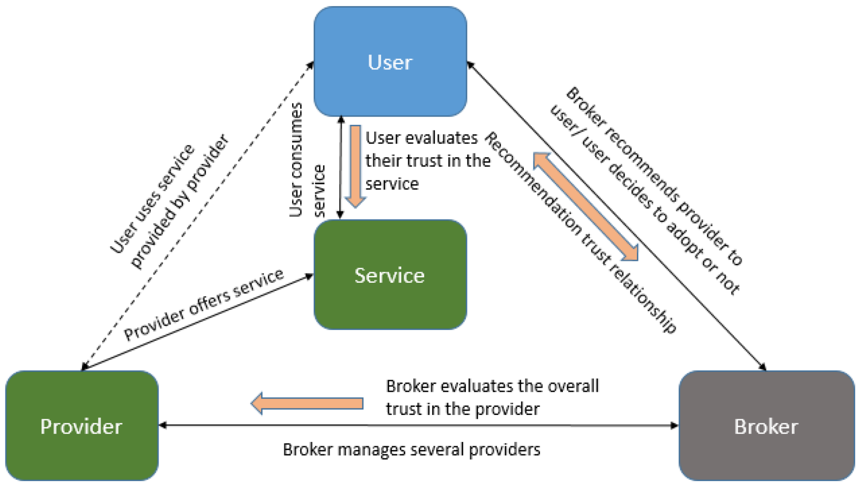
User
User evaluates their trust in the service
User consumes service
User uses service provided by provider
Service
Provider offers service
Recommendation trust relationship
Broker recommends provider to user/ user decides to adopt or not
Provider
Broker
Broker evaluates the overall trust in the provider
Broker manages several providers

Fig. 3. Trust parties and their relationship in TSLAM.

Table 3. The Evaluation Indexes of User-to-broker Trust

	Recommendation accuracy (ra)	Recommendation timeliness (rt)	Recommendation reliability (rr)
Initial score	50	100	50
Positive experience	if $ra < 100$, $ra = ra + 1$		If service delivered && satisfaction>0.5, $rr = rr + 1$
Negative experience	if $ra > 0$, $ra = ra - 1$	if $rt > 0$, $rt = rt - 1$	If service delivered && satisfaction<0.5, $rr = rr - 1$
			If service not delivered, $rr = 0$

their trust in the services according to their functional attributes and the recommendation trust of the recommenders (brokers). Therefore, the temporary trust of users in providers should be more accurately called the trust of user in service. In all, there are four types of trust in TSLAM: (1) trust of users in brokers, (2) trust of users in services, (3) trust of brokers in users, and (4) trust of brokers in providers. Figure 3 shows the relations between the trust parties.

In TSLAM, trust between users and brokers is evaluated using direct transaction experience. When users choose services, they evaluate the trust of a service by utilizing the combination of the functionality of the service and the recommendation trust of the brokers, whereas the trust of the brokers in providers is based on both direct experience and user feedback (recommendation).

4.3 The computation of the trust degree in TSLAM

4.3.1 Trust of the User in the Broker. Table 3 shows the evaluation indexes included in the user-to-broker trust and the calculation method.

Recommendation accuracy measures the similarity of the recommended service to the actual requirement. A positive recommendation experience means the service recommended by a certain broker is accepted. In this case, the recommendation accuracy of the broker is increased by one and that of the other brokers who also provide recommendations is decreased. Recommendation timeliness means whether or not the broker can provide the service recommendation in time. If the recommendation is offered within the deadline, then the value of the recommendation timeliness

Table 4. The Evaluation Indexes of User-to-service Trust

	Functionality (f)	Recommendation trust
	Similarity(functionalities, provision)	$T(user_i, broker_k, tc)$

Table 5. The Evaluation Indexes of Broker-to-user Trust

	Contribution (c)
Initial score	0
	See Equation (5)

remains the same. Otherwise, it is reduced by one. Recommendation reliability means whether or not the actual experience of the service is consistent with the recommended SLA. If so, then it is increased by one, and if not, it is reduced by one.

The overall trust rating of a certain broker is the average value of the three indexes and is normalized to the range of [0,1] using Equation (3),

$$T(user_i, broker_j, tc) = \alpha * \frac{ra_j}{100} + \beta * \frac{rt_j}{100} + \gamma * \frac{rr_j}{100}, \quad \alpha + \beta + \gamma = 1, \tag{3}$$

where α, β, and γ are the weights of the above three factors in calculating the integrated trust.

4.3.2 *Trust of the User in the Service.*

Two factors are taken into consideration in the calculation of user-to-service trust, as shown in Table 4. Functionality refers to the similarity of the functionalities of the service to the actual requirements of the user. Recommendation trust is the trust of the user in the broker who recommends the service. The overall user-to-service trust is the combination of the two. Equation (4) shows the method,

$$T(user_i, service_j, tc) = T(user_i, broker_k, tc) * \frac{Req_i * Pro_j}{||Req_i||Pro_j||}, \tag{4}$$

where $T(user_i, broker_k, tc)$ is the recommendation trust of $user_i$ in $broker_k$ who recommends $service_j$. Req_i is the service request vector and Pro_j is the provision vector. We use cosine similarity to compare the similarity of the request and the provision.

4.3.3 *Trust of the Broker in the User.*

The trust of the broker in the user in this article mainly refers to the contribution of a certain user to the trading volume of the broker. The broker's trust in users is an evaluation of the credibility and importance of user feedback. The initial score and the computation method of broker-to-user trust are shown in Table 5.

$$T(broker_i, user_j, tc) = \frac{User\_Adopt\_Number_j}{TotalRecommedation\_Number_i}, \tag{5}$$

where $User\_Adopt\_Number_j$ indicates the number of times the user has adopted the broker's recommendation, and $TotalRecommedation\_Number_i$ refers to the total number of successful recommendations of the broker.

4.3.4 *Trust of the Broker in the Provider.*

The trust of the broker in the provider includes the direct experience with the provider (reliability) and the feedback of the users (satisfaction). Table 6 shows the indexes of broker-to-provider trust. Reliability refers to whether or not the provider offers the agreed service. Satisfaction is calculated based on the satisfaction of the user's feedback combined with the user's trust.

Table 6. The Evaluation Indexes of Broker-to-provider Trust

	Reliability (r)	Satisfaction (s)
Initial score	100	50
Positive experience		if $s < 100, s = s + 1$
Negative experience	if $r > 0, r = r - 1$	if $s > 0, s = s - 1$

The overall trust rating of a certain provider is the average value of the two indexes and is normalized to the range of [0,1] using Equation (6),

$$T(broker_i, provider_j, tc) = \alpha * \frac{\sum_{k=1}^{n} T(broker_i, user_k, tc) * s_{kj}}{100} + \beta * \frac{r_j}{100}, \quad \alpha + \beta = 1, \quad (6)$$

where α and β are the weights of the two indexes in calculating the integrated trust. $T(broker_i, user_k, tc)$ is the trust of $broker_i$ in $user_k$ and s_{kj} is the satisfaction of $user_k$with $Service_j$.

4.4 Trust Decision

In a cloud market, when A wants to trade with B, A first checks the trust of B $T(A, B, tc)$. If $T(A, B, tc)$ is greater than a certain threshold, then A starts to transact with B; otherwise, A will not continue. However, different trust entities use trust in different ways. For users, trust is the most basic factor on which they decide whether or not to adopt the broker's recommendations, while user-to-service trust is the basis for service selection. As for brokers, the broker-to-provider trust determines whether or not a certain provider will be recommended to some users, while broker-to-user trust determines to which extend the user influences the judgement of the broker.

5 AGENT BEHAVIOR IMPLEMENTATION

5.1 Learning Agents

Learning has the advantage of allowing agents to initially operate in unfamiliar environments and to become more competent as time goes by. The most important distinction is between the "learning element," which is responsible for making progress, and the "performance element," which is responsible for selecting external actions.

The learning element uses feedback from the "critic" on how the agent performs and determines how the performance element should be modified so that the agent will do better in the future. The performance element is what we have previously considered the entire agent: It takes in percepts and decides on actions. The learning agent model is shown in Figure 4.

5.2 User Agent

5.2.1 User Behaviors. User agents simulate cloud users' behaviors with the aim of meeting their needs and achieving their goals. Therefore, behaviors of user agents comprise four parts: (1) generation of requests, (2) sending requests, (3) choice of services, and (4) use and evaluation of services.

Figure 5 shows the main algorithm flowchart of user agents. First, user agents periodically and randomly generate a new service request according to service preferences. Second, they send requests to several broker agents they trust. Third, they choose service partners according to the similarities between the recommendation services and their requirements. Last, they consume the selected services, send feedback on their satisfaction to the brokers and then update the recommendation trust of the brokers.

It is natural that users tend to cooperate with brokers with a larger volume of transaction records and a higher degree of trust. Therefore, when user agents choose services, the decision they make is

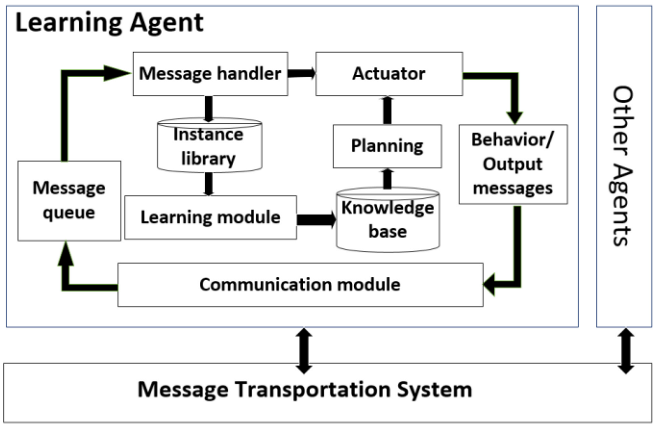

Fig. 4. Learning agent model.

based on the combination of the similarity of the recommended service with the recommendation trust of the recommender/broker.

When a user agent receives a recommendation message PROPOSE (*BrokerAgent, Recom_Provider, SLA_Description*) from a broker agent, it compares the real service demand with the provision and then chooses the most suitable one. At the same time, it updates the trust value of the corresponding partners. Algorithm 1 shows the above process.

In TSLAM, cloud users use Equation (4) introduced in Section 4.3.2 to compute the expected satisfaction (similarity) of the different services recommended by the different brokers. If no service is chosen or no broker replies, then the user discards this request and sends a new request after a certain time. In addition, in every fixed period, as long as the number of brokers is not saturated, users try to find new brokers.

5.2.2 Reflection of the Implicit Requirements. In TSLAM, Service_Description in CFP (UserAgent,Service_Description) is the only open part of the user request that is slightly different from Request_Detail, which is the complete description of the request. Because of this, brokers do not know the exact service preferences of their customers. Moreover, since users use their real service preferences to choose and evaluate the services, brokers may gradually learn the preferences using the learning mechanisms.

5.3 Broker Agent

Broker agents represent the third-party agencies in the cloud market, specializing in service recommendation. The main behaviors of a broker agent are as follows: (1) select appropriate providers to manage from the provider list, (2) recommend appropriate services to users, (3) maintain the credibility of their users and resource providers, and (4) try to learn their customers' service preference from the historical transaction data for the subsequent selection of service resources.

Of these behaviors, the core ones are service matching and service preference learning. Figure 6 shows the main processes of the service matching algorithm. First, brokers listen to the service requests from users, and, once received, they immediately start the service recommendation process. Brokers select the most suitable services according to the specific service-matching rule from the services they manage. Once the recommendation is adopted, the broker contacts the corresponding provider for confirmation and after this, he sends a confirmation to the user. Finally, based on the performance of the provider and the user during the transaction, the broker updates their trust values.

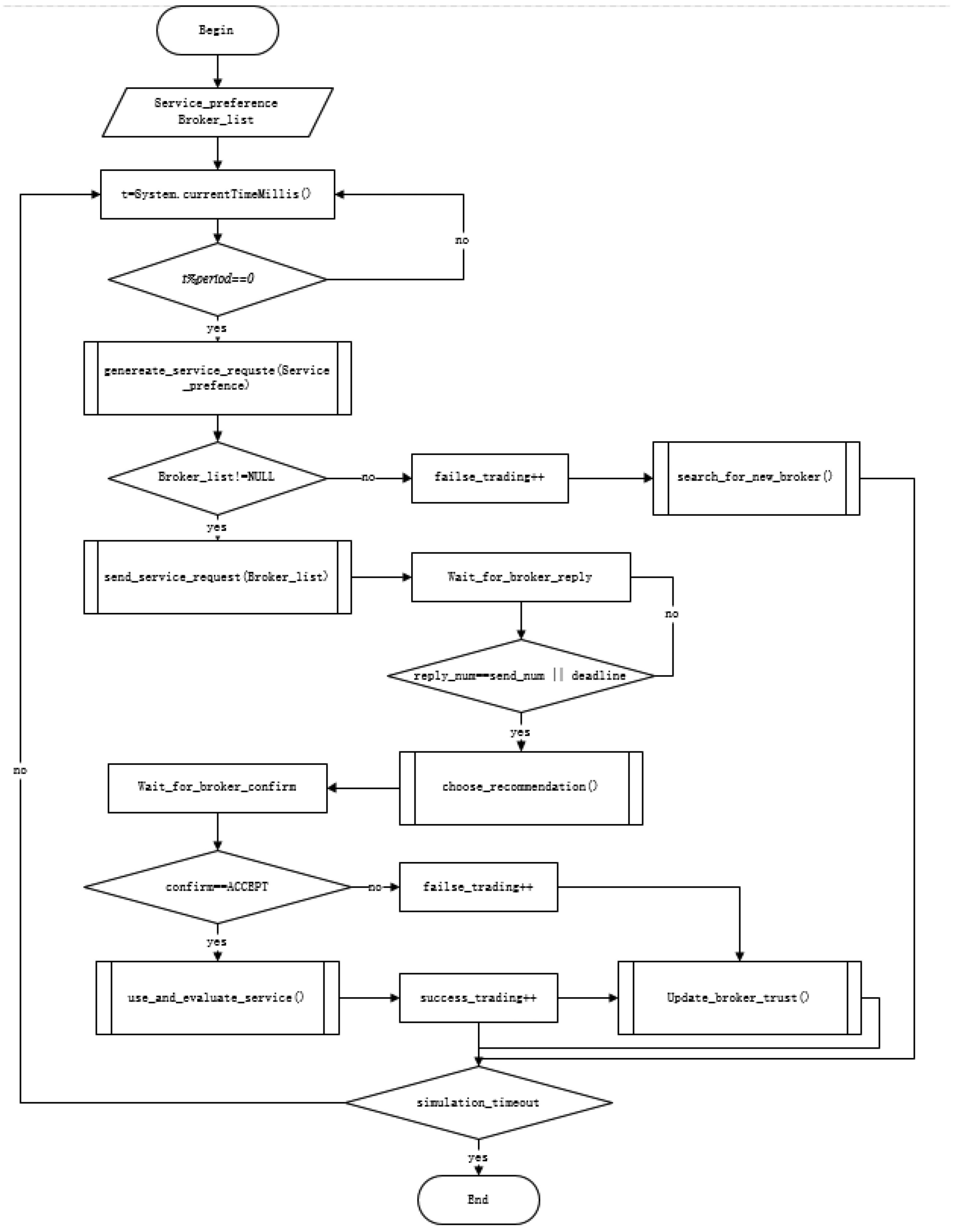

Fig. 5. Algorithm flowchart of user agents.

The main difference between this article and the general service matching process is that, in this article, the service requests of users are semi-open, which means brokers can only calculate the similarity of the open part and thus recommend a service. However, the attributes of the recommended services are completed. Therefore, brokers can gradually learn the real intent and preferences of users according to whether or not they adopt the services and the final evaluation

ALGORITHM 1: When user agents receive message PROPOSE (*BrokerAgent*, *Recom_Provider*, *SLA_Description*)

```
Calculate_similarity ($Request\_Detail, SLA\_Description$)
if all recommendations received or timeout then
   broker$_i$ with smallest similarity $\rightarrow$ $chosen\_broker$
   send CONFIRM($UserAgent$, ACCEPT) to $chosen\_broker$
   Recommendation_accuracy[$chosen\_broker$] += 1;
   foreach $broker_i \in request\_Broker[]$
      if reply in due time then
         if ($broker_i$ <> $chosen\_broker_i$) then
            send CONFIRM ($UserAgent$, REJECT)
            Recommendation_accuracy[$broker_i$] −= 1;
         end if
      else
         Recommendation_timeliness[$request\_broker_i$] −= 1
         if $trust[request\_brokeri] \leftarrow 0$ then
            remove($request\_brokeri, Blist$)
         end if
      end if
         update_trust ($broker_i$, Recommendation_accuracy[$broker_i$],
          Recommendation_timeliness[$broker_i$])
   end for
end if
Wait_for_confirm ($chosen\_broker$)
if msg == ACCEPT then
   Consume service . . .
   Satisfaction = Evaluate_service();
   if Satisfaction >= 0.5 then
      Recommendation_reliability[$chosen\_broker$] += 1;
   else
      Recommendation_reliability[$chosen\_broker$] −= 1;
   end if
   Send_satisfaction($chosen\_broker$);
else
   Recommendation_reliability[$chosen\_broker$] = 0;
end if
update_trust ($chosen\_broker$, Recommendation_reliability[$chosen\_broker$]);
```

of the services. In addition, we add the trust factor into the process of service recommendation. Brokers update the trust of the providers according to the performance before trading (service registration) and during the trading, the value of which will influence the next recommendation. Furthermore, the user's contribution to the broker's trading volume also directly affects the weight of the satisfaction evaluation offered by the user, because the broker's aim in obtaining the data is to adjust their resource/service structure to improve the satisfaction of their important customers.

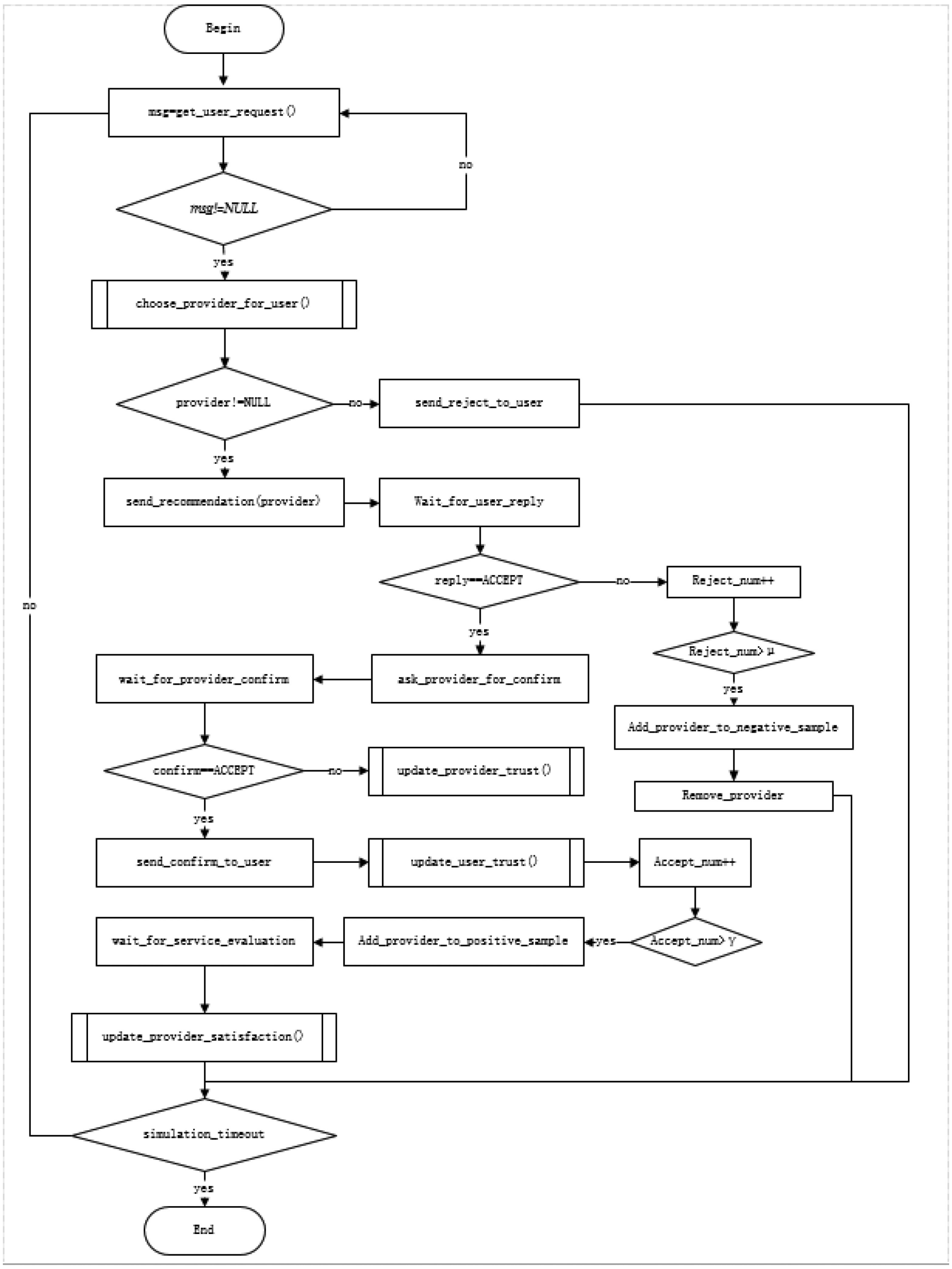

Fig. 6. Service-matching flowchart of broker agents.

A trust-enabled service recommendation mechanism and learning mechanism are introduced in the following sub-section.

5.3.1 Recommendation Mechanism. In TSLAM, $\boldsymbol{R_{recommend}}$ is used to express the probability that a certain provider is recommended. Here, $\boldsymbol{R_{recommend}}$ *is* affected by three factors: matching factor, sales factor, and trust factor (including satisfaction and reliability).

The recommendation probability is the result of the these three factors. Equation (7) shows the computation of $\boldsymbol{R_{recommend}}$,

$$\boldsymbol{R_{recommend}} = \alpha * \boldsymbol{R_{match}} + \beta * R_{sale} + \gamma * R_{trust}. \tag{7}$$

Here, $\boldsymbol{R_{recommend}}$ is the recommendation probability of a certain provider, $\boldsymbol{R_{match}}$ is the matching factor, R_{sale} is the sales factor, R_{trust} is the trust factor, and α, β, and γ are the weights of the above factors, and the sum is 1.

Matching factor $\boldsymbol{R_{match}}$ represents the similarity between the user's request and the resources, which is the similarity of vector $U(p_1, p_2, p_3, \ldots, p_n)$ to $S(p_1, p_2, p_3, \ldots, p_n)$. Here cosine similarity is used to calculate $\boldsymbol{R_{match}}$ using Equation (8),

$$\boldsymbol{R_{match}} = \frac{\sum \boldsymbol{S_{p_{kj}}} \times \boldsymbol{U_{p_{ij}}}}{\sqrt{\sum \boldsymbol{S_{p_{kj}}}^2} \times \sqrt{\sum \boldsymbol{U_{p_{ij}}}^2}}, \tag{8}$$

where $S_{p_{kj}}$ is the jth attribute value of the service S_k and $U_{p_{ij}}$ is the the requirement value of user i for the jth attribute

The sales factor is the rank of providers managed by the broker and the popularity of the cloud service providers. Sales factors can be calculated by Equation (9),

$$R_{sale} = 1 - \frac{rank_i}{N_{provider}}, \tag{9}$$

where $rank_i$ is the sales ranking of provider i in the broker agent and $N_{provider}$—the number of cloud providers managed by the broker.

The trust factor reflects the degree of reliability and satisfaction of a provider. In TSLAM, we use Equation (6) to calculate the value, which is introduced in the trust section.

Although the requests sent by users are not complete, this doesn't affect service matching. Brokers compute the similarity of the open part of a request to the corresponding part of a service (using cosine similarity), which is the first factor of recommendation. The probability that a provider is chosen by users (popularity) is the second factor, which is consistent with the sales factor in a real market, because a hotspot is more likely to be selected, and the user satisfaction combined with the recommendation trust of the user is the final factor. Therefore, TSLAM comprehensively considers the function and non-function attributes of the requirements, because the user's QoS preference will gradually manifest in his satisfaction rating of the service.

When broker agents receive a message CFP (UserAgent,Service_Description), Algorithm 2 is executed.

ALGORITHM 2: When broker agents receive message CFP (*UserAgent*, *Service_Description*)

```
foreach $provider_i \in Plist$
    Calculate $R_{\text{recommend}}[provider_i]$
end for
$Provider_i$ with maximum $R_{\text{recommend}} \rightarrow Recom\_provider$
send PROPOSE($BrokerAgent$, $Recom\_Provider$, $SLA\_Description$)
```

Algorithm 2 shows that when a broker agent receives a query from user agents, he chooses a specific provider according to the $R_{recommend}$ value. However, the user also has the right to accept or reject the recommendation. When a broker agent receives a message CONFIRM (*UserAgent*, ACCEPT/REJECT), Algorithm 3 is executed.

ALGORITHM 3: When broker agents receive message CONFIRM (*UserAgent*, ACCEPT/REJECT)

```
if (msg == ACCEPT) then
    send CFP (BrokerAgent, Service_Description) to Recom_provider
    Waiting for reply message CONFIRM (ProviderAgent, ACCEPT/REJECT)
    if (reply == ACCEPT) then
        Send CONFIRM (BrokerAgent, ACCEPT) to user agent
        Total_recommendation_num++;
        User_adopt_num[user]++;
        update_trust(user_list);
        wait for user consume the service. . . .
        wait for the user's feedback..
        if Satisfaction >= 0.5 then
            satisfaction[Recom_provider] += 1;
        else
            satisfaction[Recom_provider] −= 1;
        end if
    else
        Send CONFIRM (BrokerAgent, REJECT) to user agent
        Reliability[Recom_provider] −= 1;
    end if
else
    reject_num + +
    if reject_num > N then
        Delete Recom_provider from Plist
    end if
end if
update_trust (Recom_provider, Reliability[Recom_provider], satisfaction[Recom_provider]);
if (trust[Recom_provider] <= 0) then
    Delete Recom_provider from Plist
end if
```

Algorithm 3 shows that when a broker agent receives a user agent's acceptance, he or she launches the application to the corresponding service provider. If the provider agrees to offer services, then he or she confirms to the user that the service is available. If the provider refuses to provide the services, then its trust value is reduced. However, if the user rejects the service, then it increases the reject time of this provider. When the reject time of a specific provider exceeds a threshold, the provider is deleted from the local recommendation list. When the transaction is completed, the broker agent updates the provider's satisfaction degree according to the user's feedback on the service.

5.3.2 *Learning Mechanism.* The free market, at the beginning, was in a state of chaos, because entities did not know the specific circumstances of the other partners, including their abilities, preferences, and so on. However, to maximize their interests, in the process of a continuous game,

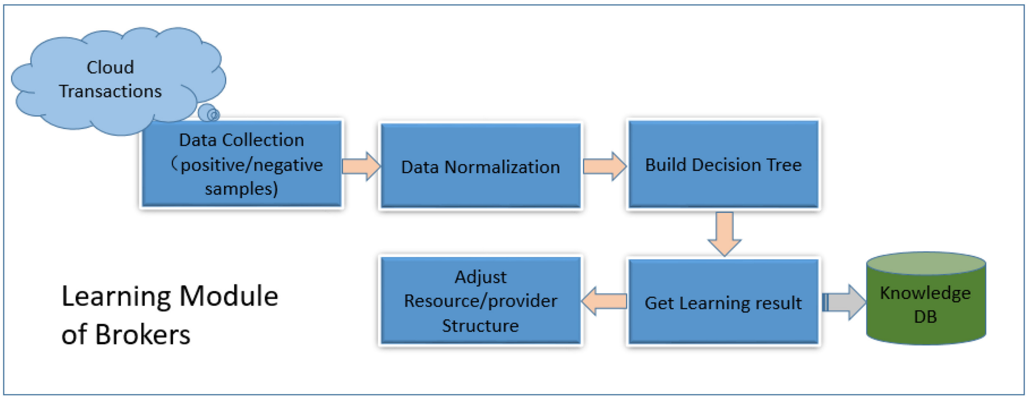
Cloud Transactions
Data Collection (positive/negative samples)
Data Normalization
Build Decision Tree
Learning Module of Brokers
Adjust Resource/provider Structure
Get Learning result
Knowledge DB

Fig. 7. Learning process of brokers.

they eventually reach a relatively stable and balanced situation. This state of equilibrium in spontaneous free markets needs to evolve over a long time. To accelerate the process, a learning ability is added in the operation mechanism of the broker agent. The learning ability of market agents has many advantages: (1) It reinforces the law of the survival of the fittest, that is, brokers and providers who care about the real needs of users are continuously strengthened, while those who do not confirm the appropriate service face the possibility of elimination; (2) it accelerates market differentiation, which means that brokers will gradually learn their customers' preferences and make clear their market positioning. When brokers can learn the main requirements of their clients very well and strengthen themselves in this respect, due to the randomness of the initial stage of service requests and responses, over time, they will inevitably lead to market differentiation along with the entire cloud services market being divided into different types.

Figure 7 shows the learning process of the brokers in TSLAM. Brokers constantly study and learn something new from trading history. They collect transaction data, and according to a certain rule, determine the service samples to enter the positive or negative set. Second, to eliminate the influence of dimensions of the different service attributes, data normalization is performed. When they gather a specific amount of transaction data, brokers start the learning module. In this article, the decision tree algorithm is adopted to obtain the learning result. At intervals, if the managed providers of the broker are not saturated, it tries to find out more from the market. At this point, to input the service attributes of the providers to the learning result of the decision tree, brokers can obtain the judgement of whether or not to continue cooperation. Therefore, the learning module helps brokers to revise their resource strategies to become more competitive in the cloud market.

The idea of this article is different to traditional recommendation systems, because here, the target of broker learning is not to recommend additional services, but to adjust the structure of the services they manage to improve user satisfaction. Moreover, user preferences in this article are not directly applied to each service matching, but to adjust the provider/service structure of a broker. The recommended providers or services that are adopted by the users and the satisfaction that is higher than a certain threshold become positive samples, and those rejected multiple times enter the negative sample set, thereby constructing the training set and generating the decision tree. When brokers want to introduce more providers, they use the decision tree model to make the judgement. Obviously, since new providers are selected according to user preferences, they are closer to the real requirements of the users and since the adjustment is ongoing, this ensures that even if users have implicit service requirements, the overall service attributes of the broker are gradually approaching the real service demand of users.

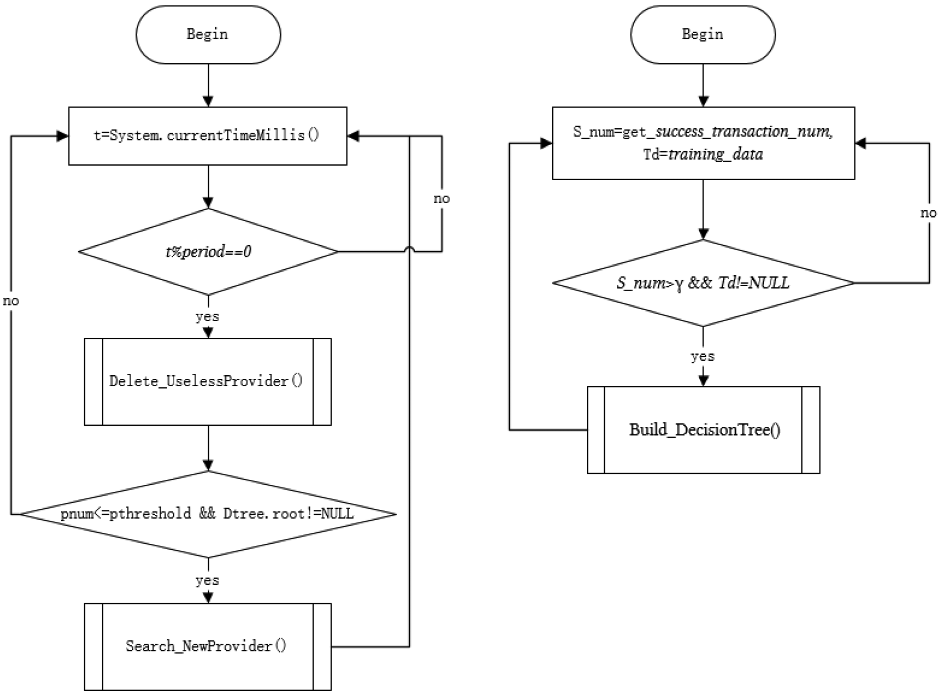

Fig. 8. Main flowchart of the learning algorithm.

(1) The main parts of the learning mechanism

In TSLAM, the learning mechanism consists of three parts: training, elimination and selection. The training uses the transaction samples to build the decision tree. Given the constraint that the number of providers managed by a broker agent is less than a certain threshold, this is in line with the reality that managing resources also incurs a cost. The elimination deletes those unfavorable providers (providers with big *reject_number*) to maintain a more competitive team. The selection periodically adds new and competitive providers. Figure 8 shows the flow of the learning algorithms where the diagram on the left shows how brokers adjust the service structure, and the diagram on the right shows how they build the decision tree.

(2) Decision tree-based learning and selection algorithm

In TSLAM, the decision tree algorithm is used to decide whether a provider is suitable or not. This algorithm comprises two parts: training and selection. In the training, the training set is organized according to the previous trading records of users' feedback, affirmation, rejection, and satisfaction. In our model, the accepted resources are treated as positive reception, while service resources that are rejected several times are counterexamples, providing the basis for the follow-up resource selection for brokers. The selection helps brokers find suitable providers who can guarantee to meet their customers' preferences according to the results of the training process. To better deal with continuous values that seem more common in the transaction context, the C4.5 algorithm is used to build the decision tree. The basic steps of the algorithm are shown as follows.

In the structure of the decision tree, each internal node represents a property and the branch of the node indicates the possible result of the attribute. In TSLAM, node attributes include price,

ALGORITHM 4: Broker agents build the decision tree

```
Input: $training\_data$, $success\_transaction\_num$, $attributeList$
Output: the $root$ of the decision tree
if ((!$training\_data$.isEmpty())&&($success\_transaction\_num > \gamma$)) then
    DecisionTree $dt$ = new DecisionTree($attributeList$, $training\_data$);
    $dt.buildDT()$;
    $root = dt.getRoot()$;
endif
```

network bandwidth, memory size, and so on. The branch nodes are the values of the corresponding attributes, namely the attribute value under the restriction of service acceptance. Algorithm 5 shows the detail of builtDT().

When a decision tree is built, broker agents are able to select the appropriate providers. Algorithm 6 shows how to make decisions based on this.

ALGORITHM 5: The procedure of *buildDT*()

```
Input: $node$, $training\_data$, $attribute\_List$
Output: $the\ decision\ tree$
if (all samples in $training\_data$ are in one category) then
  $node$.setDecision(yes or no);
  return;
else
  $node$.setDecision("node");
endif
Node_List $childtree$;
foreach $attribute_i \in attribute\_List$
    $entropy$ = calNodeEntropy($training\_data$, $attribute_i$);
    if($entropy < minEntropy$) then
        $minEntropy = entropy$;
        $chose\_attribute = attribute_i$;
    endif
endfor
$tmpList = attribute\_List$.remove($chose\_attribute$);
$attvalues = chose\_attribute$.attributevalue;
foreach $val \in attvalues$
    Node $newnode$;
    $newnode$.setValue($val$);
    $childtree$.add($newnode$);
    foreach($data \in training\_data$)
      if($data$.getvalue($chose\_attribute$).equals($val$))then
        $data\_subset$.add($data$);
      endif
      buildDT($newnode$, $data\_subset$, $tmpList$);
    endfor
endfor
$node$.setChildren($childtree$);
```

ALGORITHM 6: Judging whether a provider can be selected

```
Input: $PS\_Data, root$
Output: $judge$(Boolean value: yes or no)

boolean $judge$ = false;
Node $node$ = $root$;
while ($node$.getDecision().equals("node"))
do
    if (!$node$.hasChild()) then
      break;
    endif
    Node_List $childtree$ = $node$.getChildren();
    foreach $child \in childtree$
      PS_Value = $PS\_Data$.getvalue($child$.getAttribute());
        if($child$.getValue().equals($PS\_Value$))then
       $node$ = child;
       break;
        endif
    endfor
done
if($node$.getDecision().equals("yes")) then
      $judge$ = true;
endif
return $judge$;
```

(2) A simple example to show how the learning mechanism works

Following is a simple example to illustrate how the decision tree algorithm can help to choose providers that meet the service preference of the users.

When providers in the provider list are chosen or rejected by the users, they are added into the training list. For example, the training list managed by a broker is shown in Table 7.

Using the above training data, the decision tree is built. Figure 9 shows the result.

Then when the broker wants to choose providers that meet the service preferences of the customers, it uses the decision tree to make the judgement. For example, *Provideri* whose capability is *pi(low, small, no)* will not be chosen for the broker's service pool, because it goes to the decision of "no." Learning mechanism is an iterative process. Thus, after several iterations, broker agents gradually make clear their mainstream types of services.

Although service requests in this article are assumed to be incomplete and include implicit demands, this means that in the service matching step, brokers can only calculate the similarity between the open part of a service request and the provision and based on it, they perform recommendations. However, this does not prevent brokers from gradually learning user preferences through multiple transactions, because brokers grasp the complete information of the services, which enables them to understand the implicit requirements of users whether or not the service is adopted and the user's satisfaction after using the service. Moreover, brokers adjust their provider structure at regular intervals, and the providers selected by user preferences are closer to the actual needs of users. In this way, brokers have a greater chance of successful recommendations in the future.

Table 7. The Training List

PID	price	storage	reliable	Yes/no
1	high	big	yes	yes
2	high	big	no	No
3	high	medium	yes	No
4	high	small	yes	No
5	high	medium	no	No
6	mid	medium	no	No
7	mid	big	yes	Yes
8	mid	medium	yes	Yes
9	mid	big	no	No
10	mid	medium	yes	Yes
11	low	big	no	Yes
12	low	medium	yes	Yes
13	low	big	yes	Yes
14	low	small	no	no
15	low	medium	no	no

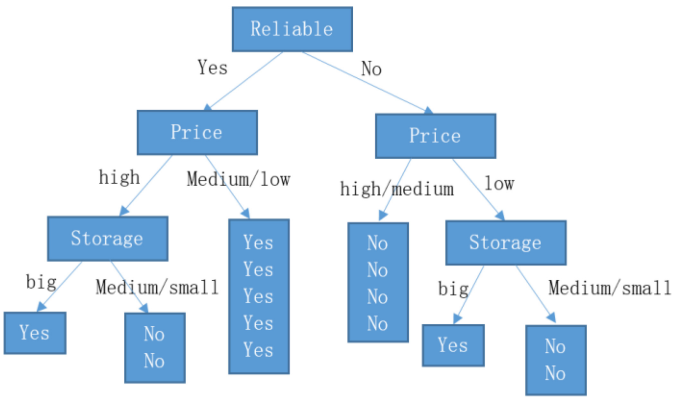

Fig. 9. Decision tree in the example.

5.4 Provider Agents

Provider agents are the managers and organizers of cloud resources, or can simply be seen as the cloud resource pool. The core behaviors of provider agents are as follows: (1) complete the registrations in the cloud platform, (2) determine whether or not to provide a service according to the load or other factors after the broker agents' service requests are received, and (3) provide service to cloud users.

When provider agents receive a message CFP (*BrokerAgent, Service_Description*) from a broker agents, Algorithm 7 is executed.

6 PERFORMANCE EVALUATION

6.1 Experiment Design

Simulations are adopted in the subsequent experiments, because real environments, especially cloud service markets, are extremely complicated and have many fuzzy and uncontrollable factors,

Table 8. Comparison of Multi-agent Simulation Tools

Name	Open source	Domain	Simplicity/ Learnability	Performance	Scalability	User support	Popularity
JADE	Yes	Distributed applications composed of autonomous entities	User-friendly GUI, many familiar features, easy to learn (many examples)	High	High	High (FAQ, mailing list, defect list, API, docs)	High (most popular)
JadeX	Yes	Distributed applications composed of autonomous BDI entities	User-friendly GUI, many features, average to learn	High	High	Average (docs, mail contact)	High
JACK	No	Dynamic and complex environments	User friendly GUI, easy to learn	High	High	High (extensive documentation and supporting tools)	High
EMERALD	Yes	Distributed applications composed of autonomous entities	User-friendly GUI, many familiar features (based on JADE platform), easy to learn	High (on JADE)	High (on JADE)	Average (documentation, mail contact)	Low

ALGORITHM 7: When provider agents receive CFP(*BrokerAgent*, *Service_Description*)

```
Evaluate the service request
if the request is beyond its capability then
    Send CONFIRM(ProviderAgent, REJECT) to the broker agent
else
    Send CONFIRM(ProviderAgent, ACCEPT) to the broker agent
endif
```

which cause difficulty in repeating experiments under the same conditions, and also adjusting one single index and its value in each experiment and accurately measuring the impact of our methods on service matching. Therefore, we evaluate our approach using JADE [57], a multi-agent framework by deploying it on the Ali-Cloud server ECS, configured with a four-core CPU, 16G memory and operating system Windows Server2012 64bit Chinese Edition.

6.1.1 The Selection of the Simulation Tool. In the past two decades, to study multi-agent systems, the research community and many companies have developed a variety of agent platforms, which give researchers many choices. Currently, JADE is the most popular, since it is industry driven, free, open source, high performance, and stable. Reference [58] analyzed, classified, and compared 24 multi-agent simulation tools in many aspects. Here, we select 4 that are in complete compliance with FIPA specifications for comparison, because FIPA standards specify many powerful and well tested protocols that enable agents to cooperate and interoperate in the dynamic, flexible and reconfigurable cases. From Table 8, we can see that JADE has more advantages over the others.

6.1.2 The Simulation System Based on JADE. The agent-based cloud market system comprises five parts: communication network, JADE main container, agent libraries, agent behaviors, and runtime interface. Figure 10 shows the overall design of the simulation platform.

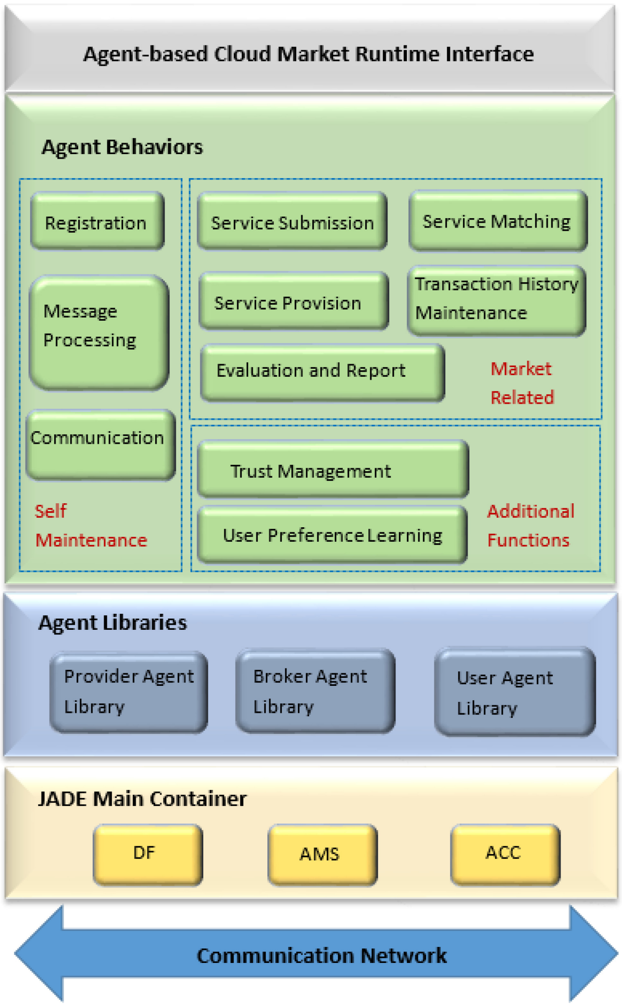

Fig. 10. Overall design of the simulation system.

The communication network provides the interactive environment and communication channels for different types of agents. JADE's main container provides a runtime environment for the execution of all the other agents, which is created automatically when a JADE platform starts. It consists of three components: Agent Management System (AMS), Directory Facilitator (DF), and Agent Communication Channel (ACC). AMS provides white page and life cycle services, maintains Directory of Agent Identifiers (AID) and the states of all agents. DF provides a yellow page service. ACC controls all the message exchanges of the platform.

The agent library is the container of agent instances. Despite the automatically generated agents of JADE, in TSLAM, there are three types of agents: provider agents, broker agents, and user agents. Thus, agent libraries are used to store instances of the agents. In a self-organizing system, agents use behaviors to sense the surroundings, and they also use behaviors to actively adjust and influence the surroundings. In TSLAM, we design three types of agent behaviors: self-maintenance behaviors, market-related behaviors, and additional function behaviors. Self-maintenance behaviors contain the basic behaviors for keeping the survivability, dynamics, and sociality of an agent

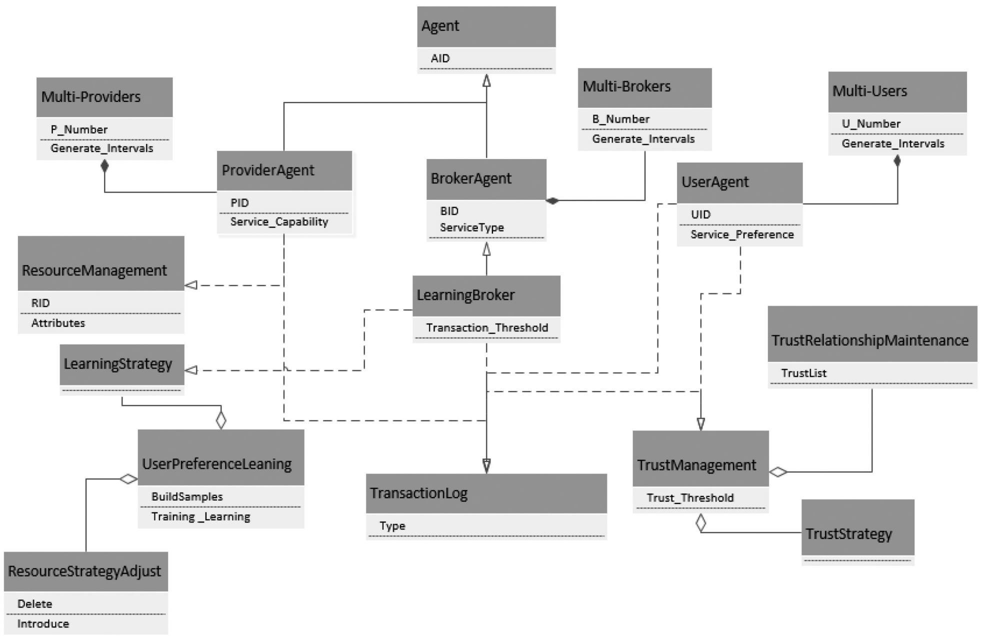

Fig. 11. UML diagram of agent classes.

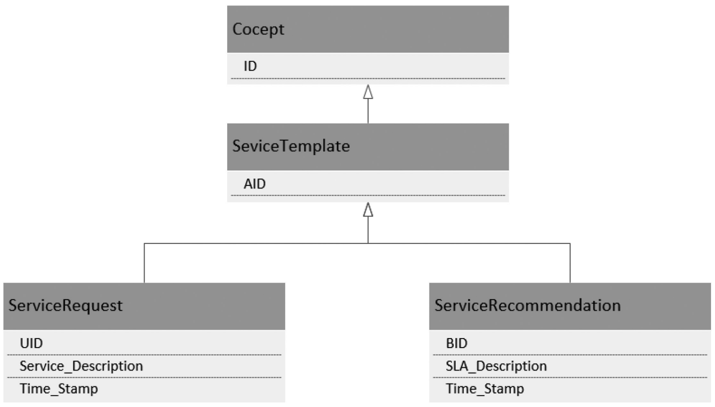

Fig. 12. UML diagram of service classes.

in the lifecycle. Market-related behaviors are designed to carry out the transactions and service evaluations in a cloud market. Additional function behaviors include the active learning mechanisms and the trust management mechanisms. The learning mechanisms are mainly found in broker agents, helping them analyze their customers' service preferences and optimize resources. The trust management mechanisms ensure cloud entities conduct transactions under credible conditions thus improving the success rate of transactions.

The runtime interface is a user-friendly interface that allows observers to observe the transactions in the market.

The simulation system is developed on NetBeans IDE 8.1. Figures 11 and 12 show the UML diagram of the agents.

The main ideas of the development are as follows. First of all, we start Multi-Brokers, Multi-Users, Multi-Providers three agents after launching the IDE. Multi-Brokers will automatically generate the specified number of LearningBrokers, Multi-Users will automatically generate the specified number of UserAgents, and Multi-Providers will automatically generate the specified number of ProviderAgents. Each market agent, such as LearningBroker, UserAgent, and ProviderAgent, randomly completes its own parameter configuration at the time of creation, then registers with the JADE Main Container and requests a list of global agents. Under the control of a certain probability and saturation, UserAgent will register with some LearningBrokers, and ProviderAgent is also the case. Those successfully registered ProviderAgents form the initial provider collection of several LearningBrokers. Because the registration is done at a certain probability, this ensures the initial resource structure and customer group for each broker are random. In the process of market transactions, each UserAgent issues a new service request periodically according to its preference. The familiar LearningBrokers perform service recommendation, and then the UserAgent makes a selection. Then the UserAgent and the ProviderAgent directly interact. After the transaction, UserAgent sends the service evaluation to the LearningBroker.

The main process of learning is as follows: When a LearningBroker recommends a service for a certain UserAgent, it decides whether the service sample can be added into the learning base according to a certain threshold of service satisfaction and also whether the recommended service is adopted or not. In addition, each time when the number of transactions accumulates to a certain amount, a LearningBroker starts the learning algorithm and obtains the user preference to guide the next iteration selection of providers.

In the simulation system, ServiceRequest class and SeviceRecommendation class both inherit from the Concept class. Moreover, to distinguish the service requests initiated by a same user but at a different time, a Time_Stamp attribute is supplemented by the class.

6.1.3 The Implementation of the Simulation Environment. There are many types of cloud services, such as cloud storage services, cloud virtual machine services, cloud database services, and so on, provided by different providers and each having different performance attributes. Although different services differ in their attributes, service matching processes are almost the same. Therefore, without loss of generality, we select the cloud server service as the targeted service in our experiments.

Take the cloud server service offered by the largest cloud service provider Ali-Cloud as an example. The main service parameters comprise three categories: instance (including operating system, geography, CPU + memory, price), disk (including disk type, capacity, price), and bandwidth (including specifications and price). The type of operating system and the geography are both the hard constraints and have little effect on the service classification in the virtualized scenario, therefore, in the simulation experiments, we select (1) CPU+memory (ram), (2) disk capacity (space), (3) bandwidth (bd), and (4) price as the metrics. To better demonstrate the reliability and availability of services, we also include an indicator (5) reliability (reliable). The value of each attribute in the following experiments is derived from the SLA of the Ali-Cloud ECS service [59].

Table 9 shows the principal types of ECS and their SLA parameters. As can be seen from the data in the table, Ali-Cloud provides corresponding solutions for different demands, and even for the same type of demand, it provides a wide range of performance options. However, this is not conducive to the analysis of user preferences in the subsequent experiments. To simplify the classification problem and highlight the differences between the different types of user requirements, based on the Ali-Cloud ECS service types, we let the value of a service be in the reference range but take a relatively large value on the corresponding category attribute, while taking a small value

Table 9. The SLA Parameters of Ali-Cloud ECS Service

Service Type	CPU + Ram	Disk Capacity	Bandwidth	Price/year
Beginner	1core + 0.5GB → 16core + 32GB	20—500GB	0.1—1.2Gbps	From ￥570.00
Universal	2core + 8GB → 64core + 256GB	20—500GB	1—20Gbps	From ￥2070.00
Computation	2core + 4GB → 64core + 128GB	20—500GB	1—20Gbps	From ￥1643.00
Ram	2core + 16GB → 56ore + 480GB	20—500GB	1—20Gbps	From ￥2646.00
Storage	8core + 32GB → 56ore + 224GB	20—500GB	3—17Gbps	From ￥16497.00
Local SSD	4core + 32GB → 64ore + 512GB	20—500GB	0.8—10Gbps	From ￥9036.00
High frequency computation	2core + 4GB → 56ore + 160GB	20—500GB	1.5—10Gbps	From ￥2499.96
GPU	2core + 8GB → 54ore + 480GB	20—500GB	10—25Gbps	From ￥5697.00
FPGA	8core + 60GB → 56ore + 224GB	20—500GB	5—10Gbps	From ￥22455.00
Ebmg5	8core + 32GB → 96ore + 384GB	20—500GB	6—10Gbps	From ￥15493.68

Table 10. The Main Parameters of the Experiments

User number	Broker number	Provider number	Service frequency	Max connected brokers	Max managed providers	Service type
50	20	1000	1/min/user	5/user	100/broker	reliable/bandwidth/ space/ram/price

on the other attributes. In addition, to eliminate the impact of the dimension, we normalize the data between 0 and 100.

The initialization phase of the simulation experiment generates a series of user agents, broker agents and provider agents. Each user agent has its unique preference, therefore, its service requests are generated randomly on the basis of the preference. Each provider agent also has the above five properties, in accordance with the performance differences, which are randomly assigned a value between 1 and 100. User agents at regular intervals (in this experiment it is per minute) produce a new service request, and they only tell the broker agent part of their demands, say, only three dimensions of the accurate demands, i.e., using a semi-open mode. However, when the transactions are finished, the service satisfaction is evaluated according to the complete service demands. Therefore, the broker agent does not know the exact user service preference and has to sum up and learn in the long run after a number of services have been supplied. Table 10 shows the parameters of the subsequent experiments.

To verify the performance of TSLAM, the experiments are conducted from the following three aspects: (1) whether or not the broker agent can accurately learn the user agents' service preferences with an increasing number of iterations, and whether or not they can determine the mainstream service type on their own so as to achieve market differentiation; (2) in the dual role of the trust and learning mechanism, whether or not the success ratio and user satisfaction can be improved and whether or not the cloud market can evolve faster; and (3) the efficiency of the trust mechanism.

6.2 Experiment 1: The Effect of the User Preferences Learning Model

Fifty user agents were generated randomly in the experiment, each of which chose one service preference from a total of five kinds (price, ram, space, bd, reliable) randomly in the initial stage. At the same time, twenty broker agents and one thousand service providers were produced. Broker

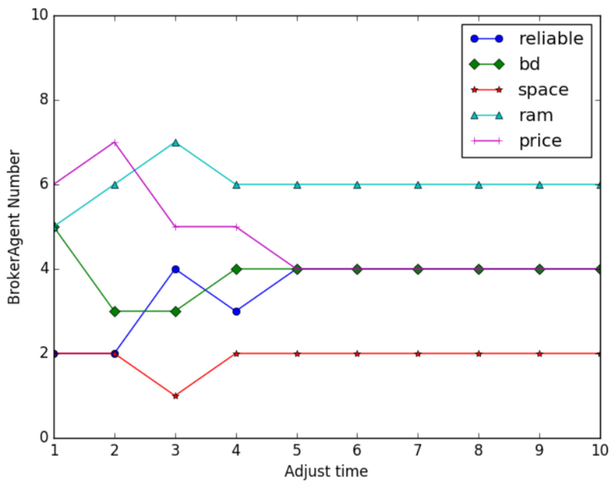

Fig. 13. Cloud market differentiation revolution in the ideal environment.

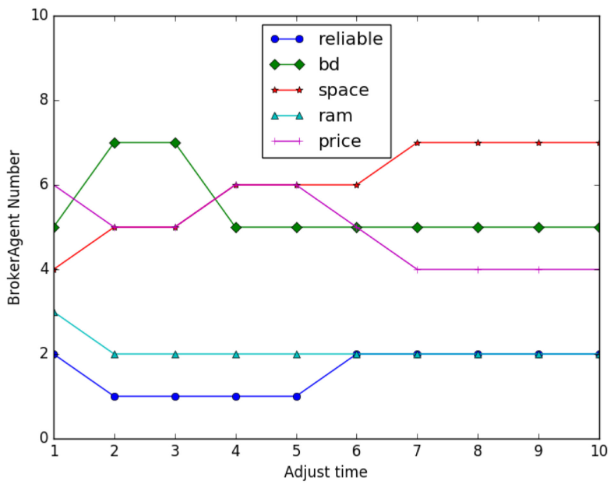

Fig. 14. Cloud market differentiation evolution in the presence of 10% malicious providers.

agents were responsible for putting up a communicating bridge between users and providers, and they needed to learn the service preferences of their users constantly during the transactions. Provider agents were responsible for providing services with different performance, reliability and cost. The purpose of the experiment was to observe the division process of the broker agents. We conducted two groups of experiments, one of which was in an ideal environment, in which providers always provided services in accordance with the quality of claims, the other was in the presence of a small amount (10%) of malicious providers who provided services below their stated quality. The differentiation results of the broker agents are shown in Figures 13 and 14.

In the chart, the vertical axis represents the number of brokers whose main service type is one of the five in the market and the horizontal axis represents the number of iterations. Thus, changes in the curve reflect the changing number of brokers with different service types reflecting the influence of the learning mechanism. It can be seen from the experiment results that in the early iteration, the service tendency of each broker agent varied, showing that the learning process continued and the differentiation was not very clear, then at a later stage, changes gradually eased, after which ultimately, the broker agent' preferences no longer changed, which indicates not only the convergence of the algorithm but also the cloud market reaching an equilibrium state. In addition, the comparison of Figures 13 and 14 shows that this algorithm, regardless of the ideal state, or in the presence of malicious providers was able to ensure the effectiveness of the differentiation. However, convergence speed under ideal conditions was relatively faster.

6.3 Experiment 2: Performance Comparison of Cloud Market under Different Strategies

In TSLAM, the trust mechanism is used to find the "bad" trading entities. Here, "bad" or "malicious" indicates the behaviors of providers that do not provide services in accordance with the declared quality, do not respond in a timely manner caused by congestion, make false recommendations, and so on. At the same time, the learning mechanism in TSLAM helps brokers learn the users' service preferences and determine their market positioning.

To better analyze the performance of TSLAM, several models were compared: (1) the proposed model in this article (TSLAM), (2) agent-based cloud service matching model [33] (G&Sim model), (3) T-broker model [9], (4) the price priority model (Price model), (5) the performance priority model (Performance model), and (6) random trading model (Random model).

The G&Sim model is one of the most representative agent-based cloud service matching models. The T-Broker model appears to be similar to TSLAM, which is also a trust-based service matching model. The latter three models are the subtraction models of TSLAM, which are obtained by removing some mechanisms from the complete proposed model in this article. The Price model and Performance model remove the learning part of TSLAM and adopt different recommendation rules. The Price model aims to minimize cost, while the Performance model recommends services with the best function attributes. The Random model removes both the learning and trust mechanism from TSLAM and uses a random recommendation method.

Although the other models do not consider a situation where implicit requirements exist, this doesn't matter, because in the service matching step, brokers can calculate the similarity between the open part of a request and the corresponding part of a service. Moreover, after the transactions, users evaluate the services based on their complete demands, which ensures user satisfaction is comprehensive and meaningful.

However, another problem is that T-broker is actually essentially different from TSLAM, because it uses a centralized broker. In the case of a centralized broker, since there is only one service matching broker in the market, without any other options, users have to accept the recommendation, so the transaction success rate can be always maintained at a high level. Thus, comparing the success rate and convergence rate with T-broker doesn't make much sense. Therefore, in the following experiments, T-broker is only used in the comparison of user satisfaction.

In this article, user satisfaction refers to the identical degree of the users' actual obtained resource level with their expected resource level. Since we consider five aspects of QoS requirements: price, ram, space, bandwidth, and reliability, we use Equation (10) to calculate a single user's satisfactory degree,

$$Sat_i = \ln\left(\frac{\mu(Eprice_i)}{Price_i} + \frac{\rho(Eram_i)}{Ram_i} + \frac{\varphi(Espace_i)}{Space_i} + \frac{\tau(Ebd_i)}{Bd_i} + \frac{\omega(Ereliable_i)}{Reliable_i}\right), \quad (10)$$

where $Eprice_i$ is the expected price of $user_i$. Accordingly, $Eram_i, Espace_i, Ebd_i, and\, Ereliable_i$ are the expected ram, space, and bandwidth of $user_i$. The total user satisfaction is the average value

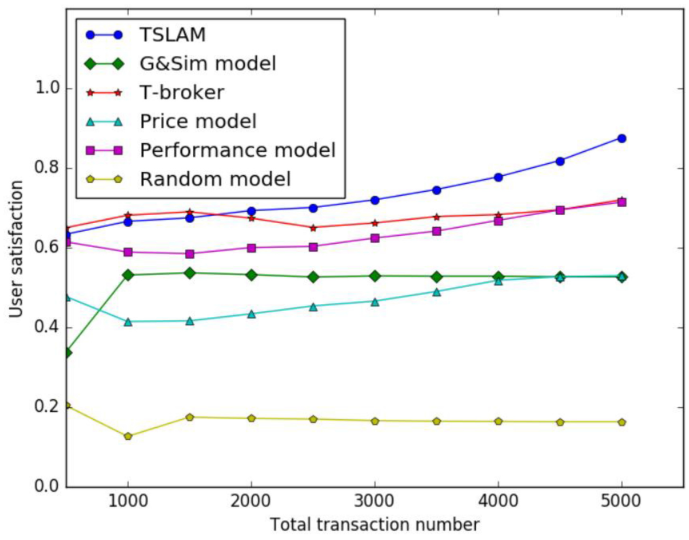

Fig. 15. Empirical result of user satisfaction under different strategies (without malicious providers).

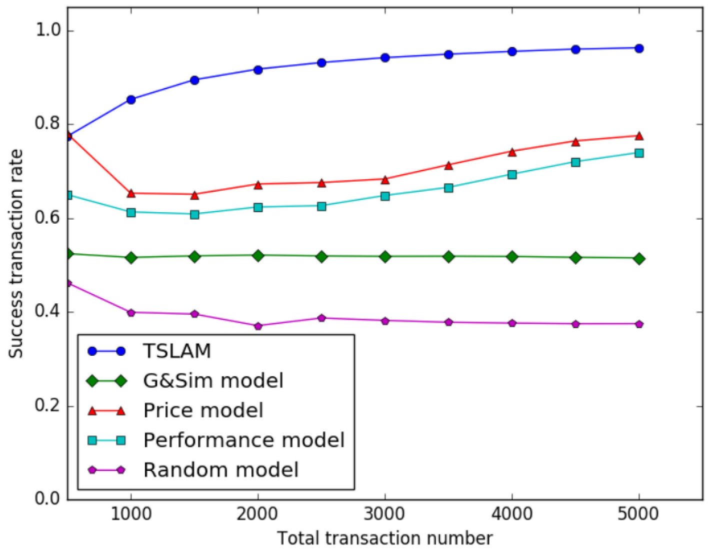

Fig. 16. Empirical result of transaction success rate under different strategies (without malicious providers).

of all the users,

$$Sat = \frac{1}{n}\sum_{i=1}^{n} Sat_i. \tag{11}$$

We carried out two sets of experiments in different situations: (1) an ideal cloud market environment (without any malicious providers) and (2) an unstable market environment (with 40% malicious providers). Figures 15, 16, and 17 show the experiment results of the ideal market environment and Figures 18 and 19 show the results of the second situation.

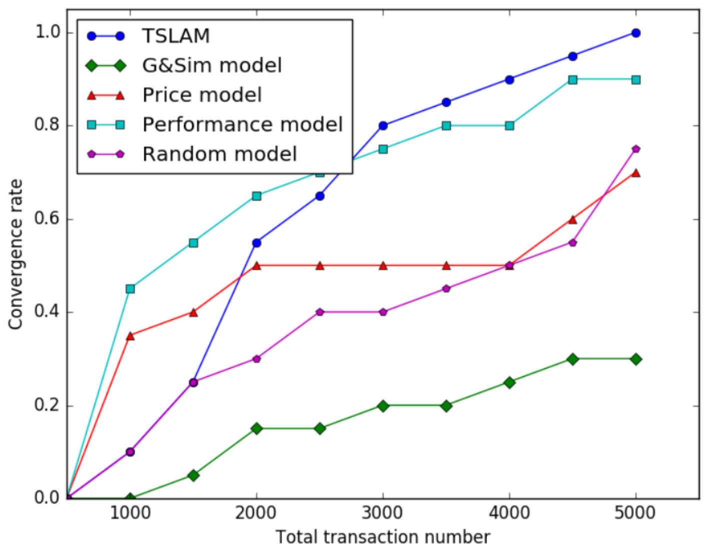

Fig. 17. Empirical result of convergence speed under different strategies (without malicious providers).

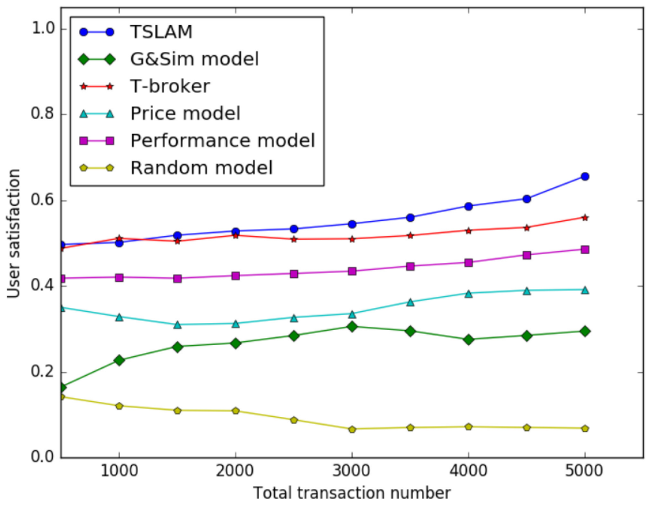

Fig. 18. Empirical result of user satisfaction under different strategies (with 40% malicious providers).

The experiment results of these different strategies are compared. The horizontal coordinate represents the total number of transactions. Figure 15 reflects the changed curve in user satisfaction, Figure 16 reflects the variation in the transaction success rate, and Figure 17 shows the change in the curve of the market convergence ratio (the volatility of the stability of the broker in a very small range). By processing, the results of the experiment data were limited to within 0–1.

From the experiment results, we can see that the transaction success rate and the user satisfaction of TSLAM are higher than the other models. The random trading model has the worst effect, which indicates that the free market is chaotic and inefficient. TSLAM is equipped with a learning module, which ensures the brokers gradually learn user preferences through the accumulation of transaction volume. Thus, in the new round of the provider introduction, brokers are able to select

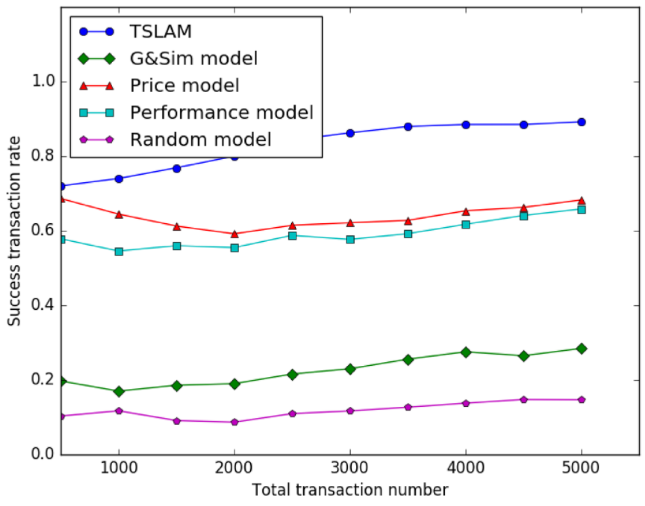

Fig. 19. Empirical result of success transaction rate under different strategies (with 40% malicious providers).

providers based on the preferences, thereby optimizing its resource structure and making it closer to the users' demands. From the curve, we can see that user satisfaction and success rate are always on the rise. Without a learning ability, the other models do not perform as well as TSLAM. However, T-broker uses a centralized mechanism, and the only broker in the market manages all the service providers. This measure enables the broker to implement recommendations for users, so the initial satisfaction is higher than TSLAM. Nevertheless, T-broker has no learning ability for the potential demands of users, so it can maintain the performance at a certain level, but not at a significant rise. The Price model and the Performance model remove the learning module from TSLAM. So, when faced with users' implicit requirements, they use a one-size-fits-all approach, either minimizing the cost or maximizing the functional performances, hence their satisfaction is lower than TSLAM. The G&Sim model is a similar situation.

From the perspective of convergence rate, TSLAM converges faster than the other models. The reason for this is that after a period of transactions, brokers are clear about their customers' real requirements, and the providers that are managed by one broker gradually become saturated as the recommendation success rate and the positive samples increase, which indicates that the status of the broker is stable and the service classification is finished. However, we also notice that although there are no trust or learning mechanisms in the other models, with the increasing number of transactions, the transaction success rate and user satisfaction gradually increase, despite the pace of ascension being slower than TSLAM. This can be explained by the fact that an implicit learning mechanism does exist in the fundamental layer of the traditional markets that is gained through the numerous direct exchanges and transactions between consumers and providers, despite the fact that this process is very slow.

The second set of experiments has 40% of malicious providers in the market. To highlight the contrast effect, malicious providers here are not those who do not provide services according to their SLAs but are those who do not provide services at all. The purpose of the experiments is to observe the impact of trust plus the learning mechanism on service matching. Figures 18 and 19 show the results. It can be seen that the G&Sim model and Random model are most affected by the untrusted nodes, and their performance drops obviously, because they do not have a trust mechanism. T-broker has a trust-equipped broker to help users find services. Price model and Performance model retain the trust module of TSLAM. Trust-enabled service matching models help

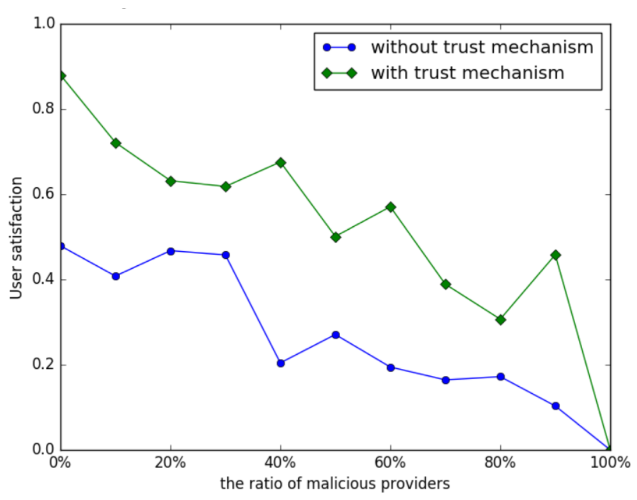

Fig. 20. Empirical results of user satisfaction with different ratios of malicious providers.

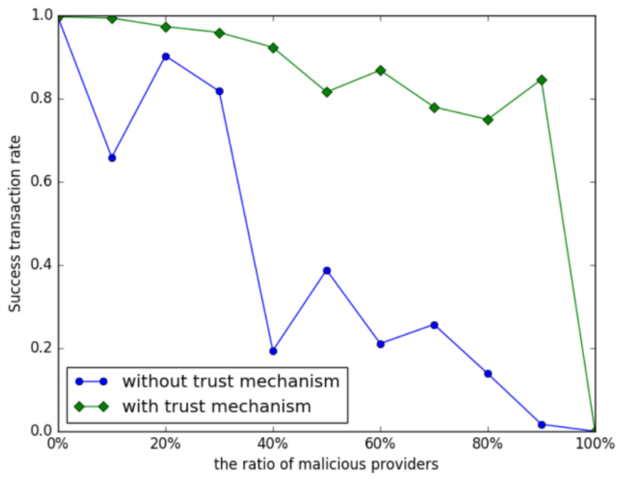

Fig. 21. Empirical results of transaction success rate with different ratios of malicious providers.

cloud entities to gradually know the integrity of others, and hence avoid trading with malicious nodes. All these strategies reduce the likelihood of invalid or false transactions and improve user satisfaction. Moreover, TSLAM again performs better than the other models in terms of satisfaction and success rate owing to the additional learning ability.

In conclusion, the learning and trust mechanisms in TSLAM improve the efficiency of the cloud services market and accelerates market differentiation.

6.4 Experiment 3: Evaluation of the Effectiveness of the Trust Mechanism

This set of experiments individually assesses the impact of the trust mechanism on the performance of the cloud market. Different proportions of malicious nodes, including malicious provider nodes and malicious broker nodes, were specially added in the experiments. Malicious providers refer to those providers who refuse to serve in accordance with the claims of the SLA or perform a denial of service attack on the connected users because of service congestion or other reasons.

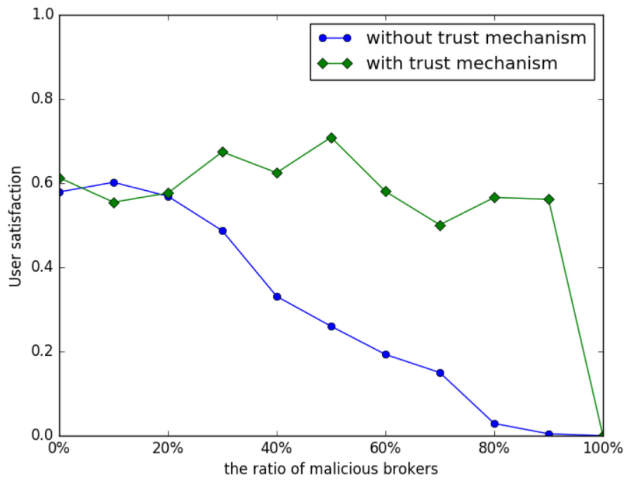

Fig. 22. Empirical results of user satisfaction with different ratios of malicious brokers.

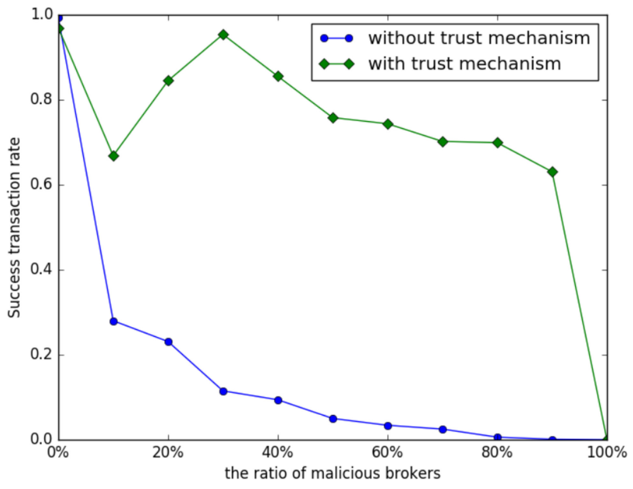

Fig. 23. Empirical results of transaction success rate with different ratios of malicious brokers.

Malicious brokers are those who provide false recommendations or close the service interface after the users have accepted their recommendation. Figures 20–23 show the results of the experiments.

Figures 20 and 21 reflect the results of the experiments by adding different proportions of malicious providers. The abscissa represents the malicious provider proportion. To evaluate whether the trust mechanism can effectively assist the other trading mechanisms, the timely detection of the malicious nodes and market stability maintenance, the proportion of malicious providers in the experiment is increased from 0% to 100%, although the ratio of malicious nodes in the real market may not be so high.

It can be seen from the experiment results that even if bad providers who refuse to provide services according to the SLA exist, a trust-injected cloud service trading model can still ensure the overall success rate of transactions and customer satisfaction.

Figures 22 and 23 show the experiment results with different proportions of malicious brokers. It can be seen that even if some bad brokers who occasionally offer false recommendations exist,

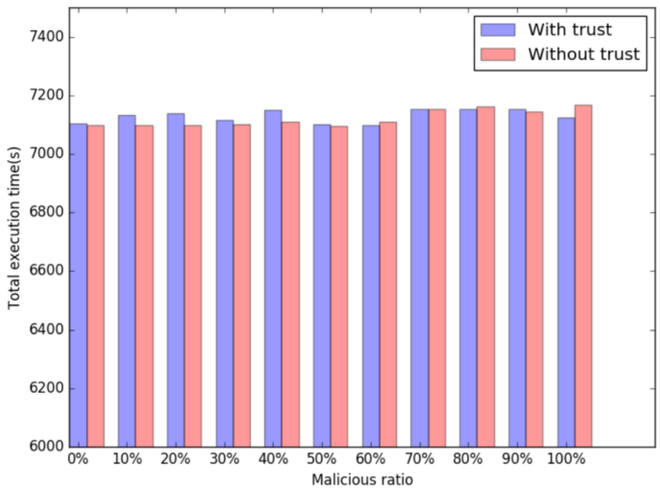

Fig. 24. Effect of trust on total execution time.

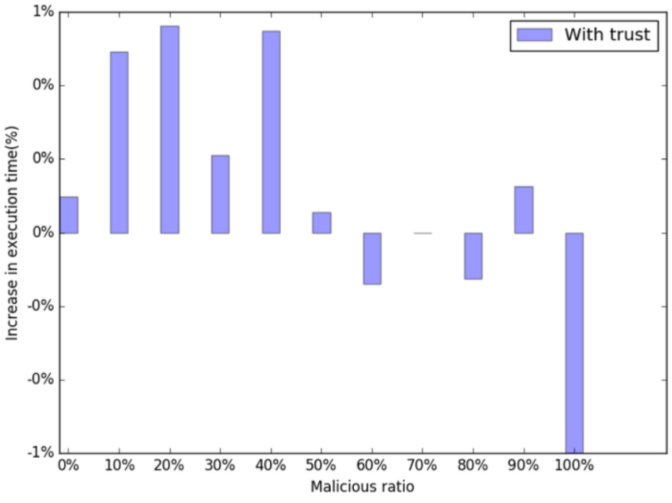

Fig. 25. Percentage of increased time overhead by trust.

trust mechanism-assisted cloud market trading strategy is still able to maintain a high transaction success rate, customer satisfaction and market stability.

To measure the overhead of trust, we also compared the impact on the total execution time with and without trust. The results in Figures 24 and 25 show that the total execution time is very close. Figure 25, which is quantified by a percentage of the additional time overhead, indicates that the actual increased system execution time is negligible in large-scale transactions. In addition, a few nodes show a shorter execution time when trust is loaded, because after the trust mechanism has been running for a period of time, since the users have accumulated the reputation of others, they are more likely to select a reliable transaction partner, thereby avoiding the re-selections in the random transactions, thus, to a certain extent, reducing the time.

7 CONCLUSIONS AND FUTURE WORK

This article presented a novel cloud market model based on the multi-agent platform and trust mechanism. The new model uses smart agents to replace the behaviors and targets of the different entities in the cloud market, comprising a three-layered cloud trading structure. Relying on the dual role of trust and learning mechanisms, the efficiency of the market is improved and market differentiation is accelerated. The introduction of the trust mechanism helps the cloud entity better identify honest and dishonest candidates in the market, thus improving the ratio of successful transactions. The learning mechanism helps brokers better analyze their customers' service preferences for subsequent development, accelerates the classification and differentiation, and ultimately realizes market convergence. Three kinds of simulation experiments based on the JADE platform verify the validity of the model from different perspectives.

However, there are still some imperfections in TSLAM. For example, the number of providers managed by a broker has a definite limit (called saturation). Obviously, the more providers a broker manages, the easier he can find a suitable service for the user. But at the same time, the cost of management and selection increases, which means it is not the case where the bigger the better. Unfortunately, at present, we have not drawn a final conclusion as to the most suitable number. Furthermore, we also need to investigate how often the learning module should run, how many transaction times is the best base for learning, what is the best method to generate positive and negative samples, and so on. All these problems will become our future work.

ACKNOWLEDGMENTS

The authors thank the editor and the anonymous reviewers for their insightful and constructive comments and suggestions on improving this article.

REFERENCES

[1] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th Utility. *Fut. Gener. Comput. Syst.* 25, 6 (2009), 599–616.

[2] P. Liu. 2015. *Cloud Computing* (3rd ed.). Electronic Industry, Beijing, China.

[3] China Information and Communication Research Institute. 2016. Cloud Computing White Paper. Technical Report. China Information and Communication Research Institute.

[4] T. Cuong, H. Nguyen, E. Huh, and C. Hong et al. 2016. Dynamics of service selection and provider pricing game in heterogeneous cloud market. *J. Netw. Comput. Appl.* 69 (2016), 152–165.

[5] G. Hao, Y. Jun, and M. Yi. 2014. Trust-oriented QoS-aware composite service selection based on genetic algorithms. *Concurr. Comput.: Pract. Exper.* 26, 2 (2014), 500–515.

[6] M. Abourezq and A. Idrissi. 2015. Integration of Qos aspects in the cloud service research and selection system. *Int. J. Adv. Comput. Sci. Appl.* 6, 6 (2015), 111–122.

[7] S. Yan, X. Zheng, and D. Chen. 2010. A user-centric trust and reputation method for service selection. In *Proceeding of the International Symposium on Intelligence Information Processing and Trusted Computing*. IEEE, 101–105.

[8] Y. Kim and K. Doh. 2013. Quantitative trust management to support QoS-aware service selection in service-oriented environments. In *Proceedings of the IEEE International Conference on Parallel and Distributed Systems.* IEEE, 504–509.

[9] X. Li, H. Ma, F. Zhou, and W. Yao. 2015. T-Broker: A trust-aware service brokering scheme for multiple cloud collaborative services. *IEEE Trans. Inf. Forens. Secur.* 10, 7 (2015), 1402–1415.

[10] W. Li, J. Wu, Q. Zhang, K. Hu, and J. Li. 2014. Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies. *Clust. Comput.* 17, 1 (2014), 1013–1030.

[11] K. Sim. 2006. A survey of bargaining models for grid resource allocation. *ACM SIGECOM: E-comm. Exch.* 5, 5 (2006), 22–32.

[12] M. Mamei and F. Zambonelli. 2003. Self-organization in multi agent systems: A middleware approach. In *Proceedings of the International Workshop on Engineering Self-Organising Applications (ESOA'03).* Springer, Berlin, 233–248.

[13] J. Holland and J. Miller. 1991. Artificial adaptive agents in economic theory. *Am. Econ. Rev.* 81, 2 (1991), 365–370.

[14] A. Toosi, K. Vanmechelen, F. Khodadadi, and R. Buyya. 2016. An auction mechanism for cloud spot markets. *ACM Trans. Auton. Adapt. Syst.* 11, 1 (2016).

[15] R. Buyya, C. Yeo and S. Venugopal. 2008. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, 5–13.

[16] B. Song, M. Hassan, and E. Huh. 2009. A novel cloud market infrastructure for trading service. In *Proceedings of the International Conference on Computational Science and Its Applications*. IEEE, 44–50.

[17] S. Arifulina, F. Mohr, G. Engels, M. Platenius, and W. Schafer. 2015. Market-specific service compositions: Specification and matching. In *Proceedings of the IEEE World Congress on Services*. IEEE, 333–340.

[18] E. Badidi. 2013. A cloud service broker for SLA-based SaaS provisioning. In *Proceedings of the International Conference on Information Society (i-Society'13)*. IEEE, 61–66.

[19] T. Deng. 2017. Analysis of user behavior in cloud broker. In *Proceedings of the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS'17)*. IEEE, 157–160.

[20] D. Rane and A. Srivastava. 2015. Cloud brokering architecture for dynamic placement of virtual machines. In *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 661–668.

[21] M. Aazam and E. Huh. 2014. Broker as a service (BaaS) pricing and resource estimation model. In *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, 463–468.

[22] S. Aldawood, F. Fowley, C. Pahl, D. Taibi, and X. Liu. 2016. A coordination-based brokerage architecture for multi-cloud resource markets. In *Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW'16)*. IEEE, 7–14.

[23] S. Wagle. 2014. SLA assured brokering (SAB) and CSP certification in cloud computing. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 1016–1017.

[24] O. Wenge, D. Schuller, and R. Steinmetz. 2014. Towards establishing security-aware cloud markets. In *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, 1027–1032.

[25] P. Pawar, M. Rajarajan, T. Dimitrakos, and A. Zisman. 2014. Trust assessment using cloud broker. In *Proceedings of the 8th IFIP International Conference on Trust Management (IFIPTM'14)*. Springer, 237–244.

[26] W. Abderrahim and Z. Choukair. 2015. Trust assurance in cloud services with the cloud broker architecture for dependability. In *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC'15)*. IEEE, 778–781.

[27] M. Gupta and B. Annappa. 2016. Trusted partner selection in broker based cloud federation. In *Proceedings of the 2016 International Conference on Next Generation Intelligent Systems (ICNGIS'16)*. IEEE, 1–6.

[28] K. Sim. 2009. Agent-based cloud commerce. In *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, 717–721.

[29] K. Jun, L. Boloni, and K. Palacz et al. 2000. Agent-based resource discovery. In *Proceedings of the 9th IEEE Heterogeneous Computing Workshop*. IEEE, 43–52.

[30] K. Sim. 2006. Grid commerce, market-driven G-negotiation, and grid resource management systems. *IEEE Trans. Manage. Cybernet. B* 36, 6 (2006), 1381–1394.

[31] K. Sim. 2013. Complex and concurrent negotiations for multiple interrelated e-markets. *IEEE Trans. Cybernet.* 43, 1 (2013), 230–245.

[32] K. Sim. 2010. Towards complex negotiation for cloud economy. In *Proceedings of the International Conference on Grid and Pervasive Computing (GPC'10): Advances in Grid and Pervasive Computing*. Springer, Berlin, 395–406.

[33] J. Gutierrez-Garcia and K. Sim. 2015. Agent-based cloud bag-of-tasks execution. *J. Syst. Softw.* 104, 6 (2015), 17–31.

[34] S. Son and K. Sim. 2015. Adaptive and similarity-based tradeoff algorithms in a price-timeslot-QoS negotiation system to establish cloud SLAs. *Inf. Syst. Front.* 17, 3 (2015), 565–589.

[35] X. Li and X. Gui. 2010. Cognitive model of dynamic trust forecasting. *J. Softw.* 21, 1 (2010), 163–176.

[36] Y. Tan and C. Wang. 2015. Trust evaluation based on user behavior in cloud computing. *Microelectron. Comput.* 32, 11 (2015), 147–151.

[37] S. Sanadhya and S. Singh. 2015. Trust calculation with ant colony optimization in online social networks. *Proc. Comput. Sci.* 54, 8 (2015), 186–195. DOI: https://doi.org/10.1016/j.procs.2015.06.021

[38] E. Ugur, S. Sen, and A. Burak. 2015. GenTrust: A genetic trust management model for peer-to-peer systems. *Appl. Soft Comput.* 34, 9 (2015), 693–704.

[39] S. Wang, L. Zhang, and H. Li. 2010. Evaluation approach of subjective trust based on cloud model. *J. Softw.* 21, 6 (2010), 1341–1352.

[40] X. Xie, L. Liu, and P. Zhao. 2012. Trust model based on double incentive and deception detection for cloud computing. *J. Electr. Inf. Technol.* 34, 4 (2012), 812–817.

[41] K. Ahmadi and V. Allan. 2016. Trust-based decision making in a self-adaptive agent organization. *ACM Trans. Auton. Adapt. Syst.* 11, 2, Article 10 (2016), 25.

[42] W. Li, L. Ping, and X. Pan. 2009. Trust model to enhance security and interoperability of cloud environment. In *Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09)*. Springer, Berlin, 69–79.

[43] W. Li, L. Ping, and X. Pan. 2010. Use trust management model to achieve effective security mechanisms in cloud environment. In *Proceedings of the 1st International Conference on Electronics and Information Engineering (ICEIE'10)*. IEEE, 14–19.

[44] W. Li, L. Ping, Q. Qiu, and Q. Zhang. 2012. Research on trust management strategies in cloud computing environment. *J. Comput. Inf. Syst.* 8, 4 (2012), 1757–1763.

[45] X. Li, J. He, and Y. Du. 2015. Trust based service optimization selection for cloud computing. *Int. J. Multimedia Ubiq. Eng.* 10, 5 (2015), 221–230.

[46] C. Hu, J. Liu, and J. Liu. 2011. Services selection based on trust evolution and union for cloud computing. *J. Commun.* 32, 7 (2011), 71–79.

[47] Y. Wang, J. Zhou, and H. Tan. 2015. CC-PSM: A preference-aware selection model for cloud service based on consumer community. *Mathematical Problems in Engineering*, Hindawi Publishing Corporation Article ID 170656 (2015), 13.

[48] X. Meng, J. Ma, D. Lu, and Y. Wang. 2014. Trust and behavioral modeling based two layer service selection. *J. Xidian Univ.* 41, 4 (2014), 198–204.

[49] S. Yan and X. Zheng. 2010. A user-centric trust and reputation method for service selection. In *Proceedings of the 2010 International Symposium on Intelligence Information Processing and Trusted Computing*. IEEE, 101–105.

[50] C. Hang and M. Singh. 2011. Trustworthy service selection and composition. *ACM Trans. Autonom. Adapt. Syst.* 6, 1, Article 5 (2011), 17.

[51] H. Wang, C. Yu, L. Wang, and Q. Yu. 2015. Effective bigdata-space service selection over trust and heterogeneous QoS preferences. *IEEE Trans. Serv. Comput.* 11, 4 (2015), 644–657.

[52] B. Cao, B. Li, and J. Liu. 2013. An on-demand service composition method based on trustworthy quality of service. *J. Xi'an Jiaotong Univ.* 47, 2 (2013), 131–138.

[53] R. Du, J. Tian, and H. Zhang. 2013. Cloud service selection model based on trust and personality preferences. *J. Zhejiang Univ. (Eng. Sci.)* 47, 1 (2013), 53–61.

[54] W. Li, J. Wu, J. Cao, and K. Hu. 2016. Trust-based multi-attribute decision resource location algorithm for peer-to-peer cloud systems. *Syst. Eng.- Theory & Practice* 36, 4 (2016), 1047–1056.

[55] W. Li, L. Ping, J. Li, and Q. Qiu. 2012. Cloud service discovery algorithm based on trust fuzzy comprehensive evaluation. *ICIC Expr. Lett. B* 3, 2 (2012), 1–6.

[56] W. Li, X. Pan, Q. Zhang, and L. Ping. 2011. A novel job scheduling model to enhance efficiency and overall user fairness of cloud computing environment. In *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER'11)*, 1–5.

[57] Jade company. JADE document. Retrieved from http://jade.tilab.com/.

[58] K. Kravari and N. Bassiliades. 2015. A survey of agent platforms. *J. Artif. Soc. Soc. Simul.* 18, 1 (2015), 1–18.

[59] Ali-Cloud. The SLA of Ali-Cloud ECS (plastic Compute Service). Retrieved from https://cn.aliyun.com/product/ecs?spm=5176.doc29692.416540.27.rz9CKZ

[60] J. Xu. 2015. A cloud service self-organizing based inter-cloud mechanism and its platform research. M.S. dissertation, Shanghai Jiao Tong University, 2015.

Received February 2018; revised February 2019; accepted March 2019