Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 8194832, 19 pages
http://dx.doi.org/10.1155/2016/8194832

Research Article

Research on Methods for Discovering and Selecting Cloud Infrastructure Services Based on Feature Modeling

Huamin Zhu, Lifa Wu, Kangyu Huang, and Zhenji Zhou

Institute of Command Information System, PLA University of Science and Technology, Nanjing, China

Correspondence should be addressed to Lifa Wu; wulifa@vip.163.com

Received 15 December 2015; Revised 24 July 2016; Accepted 24 August 2016

Academic Editor: Zhimin Huang

Nowadays more and more cloud infrastructure service providers are providing large numbers of service instances which are a combination of diversified resources, such as computing, storage, and network. However, for cloud infrastructure services, the lack of a description standard and the inadequate research of systematic discovery and selection methods have exposed difficulties in discovering and choosing services for users. First, considering the highly configurable properties of a cloud infrastructure service, the feature model method is used to describe such a service. Second, based on the description of the cloud infrastructure service, a systematic discovery and selection method for cloud infrastructure services are proposed. The automatic analysis techniques of the feature model are introduced to verify the model's validity and to perform the matching of the service and demand models. Finally, we determine the critical decision metrics and their corresponding measurement methods for cloud infrastructure services, where the subjective and objective weighting results are combined to determine the weights of the decision metrics. The best matching instances from various providers are then ranked by their comprehensive evaluations. Experimental results show that the proposed methods can effectively improve the accuracy and efficiency of cloud infrastructure service discovery and selection.

1. Introduction

The Infrastructure as a Service (IaaS) is an important service model in cloud computing, and it can provide users with fundamental computing resources, for example, processing, storage, and network, where users are able to deploy the desired operating systems and applications without paying attention to the management and control of the underlying cloud infrastructure [1]. IaaS is provided to users by a service instance that may be a different combination of processing, storage, and network resources that satisfies the diversified needs of the users. Cloud computing with potentially huge business opportunities has attracted a growing number of IT service providers to build infrastructures that can provide dozens, hundreds, or even tens of thousands of IaaS instances. Currently, there are over 100 public IaaS providers, and just for Amazon's Elastic Compute Cloud (EC2), the number of service instances is over 16000 [2]. How to select the best IaaS instance from these large numbers of alternatives has become a serious challenge to users.

Different IaaSs show differences in resource configuration and service quality, and thus the discovery and selection of the IaaS instances not only need to perform fine-grained matching on the resource configuration according to user needs, but also need to find the optimal instance based on comparison of the quality of service (QoS) of the matching instances. It is obviously a difficult and error-prone task to search the matching instances in the huge configuration space of an IaaS, which necessitate automatic searching methods to support it. Moreover, a set of scientific QoS metrics and an effective multicriterion decision-making method for ranking the matching instances are also needed. At present, although many scholars have done extensive research on the discovery and selection of IaaSs [3–10], the following challenges remain: (1) There exists a lack of customized description methods for IaaSs, and the prevalent natural language description cannot be used for automatic processing by computers. (2) The existing methods are unable to describe or formalize the configuration space of an IaaS completely; thus, these methods can only handle a subset of the entire configuration

spaces when discovering and selecting an IaaS, which results in a suboptimal service instance. (3) Due to the existing coarse-grained description of IaaSs, the matching between the user request and the IaaS offerings can only be done in a coarse-grained way, so that some service details, such as cost, cannot be accurately calculated. (4) Also lacking are systematic theory and method for describing, discovering, and selecting IaaSs. (5) There is no scientific and unified measurement metric system for the QoS evaluation of IaaSs. The existing IaaS decision-making methods assign metric weight mainly based on the subjective weighting method, which is likely to lead to an inaccurate evaluation of QoS.

From the above analysis, it is not difficult to see that the formalization and fine-grained description of the complete configuration space of an IaaS is the key to discovering and selecting a truly optimal IaaS instance. In general, the configuration space of an IaaS comprises a large number of service instances, and there are two ways of providing IaaS instances. The first is to provide great many standard instances from which users can choose; for example, in EC2, there are m4.large, c3.large, and r3.xlarge instances of the virtual server, and each instance has some basic options with fixed values as well as some configurable options, for example, m4.large (2.4 GHz Intel Processor, 2 vCPU, 8 GB RAM, EBS-only, 450 Mbps bandwidth) has configurable location, cost model, and operating system (OS) options [11]. The second way of providing an IaaS instance is to specify the value range of the configurable options, and any combination of all the option values within the range is an IaaS instance. There are some configurable service options in an IaaS using either approach, such as virtual machine (VM) type, RAM, OS, cost model, location, network bandwidth, and any combination of these options is an IaaS instance such that all the instances form the complete configuration space. Therefore, an IaaS can be regarded as a variability-intensive system, and it is pivotal to deal with the variability of an IaaS to fully obtain the fine-grained description of its entire configuration space.

Kang et al. [12] proposed a feature-oriented domain analysis method (FODA) in 1990. Here, features and the feature model (FM) are first introduced to capture and organize changing software requirements in special fields [13]. In a FM, features are used to represent the system functional and nonfunctional characteristics, and feature relations are used to depict the organizational structure and internal relations of these characteristics. Such an "entity-relation" description structure in a FM exhibits highly customizable properties, which has led FMs to become a very effective domain commonality/variability description method. With the continuous development of software reuse research, FMs have become the *de facto* standard for software product line commonality/variability modeling, and they can be used as a good description model for variability-intensive systems [14, 15]. Therefore, it is very appropriate to use a FM to describe an IaaS. In this paper, we design a formal description method for IaaSs by means of feature modeling and propose a systematic discovery and selection method for IaaSs based on the description method. The main contributions in this paper are as follows:

(i) A formal IaaS description method based on FMs is designed, which is able to describe the complete configuration space of an IaaS in a standard way. We define the metamodel of an IaaS using the unified modeling language (UML) on the basis of existing methods and then define the IaaS domain model based on the metamodel and the IaaS domain knowledge. Next, we present the methods for creating service and demand models based on the metamodel and the domain model. In addition, an equivalent concept ontology of IaaS FMs is also given to cope with the heterogeneous expressions in different FMs.

(ii) Based on the designed IaaS description model, we propose a method to realize the automatic process of discovery and selection of the best IaaS instance. The automatic analysis techniques of a FM are introduced to extract the service configuration information in the FM, check the validity of an IaaS FM, and perform the searching for the matching IaaS instance based on user needs.

(iii) The important nonfunctional metrics and their measurement methods for IaaSs are studied and determined, and the metrics weighting method based on the combination of the subjective and objective weights is proposed, which makes the weight-assignment process of the decision metrics more reasonable. The multicriteria decision-making method for the matching IaaS instances is also introduced.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 presents the formal IaaS description method based on FMs. Section 4 introduces the proposed discovery and selection method of IaaSs. Section 5 describes the automatic analysis techniques and the matching method of IaaS FMs. Section 6 presents the method of ranking IaaSs. Section 7 presents an experiment to evaluate the feasibility and effectiveness of the methods introduced in this paper. Finally, the last section concludes the paper and proposes future work.

2. Related Work

In addition to lots of studies, there are some commercial services to support the discovery and selection of IaaSs. Amazon [16] and RackSpace [17] provide exclusive IaaS configuration platforms to support comparisons between their own different service instances. CloudScreener [18], CloudHarmony [2], Cloudorado [19], and PlanForCloud [20] are able to provide a commercial service for the configuration and comparison of IaaS instances, but they can only cover the main IaaS providers and can only support comparisons of a specified QoS metric, such as fee. Research also indicates that the recommended results from the commercial services or platforms are sometimes suboptimal [21]. In academic researches, the related methods for discovering and selecting an IaaS can be divided into the following categories.

(1) Semantic-Technique-Based Description and Matching Methods for IaaSs. Dastjerdi et al. [3] used the web service

modeling ontology (WSMO) as the description model of virtual appliances and virtual units in cloud computing and presented the semantic matching method of the supply and demand on IaaSs. Zhang et al. [22] created an IaaS knowledge ontology including functional and nonfunctional concepts, attributes, and relationships among them and proposed a semantic search model for heterogeneous IaaSs. Rodríguez-García et al. [4] created semantic IaaS service vectors by implementing text processing and adding semantics to the IaaS description and then sorted the IaaSs according to the results of semantic matching on IaaS service vectors and user demands. A similar method was given in [23]. Liu et al. [24] created an IaaS function ontology and presented the semantic matching method of IaaSs and the method of selecting IaaSs based on the QoS metrics. Kang and Sim [5] created a cloud service ontology and presented the semantic matching method of IaaSs from three aspects, including the concept, the object attributes of concepts, and the data attributes of concepts. The above studies described IaaSs and performed the matching of supply and demand on IaaSs by means of semantic technologies used commonly in traditional web services. Although these approaches can deal well with the heterogeneous descriptions of IaaSs, they can only describe the static IaaS information and cannot deal with the numerical service parameters and structured service information; thus, the IaaS configuration space cannot be accurately and completely described. Under this condition, the discovery method based on the semantic matching can only be match to providers and cannot be matched to specific IaaS instances, making it difficult to accurately obtain the real optimal IaaS instance. In this paper, we use a FM to describe an IaaS in a structured, complete, and accurate manner and then use the automatic analysis techniques of a FM to search the matching instances of an IaaS so that a real optimal solution can be obtained.

(2) QoS-Evaluation-Based IaaS Selection Method. The Cloud Service Measurement Index Consortium (CSMIC) of Carnegie Mellon University [25] presented a service measurement index for evaluating and comparing cloud services in general, including the accountability, agility, cost, performance, assurance, security, privacy, and usability. Garg et al. [6] developed 15 evaluation metrics for IaaSs by selecting and refining the CSMIC index and used these metrics to evaluate the IaaS QoS by means of the AHP. However, some metric measurement methods given by them are not feasible, and the AHP can only deal with a small amount of alternative IaaS instances. Li et al. [7] divided the QoS metrics of IaaSs into three categories, performance, economy, and security, and then designed the corresponding evaluation metrics for computing services, storage services, and comprehensive services. They provided some good metric directories for evaluating the IaaS QoS, but the security measurement was too simple. In [6, 26], the authors determined the weights of QoS metrics only according to the subjective comparison and judgment, and thus it is hard to guarantee the objectivity of evaluations of the IaaS QoS. In this paper, we evaluate the IaaS security in a systematic and objective manner and present a combination weighting method, by which the weight distribution can not only be more objective, but also reflect the subjective preference of decision makers.

(3) FM-Based IaaS Description and Selection Methods. Wittern et al. [8] proposed a method of selecting an IaaS based on variability modeling. However, the authors only gave a very simple domain model and did not provide a method of coping with the heterogeneous expressions of IaaSs. García-Galán et al. [9] also proposed the use of a FM to describe an IaaS. The authors took EC2 as an example to introduce the specific methods of creating an IaaS FM. Quinton et al. [14] introduced attributes to extend a Boolean FM and then used the extended FM to describe an IaaS. The authors also defined a domain knowledge ontology to cope with the heterogeneous expressions in FMs. Finally, they realized the automatic IaaS configuration. García-Galán et al. [10] proposed a method founded on software product line to configure the Amazon IaaS. The authors described the Amazon EC2, EBS, S3, and RDS services by means of FMs and used an offline solver to find the most appropriate configuration instance. Dougherty et al. [27] also described an IaaS as a FM. They used "feature" to represent the main functional units and "attribute" to represent the energy consumption of each functional unit. Their goal is to automate the IaaS configuration in the most energy-efficient and economical way. However, in [9, 10, 14, 27], the authors did not present a standardized definition for the IaaS FM, and thus the universally applicable methods of creating IaaS service and demand models could not be achieved. Wittern and Zirpins [28] only provided a general conceptual FM adapted to various web services without considering the special requirements of IaaSs. In this paper, we define the metamodel and the domain model for IaaSs in a standardized manner and provide the general method for creating the IaaS service and demand models and then propose the automatic discovery and selection methods of IaaSs based on them.

3. Feature-Model-Based IaaS Description Method

3.1. Feature Model Basics. FMs are treelike data structures, where each node represents a product feature, and the edge between two nodes represents the relationship between the corresponding features. The root of a FM represents the domain object that is described. Features are bound by means of vertical and cross-tree relationships, which define how features can be grouped in a product. The father feature and its child features are usually constrained by mandatory, optional, or cardinality vertical relationships, and the cross-tree relations mainly include REQUIRES and EXCLUDES [13, 15]. In a FM, features can represent specific functional or nonfunctional characteristics of a demand, service, component, or platform, and so forth and can also be used as abstract features that represent domain decisions, such as Linux-based features. The mandatory relationship between the father feature and its children means that the child feature must be selected when its parent feature is selected, while the optional relationship between them means that the child

feature can be selected or not selected when its father feature is selected. In addition, cardinality is primarily used to define the range of the child features that can be selected when their parent feature is selected. A FM that only contains features and feature relationships is called a Boolean FM. A Boolean FM can be extended as an attributed FM by defining feature attributes, which can represent more complex domain requirements and product models. All functional and nonfunctional property parameters of a product can be described as feature attributes. For an attributed FM, product customization can be realized by feature selection, attribute assignment, and the definition of constraint relationships among the selected features and their attributes. A correct and reasonable FM should be able to completely describe the product configuration space, and any semantic conflicts should not exist in all defined relationships.

3.2. Feature Modeling Method for IaaSs. The goal of IaaS feature modeling is to describe IaaSs and user demands for an IaaS effectively and accurately. We first define the metamodel of an IaaS to regulate the concepts in IaaS FMs and the relations among these concepts in a standardized manner, in order to provide the grammar fundamentals for creating IaaS FMs. Then we define the domain model of an IaaS based on the metamodel and the IaaS domain knowledge. The advanced and abstract characteristics of an IaaS are classified and unified by the domain model, which provides the conceptual basis for establishing IaaS FMs. Finally, the specific methods of creating the IaaS service and demand models are presented.

3.2.1. Metamodel. First, the metamodel of an IaaS is defined using UML, as illustrated in Figure 1. A metamodel is a description model of the model, which is the semantic description of the concepts and concept relationships in the model. The defined metamodel mainly includes three kinds of concepts, features, attributes, and relationships.

The main functional and nonfunctional characteristics of an IaaS instance are defined as features, such as VM, OS, location, and cost model. Features can be divided into a root feature, group features, and solitary features according to their organizational structures. Features can also be divided into a root feature, abstract features, and leaf features according to the feature abstract level. Among these, leaf features represent the most specific functional or nonfunctional characteristics. Feature group is an abstract feature used to accommodate the group features of the same type, and it can be the child of a solitary feature or another feature group. Features in the feature group are called group features, which are used to represent a series of features with common characteristics. For example, an OS feature group may contain two group features, Windows and Linux. All the features that are not included in the feature group are called solitary features, which are used to describe the relatively simple object characteristics. A solitary feature can be used as a feature group, containing a series of group features as its children. For instance, OS is a solitary feature, which contains WINDOWS, LINUX, and other group features as its children. Solitary features can be defined as mandatory or optional type, while feature cardinality is defined to constrain the range of group features that can be selected at the same time. Abstract feature and leaf feature and group feature and solitary feature are two independent feature classifications, which means that not only can abstract feature and leaf feature be defined as group feature or solitary feature, but group feature and solitary feature can also be defined as abstract feature or leaf feature.

The second kind of main concept in the metamodel is feature relationship, which is divided into two categories. The first is the refinement relationships in the vertical direction of the feature tree; that is, features are gradually refined (including decomposition, specialization, and characterization) from the root feature down to the leaf features, and the features between the root and the leaves represent the different abstract levels of characteristics. Each IaaS instance's configuration options must be refined to leaf features. The second is the horizontal cross-tree constraint relationships, which mainly contain three kinds of relationships; namely, REQUIRES, EXCLUDES, and COMPLEX CONSTRAINTS. The cross-tree constraints are generally used to describe the constraint relationships among the features of different categories and their attributes in an IaaS instance configuration; for example, price features are commonly constrained by location features (e.g., the prices of IaaS instances in Asia and North American are likely to be different). The third kind of main concept in the metamodel is feature attribute, which is usually used to describe the common properties of IaaS functions or nonfunctional characteristics, such as the numbers of vCPU, memory, and hard disk capacity. By defining the constraint relationships among the features and their attributes, we can accurately describe the configuration options of an IaaS instance. Several important definitions in the metamodel are as follows.

Definition 1 (FEATURE GROUP CARDINALITY). Assuming that $0 \le n \le n' \le k$, where n, n', and k are nonnegative integers and k is the number of all group features, we define $\langle n - n' \rangle$ as a feature group cardinality, in which n is the lower bound and n' is the upper bound, and $\langle n - n' \rangle$ means that when the feature group is selected, at least n group features must be selected simultaneously, while at most n' group features can be selected simultaneously.

Definition 2 (REQUIRES). Assuming that features A and B are from the same FM, if feature A is selected at any time it means that feature B is selected simultaneously, and then we say that feature A REQUIRES feature B.

Definition 3 (EXCLUDES). Assuming that features A and B are from the same FM, if feature A is selected at any time it means that feature B is not selected simultaneously, and then we say that feature A EXCLUDES feature B.

Definition 4 (COMPLEX CONSTRAINTS). By means of arithmetic symbols such as +, −, ∗, /, and mod, and relationship symbols such as >, >=, <=, <, ==, and !=, and logical symbols such as AND, OR, NOT, IMPLIES, and IFF (if

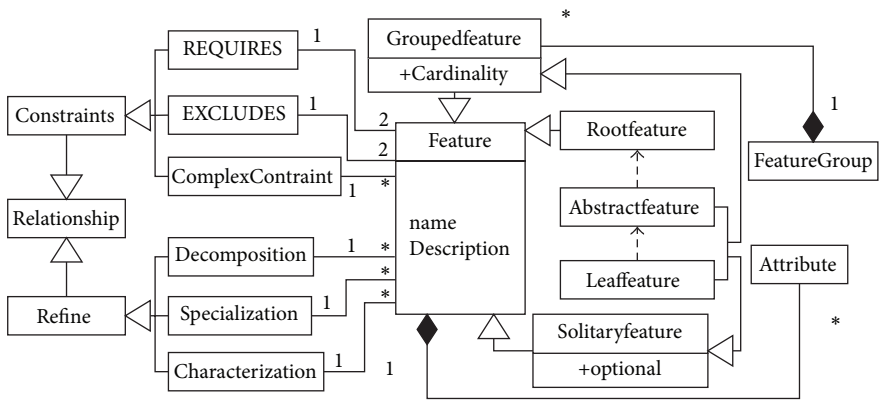

Figure 1: Metamodel of an IaaS.

only if), we can associate certain features with certain feature attributes in the FM to form a variety of logical expressions, which are used to describe a particular relationship that must be met among these features and feature attributes. We call this type of relationship among the features and feature attributes COMPLEX CONSTRAINTS.

In summary, we perform a design of the metamodel that provides the grammar supports for creating a structured IaaS FM. We give the definition of the feature, attribute, relation, and other key concepts in an IaaS FM. In addition, we introduce the roles and usage of various features and also describe the vertical refinement relations and the cross-tree constraints in an IaaS FM.

3.2.2. Domain Model. Because the current IaaS description has no uniform information structure and terminology, the information structure and terminology of IaaS FMs will appear random if they are created relying only on the syntax basis of the metamodel. Thus, it will be difficult to perform the matching of the IaaS service and demand models. In fact, there is a consistent functional and nonfunctional characteristics category existing in IaaSs. Therefore, an IaaS domain model is created based on the metamodel and the IaaS field knowledge so as to unify IaaS FMs' information structure and terminology. The domain model classifies and regulates the advanced and abstract features of IaaSs and defines all necessary attributes, thus providing a conceptual basis for the creation of IaaS FMs. The major structure of the domain model is illustrated in Figure 2. A summary of important attributes of the main IaaS features is shown as Table 1.

The definition of the domain model follows the principles below: first, those features and feature directories that are more definite and widely recognized should be included as completely as possible; as such, based on the domain model, the IaaS service and demand models can be created in a more unified and consistent structural form. Second, it is necessary to avoid too much refinement of the abstract features that may have a variety of representations, so as to meet the actual needs for describing IaaSs. Third, the important common attributes of the IaaS function characteristics should be defined in a uniform manner, because these attributes will be the main parameters that are evaluated and compared when performing the supply and demand matching and horizontal comparison of IaaSs. Fourth, the domain model should maintain good extensibility because of the constantly development and updating of IaaSs in cloud computing.

Table 1: Summary of important attributes of the main IaaS features.

Feature	Attribute	Type	Remarks
IaaS	periodcost	real	Periodic cost
	availability	real	
	securitylevel	real	Security level
	netdelay	int	Network latency
VM	HDD	string	Solid state or magnetic
	memory	real	
	SPECCPU2006	int	CPU benchmark value
	scaling latency	int	Scaling resource latency
LoadBalancer	costmonth	real	Cost per month
	num	int	Load balancer number
Network	availability	real	
	supportipv6	bool	Support for IPv6
	estimatetraffic	real	Estimate traffic per month
Storage	availability	real	
	persistence	real	Storage persistence
Freetraffic	freelimit	real	Free traffic per month
	extraprice	real	Extra traffic price

3.2.3. Service Model. As an example, Figure 3 illustrates the major structure of the created IaaS service model from DigitalOcean [29]. Based on the metamodel and the domain model defined above, the service model of an IaaS can be created to represent the actual IaaS offerings. The key points of creating an IaaS service model are as follows: first, select features from the domain model and refine the abstract features among them according to the original IaaS description. Second, because the service model has inherited all the attributes of the related features in the domain model, it only needs to select the required feature attributes to specify

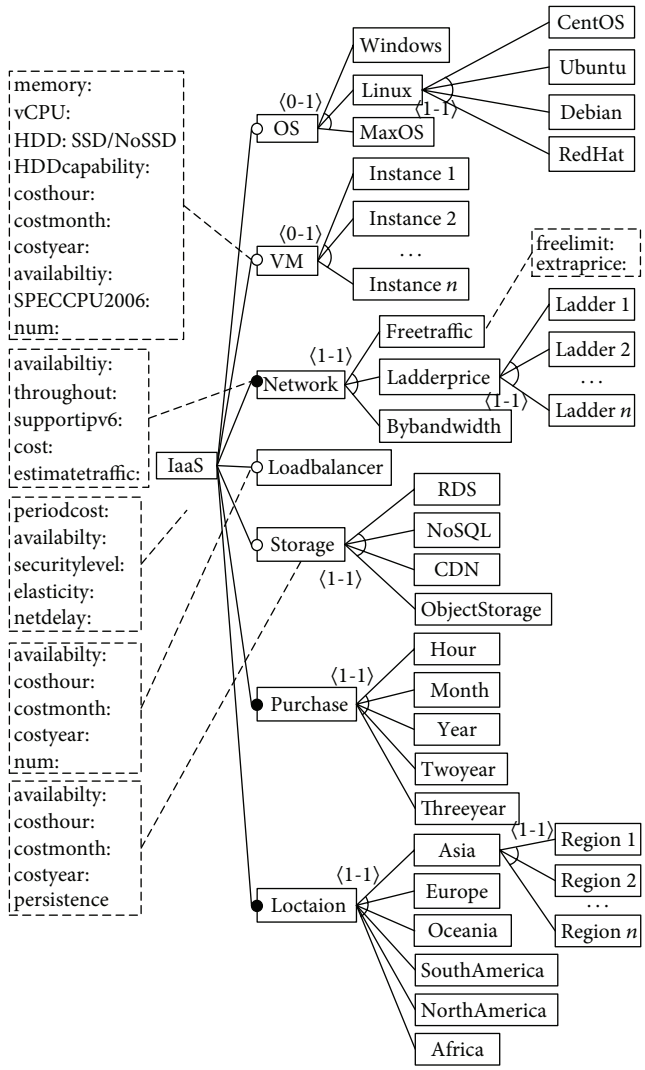

Figure 2: Major structure of the IaaS domain model.

their value ranges. Finally, the entire IaaS configuration space is described by defining many COMPLEX CONSTRAINTS and, in general, each of the COMPLEX CONSTRAINTS can represent an IaaS configuration instance. These COMPLEX CONSTRAINTS are included in the plain-text file of the IaaS service model; see Section 7.1.3 for details.

3.2.4. Demand Model. Figure 4 illustrates the major structure of a demand model for computing services. The demand model is used to describe user needs in an IaaS, which is created by using a similar method of creating the IaaS service model. Similarly, the first thing that should be done is to select features from the domain model and refine the abstract features among them according to user needs. The demand model has also inherited all the attributes of the related features in the domain model. It is also necessary to select the required feature attributes to specify their values or value ranges. The demand model represents a minimum of service requirements, meaning that the IaaS instances satisfying user needs usually number more than one. Therefore, when the IaaS supply and demand matching is completed, choosing the best IaaS instance is required. For example, when it is necessary to optimize service cost, we can select the IaaS instance satisfying user needs that has the minimum cost.

3.3. Concept Ontology of IaaS FMs. While the domain model has achieved a unified classification and specification of the advanced and abstract features in IaaS FMs, it is still necessary to refine the abstract features into the leaf features in the process of creating an IaaS FM, which may lead to the use of different terms to represent the same concept; for example, given geographical region feature, the terms "region" or "location" may be used in different IaaS FMs. In order to make all IaaS FMs mutually "understood" so that matching the IaaS supply and demand can be carried out, we have created an IaaS concept ontology to realize the mutual mapping of equivalent feature concepts; some parts of this are illustrated in Figure 5. The concepts of "VM" and "Virtualmachine," "Region" and "Location," "Windowserver_2003" and "WindowsServer2003," and "Ladderprice" and "Multilevelprice"

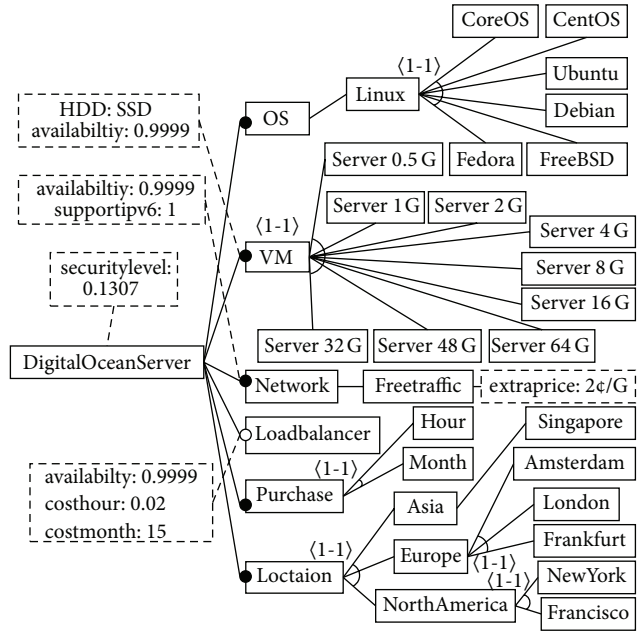

Figure 3: Major structure of the DigitalOceanServer service model.

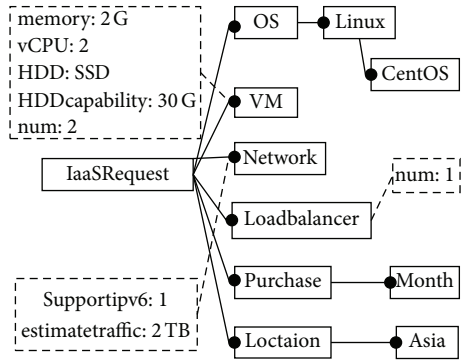

Figure 4: Major structure of an IaaS demand model.

are all equivalent. In addition, it should be noted that the domain model has unified and defined all important feature attributes considering the consistency of the IaaS feature attributes; therefore, when the IaaS FMs are created, it must be done using the unified feature attributes specified by the domain model.

3.4. Characteristics Analysis of the Description Method. First, it is not difficult to create and maintain an IaaS FM with the metamodel and domain model acting as the grammatical and concept bases. Graphical tools or customized machine languages can be developed to create and edit an IaaS FM. Second, our IaaS description method has good extensibility and compatibility and can continuously update the domain model to improve the conceptual repository according to the development of IaaSs. The concept ontology of IaaS FMs acts as the bridge between the different terms with the same meaning in different IaaS FMs. Finally, the metamodel provides strict grammatical norms for the creation of an IaaS FM, and this paper will check the validity of an IaaS FM based on the automatic analysis operations of a FM (see Section 5.1).

4. IaaS Discovery and Selection Method

Based on the formal IaaS description presented in Section 3, we propose a systematic discovery and selection method for IaaSs. An architecture of the system implementing the method is designed as Figure 6.

Four roles are managed in the architecture: User, Broker, IaaS Provider, and MTCCAs (cloud service monitoring, testing, certificating, and comparing agencies). The IaaS Provider usually publishes the IaaS service information on its portal website and sometimes chooses to perform a service certification or registration via MTCCAs. For example, the IaaS Provider may submit a security assessment report or perform a security certification in Cloud Security Alliance [30] (CSA).

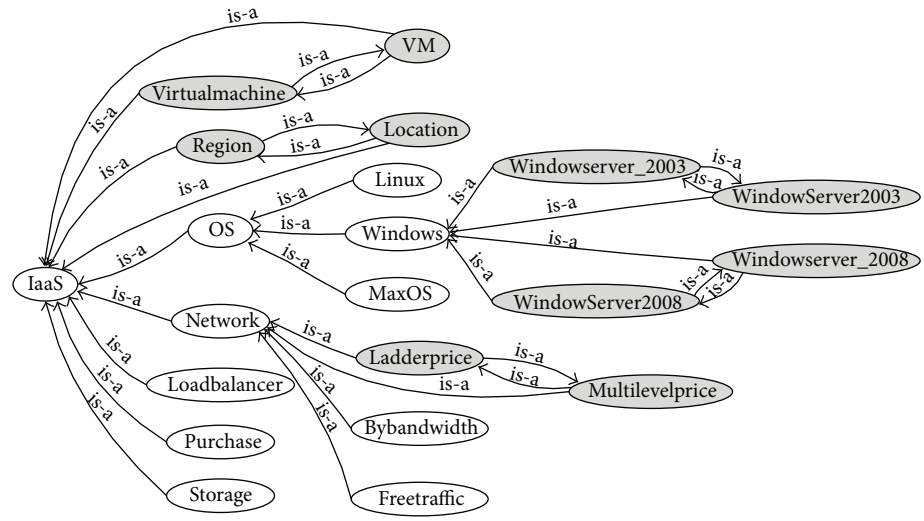

Figure 5: Concept ontology of IaaS FMs.

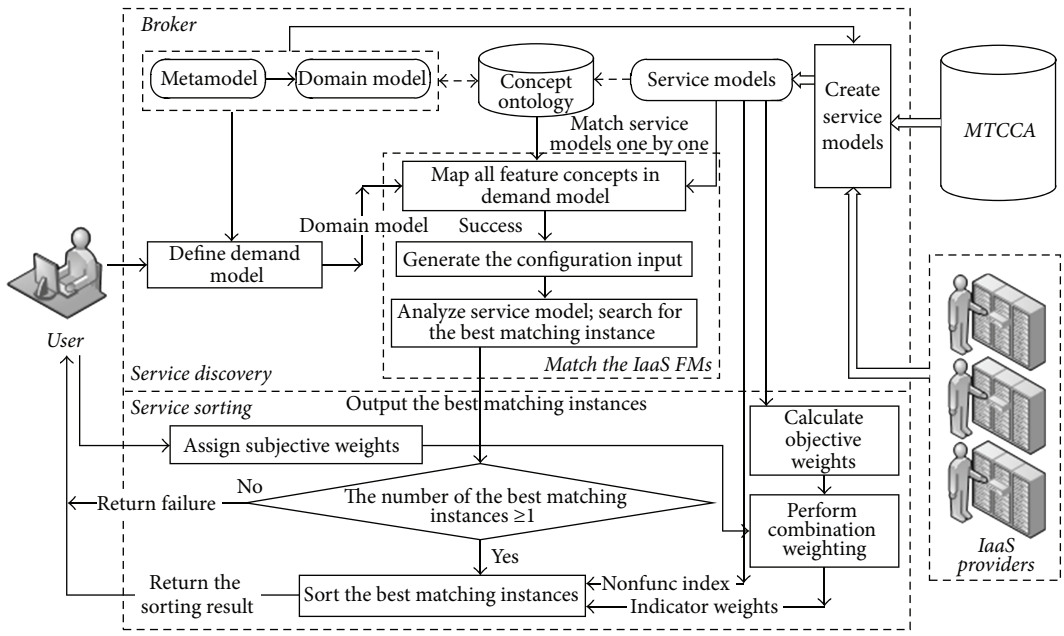

Figure 6: Architecture of the discovery and selection system of IaaSs.

MTCCAs such as CloudHarmony, CloudScreener, CSA, and Cloudorado mainly perform the monitoring, performance testing, security certification, or synthetic comparison of cloud services. The entirety of IaaS service information can be obtained by combining the IaaS information from the provider portal and the information from MTCCAs. The user needs to create and submit the IaaS demand model and perform the subjective weighting for important nonfunctional metrics. The Broker is used as the functional core of the architecture.

The Broker's main functions are divided into two parts: service discovery and service ranking. Regarding service discovery, the Broker first needs to regularly maintain the IaaS domain model based on the IaaS domain knowledge by adding new features and feature directories. Second, the Broker uses the IaaS related information from the IaaS Providers and MTCCAs to create the IaaS service models. The Broker is also responsible for the regular maintenance of the concept ontology of IaaS FMs. Finally, the Broker takes the user demand model, the concept ontology, and the IaaS service models as the inputs of the IaaS FM matching module.

The matching of supply and demand on IaaS FMs consists mainly of two steps. The first step is to map the feature concepts in the demand model into the feature concepts in the current IaaS service model. If the related concepts cannot be completely mapped successfully, the matching process will proceed to the next IaaS service model to continue. If

the mapping is successful, all the feature concepts in the demand model are converted into the feature concepts in this IaaS service FM, and then the IaaS demand configuration suitable for this service FM can be generated according to the user demand model. The second step is to search for the best matching instance by automatically analyzing the current service model. If no instances satisfying user needs are searched, the matching of supply and demand on the current IaaS service model fails. Otherwise, it needs to find the best one from the IaaS configuration instances satisfying user needs by optimizing the cost and then terminate the matching process on the current IaaS service model and proceed to the next IaaS service model to continue until the matching with all IaaS service models is completed. Finally, the IaaS FM matching module returns all the best matching instances as the output results. This service discovery method will be detailed in Section 5.

Another main function of the Broker is to rank all the best matching instances. If the number of the best matching instances returned by the IaaS FM matching module is 0, it shows that there is no IaaS instance satisfying the user demand, and then failure information will be returned. If the number is 1, then the single instance will be returned directly. When the number exceeds 1, the system will automatically start the next stage, namely, service sorting. The sorting process takes the critical nonfunctional metric set and the metric weight vectors as inputs and sorts all the best matching instances by calculating their comprehensive evaluations. Finally, the ranking list is delivered to the user. This service sorting method will be detailed in Section 6.

5. Automated Analysis and Matching of IaaS FMs

FMs unify the descriptions of the IaaS services and demands, which paves the way for automated IaaS discovery. IaaS discovery is the process of searching for the IaaS instances that satisfy the user demand from many IaaS offers through matching the IaaS supply and demand. To complete matching the supply and demand on IaaS FMs, it is necessary to analyze the IaaS service models automatically. Usually, there are many features, feature attributes, and COMPLEX CONSTRAINTS in an IaaS service model; moreover, the IaaS configuration space is also large. Therefore, it is impossible to use an artificial method to analyze the IaaS service model, and thus an automatic mechanism is needed. In fact, the automated analysis of feature model (AAFM) has become a special technique; it is a set of automated analysis operations that extract information from the FMs [31]. Each analysis operation can be regarded as a black-box process, with the FM and the specific operation parameters as inputs, and it can retrieve and output the corresponding results based on the different goals of analysis. These analysis operations mainly include checking whether the FM syntax is correct, judging whether there is a semantic conflict, and verifying whether the input product configuration information is valid.

5.1. Automated Service FM Analysis

5.1.1. Checking the Service FM Validity. Before performing other analysis operations on an IaaS service model, it is necessary to determine its validity by checking its syntactical correctness and semantic conflicts. Semantic conflict checking is necessary because some semantic conflicts are very prone to be introduced when COMPLEX CONSTRAINTS on features and attributes are created in a FM. The validation operations of an IaaS service model mainly include the following:

(1) Judging the syntactical correctness of an IaaS service model text file. In Section 3, we have described the method of creating an IaaS service model in graphics. In fact, in order to perform the automated analysis on a service model, we have to choose a language to express the service model, such as XML or a custom language [15]. We just have to define the XML schema or the syntax of the language chosen before using them. We must comply with the relevant syntactical constraints of the language used when creating the service model text file.

(2) Checking whether the service FM is empty. It should be noted that any contradictions among the relations and constraints of an IaaS service model could lead to the failure of the representation of the IaaS, and we can discover the possible contradictions by analyzing the IaaS service model.

(3) Checking whether the type definitions of all features are reasonable and whether there are semantic conflicts in the cross-tree constraints. For example, when the parent feature is a mandatory type, the only child feature is naturally a mandatory type. Otherwise, an error is generated. The cross-tree constraints are also prone to semantic conflicts; for example, if a REQUIRES constraint between a mandatory feature and another optional feature is created, a semantic conflict occurs.

(4) Checking whether the feature cardinalities are reasonably set. For example, there might be a conflict between the min value of feature group cardinality and the type of feature group. When the feature group is mandatory type, the min value of its cardinality must be no less than 1. When the feature group is of optional type, the min value of its cardinality must be 0. In addition, the maximum value of group cardinality should be no more than the number of features in the feature group. Otherwise, a semantic conflict occurs.

5.1.2. Checking the IaaS Configuration Validity. An IaaS configuration is defined as a set of options for running an IaaS instance. For a computing instance, these options correspond to the predefined configuration points, such as OS, resource location, purchase mode, CPU size, and memory size. If at least one instance satisfying the user needs can be found in the IaaS service model according to the submitted configuration,

then the submitted configuration can be considered valid; otherwise, it is invalid. For an attributed FM, when the feature selection and attribute values assignment are completed, and the relationship constraints among the selected features and their attributes are satisfied, the service instances satisfying the configuration needs are determined.

5.1.3. Searching for the Best IaaS Instance. Usually, after a user configuration demand is inputted, many IaaS instances satisfying the demand will be searched from the service model; however, only the best one will be returned. The IaaS instances from the same provider are generally very similar in terms of availability, network latency, security, and other nonfunctional properties, while the cost difference is relatively large. Therefore, we choose to optimize the matching instances based on cost; that is, to search for the one with the least cost among all instances satisfying the user needs from the same provider.

5.2. Matching between the Service and Demand Models. The supply and demand matching operations between the service and demand models include the following:

(1) Generating the configuration input suitable for the service model from the demand model. First, all of the feature concepts in the demand model are mapped into the feature concepts in the service model based on the IaaS concept ontology such that the demand model becomes completely consistent with the service model conceptually. Second, the configuration input suitable for the service model will be generated by the mapped demand model. The configuration input represents the user needs, which mainly contain three aspects of information: features, attributes, and constraints.

(2) Searching for the configuration instance with optimal cost by analyzing the IaaS service model. This step will use the IaaS demand configuration generated in step (1) as an input and perform the automated analysis on each service model to search for the best configuration instance based on the method outlined in Section 5.1.3. If the analysis results show that the input configuration is invalid, then the matching of supply and demand on IaaS FMs fails. If the results show the input configuration is valid and many configuration instances satisfying the user needs are found, then the minimum-cost instance will be returned.

6. IaaS Ranking

According to our IaaS discovery and selection method, a series of best matching instances will be returned after completing the matching between the demand model and all service models (otherwise, service discovery fails). These best matching instances are the minimum-cost instances from different providers that can meet the users' functional demands. Users will choose the most satisfactory one from these instances. The usual approach is to evaluate and compare, synthetically, the important nonfunctional metrics affecting the IaaS QoS, and then make a choice. It is a multicriteria decision-making problem.

6.1. Important Nonfunctional Metrics of IaaSs. The establishment of a unified and effective QoS-evaluating metric system is the premise of scientific decision-making. However, the existing measurement metric systems for IaaSs are either inadequate [32, 33] or short of operational measurement methods [6, 25]. The important nonfunctional metrics affecting the IaaS QoS fall into three categories: performance, economy, and security [7]. We study and determine the main metrics affecting the IaaS QoS and their measurement methods.

(1) Cost. The economy of cloud computing is its prominent advantage, and it is also one of the factors the user is most concerned about. The cost of an IaaS instance depends on its resource allocation. If the configuration parameters of the IaaS instance are uncertain, then its cost cannot be obtained. If the service and demand models match successfully, the cost optimization of the matching instances will be performed. Thus the best instance and its cost can be obtained.

(2) Availability. This is a parameter usually promised by most providers in their service level agreement (SLA). In a certain period, the availability of the IaaS functional unit can be generally calculated in the following ways:

$$\begin{aligned}&\text{virtual machine availability} = \frac{\text{uptime}}{(\text{uptime} + \text{downtime})},\\ &\text{storage availability}\\ &= \frac{\text{number of successful requests}}{(\text{number of successful requests} + \text{number of failed requests})}.\end{aligned} \tag{1}$$

Availability can be aggregated in the following ways:

$$\begin{aligned}&\text{computing type IaaS availability}\\ &= \begin{cases}\text{virtual machine availability} \times \text{network availability} & (\text{where virtual machine number} = 1)\\ \text{load balancer availability} \times \left(\sum_{i=1}^{n}\left(\left(\frac{1}{n}\right) \times \text{virtual machine availability}\right)\right) \times \text{network availability} & (\text{where virtual machine number} > 1).\end{cases}\end{aligned} \tag{2}$$

(3) VM Performance Metric. The VM is the most important computing functional unit in cloud computing, and its performance has a direct impact on the quality of the computing service. Different IaaSs having the same VM

configuration usually show a significant difference in performance testing results. The Standard Performance Evaluation Corporation [34] (SPEC) is a nonprofit organization consisting of computer manufacturers, system integrators, universities, research institutions, and so forth. SPEC aims at establishing and maintaining the standards for evaluating computer systems. CPU2006SPEC is a CPU system testing suite introduced by SPEC, which is the general standard of CPU testing at present, including the SPECint and SPECfp subtesting modules. SPECint tests CPU integer processing performance, while SPECfp tests CPU float point processing performance. In this paper, we use the sum of the SPECint and SPECfp test scores as the final score of the VM performance test.

(4) Data-Storage Persistence. This is used to evaluate the possibility that the stored data is lost or damaged, which is represented by calculating the percentage of the lost or damaged data in a certain period: intact disk number/(intact disk number + lost or damaged disk number).

(5) Elasticity. The load of cloud computing applications is usually dynamic, such as e-commerce applications, which requires the IaaS resources to have prominent elasticity. Perfect elasticity enables users to adjust resources quickly according to load changes, so as to better meet the requirements of applications and avoid the purchase of redundant resources to save costs. Time is a crucial aspect in elasticity, which can be measured by the speed of response to the dynamic workload [33]. Considering that the differences in the resource release speeds of computing instances are relatively small, we have learned from the method proposed in [32], which tests VM scaling latency to measure resource elasticity. Scaling latency includes VM provisioning latency and system booting latency.

(6) Network Latency. This means the network delay from an IaaS resource to a target user area, which also has an important impact on the IaaS QoS. If there is an average of 20 s of network latency, the target users will face unendurable service response delay when they access the applications deployed on the IaaS resources.

(7) Security. Cloud computing is facing many security threats, such as data destruction, network attacks, privacy leaks, and VM attacks, so security is also a very important metric. In [35], we proposed a systematic method for measuring and evaluating cloud security (referred to as MMECS in this paper). MMECS measures and evaluates cloud security by calculating the security advantage of each cloud provider based on the Consensus Assessment Initiative Questionnaire (CAIQ) reports on cloud security, which have been submitted to the CSA by the cloud providers. CAIQ is a cloud security assessment questionnaire based on the Cloud Control Matrix (CCM), while the CCM is proposed by the CSA and defines a list of security control requirements that the cloud providers and their services should fulfill. The CCM references authority security standards and frameworks in the industry, such as SP800-53 R3, FedRAMP, ISO/IEC 27001, COBIT, and ENISA IAF. First, the evidences of security compliance provided by the CAIQ report are scored, and then the obtained score is used to measure the credibility of the CAIQ report. Second, we use the credibility of the CAIQ report to correct the original score of each security control domain in the CAIQ report. Finally, the security advantage of each IaaS Provider is calculated based on the corrected scores of control domains. The results are relatively objective and convincing, so we use the MMECS method to calculate the security metric of each IaaS Provider.

(8) Reputation. This is an important attribute of an IT service, and it is also a metric usually compared when users choose services, such as e-commerce services. Although many cloud service reputation calculation models have been proposed in the academic community, they are still in the theoretical research stage and not yet commercially available. Therefore, it is very difficult to obtain available reputations for IaaSs. Although the market share of an IaaS cannot substitute for its reputation, it can basically reflect the user's choice tendency and the service credibility to a certain extent, so we use the market-share data provided by CloudHarmony as a measure of the IaaS reputation.

6.2. Combination Weighting Method for Decision Metrics. The scientific and rational allocation of metric weights is of great importance to multicriteria decision-making problems. According to the data sources for calculating weights, the metric weighting methods fall mainly into two categories: the subjective and objective weighting methods. The subjective weighting method assigns metric weights based on a decision maker's subjective preference, such as the analytic hierarchy process (AHP) and Delphi. The objective weighting method calculates the metric weights based on their actual values, such as the maximizing deviations method, the entropy method, and principal component analysis [36]. The subjective weighting method reflects the decision maker's personal preference; however, this method is easily influenced by the knowledge and experiences of the decision maker. For the objective weighting method, there is an objective standard for the weight distribution, but it ignores the decision maker's subjective preferences; metric weights that are calculated totally based on the original metric data may deviate markedly from the subjective desires of the decision maker.

For acquiring more reasonable and scientific evaluation results, some researchers combine the two methods together to carry out the combination weighting [36–39]. The existing research mainly uses two representative ways of combining the subjective and objective weights: the multiplicative synthesis and weighted linear combination methods [36–39]. The multiplicative synthesis method generally uses the formula $q_j = a_j w_j / \sum_{j=1}^{n} a_j w_j$ (where a_j, w_j, and q_j are the subjective weight, objective weight, and combination weight of the jth metric, resp., and n is the number of metrics). However, this approach tends to make the large results even larger and the small ones even smaller, and sometimes it may lead to a very unreasonable result. For example, given

the subjective weights (0.6, 0.3, 0.1) and the objective weights (0.7, 0.2, 0.1), using the multiplicative synthesis method we can see from the combination result (0.857, 0.122, 0.02) that the first metric is obviously enlarged, while the third metric is decreased to the point where it can be ignored, which is obviously unreasonable. Research suggests that the multiplicative synthesis method is more suitable for the case in which the number of metrics is large and the weight distribution is more uniform [36, 37]. Regarding the weighted linear combination method, the general formula is $q_j = \delta a_j + (1 - \delta)w_j$ (where a_j, w_j, and q_j are the subjective weight, objective weight, and combination weight of the jth metric, resp., and $0 \leq \delta \leq 1$). This method overcomes the shortcomings of the multiplicative synthesis method and thus is more widely used [36–39]. The key of this method is how to determine the combination coefficient of the two weighting methods. Based on the above analysis, we plan to use the weighted linear combination method to combine the results of subjective and objective weighting. The methods proposed in [40] are used to do the subjective weighting, while the entropy method is adopted to do the objective weighting.

6.2.1. Improved AHP Weighting Method Based on Sorting. The AHP is originally conceived as a multicriteria decision-making method and is widely used as a subjective weighting method for decision metrics. The top-down hierarchical structure is used to model the relationship among decision elements in the AHP. The top node of the hierarchical structure represents the decision goal, such as QoS, and the following layer nodes are the related metrics affecting the decision goal, such as security, availability, and cost, which can be further divided. The lowest layer nodes are the alternatives. The AHP has a unique method for determining metric weights, which defines 1–9 scales to measure the relative importance degree of two metrics by combining the qualitative and quantitative ways to measure them. The relative importance degree is defined as the ratio between the weights corresponding to two metrics related to the same goal. The relative importance degree of any two metrics is determined by paired comparisons; after that, the judgment matrix is obtained. Next, to check the consistency of the judgment matrix is unavoidable. For example, if the relative importance degree of A and B equals 2, the relative importance degree of B and C equals 2, and the relative importance degree of A and C equals 4, and then the weight distribution of A, B, and C is consistent; otherwise, it is inconsistent. When the weight distribution of metrics is not consistent, the relative importance degrees must be adjusted continuously until the judgment matrix becomes consistent. Finally, the weights of all metrics will be obtained by solving the judgment matrix. How to adjust the inconsistent judgment matrix is a difficult problem in the AHP. We have improved the AHP weighting method by referencing the method proposed by Li and Wang [40], so that the obtained judgment matrix fully meets the requirement of consistency, as follows.

(1) Definition of the Relative Importance Degree Scale. The scale 1.5^{β} ($\beta = 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4$) is defined to depict the relative importance degree between metrics. The relative importance degree is 1.5^0 meaning that the two metrics are equally important. The value of 1.5 means that the former metric is slightly more important than the latter. The value of 1.5^2 means that the former metric is more important than the latter metric. The value of 1.5^3 means that the former is obviously more important than the latter. The value of 1.5^4 means that the former is absolutely more important than the latter. When β take values of 0.5, 1.5, 2.5, and 3.5, it indicates the intermediate state of the two adjacent relative importance degrees, respectively.

Table 2: Distribution of metric relative importance degrees for the cases of β and n depicted.

β	0	0.5	1	1.5	2	2.5	3	3.5	4
$n = 1$	a_3	—	a_2	—	a_1	—	—	—	a_4
$n = 2$	—	—	a_5	—	—	—	—	—	—

(2) Determination of the Subjective Weights. There are three steps in calculating the subjective weights: first, the metrics are divided into several groups according to importance from low to high. Each group represents a level of importance, and the weights of metrics in the same group are all equal or have only small differences, while the weights of metrics in the different groups have significant differences. The main purpose of grouping metrics is to determine the rationality of the results of objective weighting when the combination weighting is carried out in the future. Second, the metrics in each group are sorted according to their importance ranging from low to high, while the metrics with the same importance are sorted by their numbers. Finally, the subjective weights of decision metrics are determined according to the following method. The left-hand first one of the sorted metric sequence is taken as the main reference and the left-hand adjacent one of the current metric is taken as the auxiliary reference; then, the relative importance degree between the current metric and the main reference metric and the relative importance degree between the current metric and the auxiliary reference metric are determined. According to this method, the relative importance degree of each metric in the sequence is assigned, in turn, from the left-hand to right-hand order. Finally, the subjective weight of each metric can be obtained based on all the relative importance degree values. Table 2 shows the case of the distribution of relative importance degrees, in which β represents the relative importance degree index for each metric, and the maximum n in each column represents the number of metrics with the same weight. The determination process of the relative importance degree is as follows.

First, the metric with the smallest degree of importance, namely, the left-hand first one in the sorted metric sequence, is placed in the cell where $\beta = 0$ and $n = 1$. In turn, other metrics in the sequence are compared with the left-hand first metric and its left-hand adjacent metric, and then it is placed into the corresponding cell based on the comparison results. For example, for the metric sequence a_3, (a_2, a_5), a_1, and a_4, the process in detail is as follows: At first, a_3 is placed into the cell where $\beta = 0$ and $n = 1$. Meanwhile, a_2 is placed

into the cell where $\beta = 1$ and $n = 1$ because a_2 is slightly more important than a_3. Considering that a_5 is slightly more important than a_3 and is as equally important as a_2, a_5 is placed into the cell where $\beta = 1$ and $n = 2$. In addition, a_1 is placed into the cell where $\beta = 2$ and $n = 1$ because a_1 is more important than a_3 and is slightly more important than a_5. Finally, a_4 is placed into the cell where $\beta = 4$ and $n = 1$ because a_4 is absolutely more important than a_3 and is more important than a_1.

It has been proved that the judgment matrixes determined by the above method are consistent [40]. The subjective weight of each metric can be calculated according to the formula $w_i = 1.5^{\beta_i} / \sum_{j=1}^{n} 1.5^{\beta_j}$, where β_i and β_j are the relative importance degree indexes of metrics i and j, respectively.

6.2.2. Calculation of the Objective Weights. The entropy method is a commonly used objective weighting method that obtains the objective weight of each metric by calculating its entropy based on the raw metric data. Entropy is a measurement of system uncertainty; the higher the entropy value of a system, the greater its uncertainty, and the less its information, and vice versa [41]. We assume that the multicriteria decision-making problem has m alternatives and n decision metrics, and any metric can be expressed as x_{ij} $(1 \le i \le m, 1 \le j \le n)$. The objective weights of decision metrics are then calculated according to the following steps:

The first step is to achieve the normalized decision matrix $\mathbf{P} = (p_{ij})_{m\times n}$ by normalizing the decision matrix $\mathbf{X} = (x_{ij})_{m\times n}$, so that each decision metric is the profit type and its value is between 0 and 1.

The second step is to calculate the entropy of each metric. The uncertainty of the importance degree of any decision metric can be measured by calculating its entropy according to the following formula:

$$E_j = -\sum_{i=1}^{m} p_{ij} \ln p_{ij} \quad (j = 1, 2, \ldots, n). \tag{3}$$

From the above formula, when a certain metric in all alternatives tends to be equal, the entropy of the very metric approaches its maximum value, and thus the uncertainty of the importance degree of the metric tends to be greater. The maximum entropy $E_{\max} = \ln m$, so E_j can be normalized by $E_{\max}$ as follows:

$$e_j = -(\ln m)^{-1} \sum_{i=1}^{m} p_{ij} \ln p_{ij} \quad (j = 1, 2, \ldots, n). \tag{4}$$

The third step is to calculate the difference coefficient of each decision metric by the following formula:

$$d_j = 1 - e_j \quad (j = 1, 2, \ldots, n). \tag{5}$$

The smaller the metric's entropy, the greater the metric's difference, and the greater the role of the very metric in evaluating services and vice versa.

The fourth step is to calculate the objective weight of each metric as follows:

$$w_j = \frac{d_j}{\sum_{j=1}^{n} d_j}. \tag{6}$$

6.2.3. Combination Weighting Method Based on Comprehensive Analysis. Although the subjective weight is subjective and less accurate, it reflects the preference of the decision maker and has a superior rationality in sorting decision metrics by their importance. Therefore, the combination weighting should be mainly based on the subjective weights, and the combination coefficient of the subjective and objective weight vectors can be determined by analyzing the difference between the metric weight rankings obtained by the two weighting methods. Assuming that the subjective and objective weight vectors are identified as $\mathbf{w}_s$ and $\mathbf{w}_o$, respectively, then any linear combinations of them can be expressed as

$$\mathbf{w} = a\mathbf{w}_s + (1 - a)\,\mathbf{w}_o, \quad 0 \le a \le 1. \tag{7}$$

Assuming that the decision metrics are identified as $P_1, P_2, \ldots, P_n$, after subjective weighting, the decision metrics are divided into k groups by importance, that is, $R_1, R_2, \ldots, R_k$, in which the group R_2 is more important than the group R_1 and the group R_3 is more important than the group R_2 and so on. The method of determining the coefficient a is as follows:

(1) If there is a relatively uniform distribution and no essential difference in the weights of all decision metrics, then take $a = 0$, and the result of the objective weighting can be directly used as the result of combination weighting.

(2) If the metric weight rankings obtained by the subjective and objective weighting methods are exactly the same, in order to eliminate the influence of subjective factors, it should take only the objective weighting method to obtain the weights, which indicates that $a = 0$.

(3) If the metric weight rankings obtained by the subjective and objective weighting methods are not exactly the same, while the metric group rankings obtained by the two methods are the same, then take $a = 0.5$. For example, the metric weight ranking obtained by the subjective weighting method is P_1, (P_2, P_3) and (P_4, P_5) while the weight ranking of metrics obtained by the objective weighting method is P_1, (P_3, P_2), and (P_5, P_4). Obviously, the two orders of metric weights are not the same, but the orders of metric groups are the same, and in such circumstances, $a = 0.5$.

(4) If the weight rankings of metrics obtained by the two weighting methods are different and the orders of metric groups obtained by the two methods are also different, then take $a = 1$. This situation indicates that the results obtained by the objective weighting method deviate markedly from the subjective preferences of the decision maker, so the obtained objective weights have no reference value. The subjective weighting results will be directly used as the final metric weights.

6.3. Ranking Alternative Instances Based on Comprehensive Evaluations. For an alternative IaaS instance I_i, it is assumed

that I_i has k important nonfunctional metrics that are expressed as $x_{i1}, x_{i2}, \ldots, x_{ik}$, and the metric weight vector obtained by the combination weighting method is $\mathbf{w} = (w_1, w_2, \ldots, w_k)$; thus, the comprehensive evaluation U_i of I_i can be obtained by formula (8). The comprehensive evaluations of all the best instances returned by the matching of supply and demand on IaaS FMs can be calculated in this way. Finally, these instances are sorted based on their comprehensive evaluation. We have

$$U_i = x_{i1}w_1 + x_{i2}w_2 + \cdots + x_{ik}w_k. \quad (8)$$

7. Experiment and Analysis

7.1. Experimental Scenario, Data Sources, and Experimental Tools

7.1.1. Experimental Scenario. This experiment evaluates the feasibility and effectiveness of the IaaS discovery and selection method presented in this paper. Considering that a computing service acts as the main cloud infrastructure service having a wide range of applications, we specially formulate the following experimental requirements.

A user plans to buy a computing service from an IaaS Provider, and the customers are mainly located in southeastern China. The specific requirements of the computing service include the following: two virtual servers with identical configuration, a load balancer, two virtual CPUs per virtual server, 2 G memory, a solid-state hard disk, 30 G hard disk capacity, CentOS operating system, and support for IPv6; the estimated network traffic per month is 2 TB, and all service resources are located in Asia.

Three moderate-size and well-known IaaS providers, which are very representative and can provide the computing services in the Asia area, are selected as the IaaS providers in the experiment. The three providers are DigitalOcean [29], RackSpace [42], and Linode [43], which are all from the United States and have built their own cloud infrastructure centers in Europe, America, and Asia. Among them, DigitalOcean was founded in 2012, and currently has more than 15000 servers; RackSpace was founded in 1998 and has hundreds of thousands of servers; Linode was founded in 2003 and has tens of thousands of servers.

7.1.2. Data Sources. The service-related data of the three providers mainly derives from their portals. In addition, we have selected the following critical nonfunctional metrics for comparison according to Section 6.1. These metrics have been calculated or obtained in the following ways.

Cost. We obtain the best matching instances from the three IaaSs and their costs by matching the IaaS service and demand models.

Security. According to our measurement and evaluation method of cloud security MMECS, we first download the providers' CAIQ reports from the CSA site and then calculate the security advantage of each provider based on these reports.

SPECCPU2006. CloudHarmony carries out periodic performance testing on the service instances of several major cloud providers in the industry and their testing results are relatively objective and fair. We directly download the SPECCPU2006 results of related instances from the CloudHarmony site.

Availability. We obtain the availability of each computing service instance by formula (2).

Network Latency. Under an 8 M bandwidth network environment in Nanjing, by means of network testing services provided by CloudHarmony, we test the network latency of the best matching instance from each provider ten times. We remove the minimum and maximum values and then use the average value of the remaining eight tests as the final result.

Market Share. We directly download the market-share data from the CloudHarmony site.

Scaling Latency. We measure scaling latency by recording the time from when the instance is requested to when it is available to be used. We test the scaling latency for each instance ten times and remove the minimum and maximum values and then use the average value of the remaining eight tests as the final result.

7.1.3. Editing and Analysis Tools of IaaS FMs. We have selected the FaMa [44] framework to create and analyze the IaaS FM. The FaMa framework is an open-source automated analysis tool for FMs with four different reasoners, which support more than 20 different analysis operations. FaMa supports creating a FM containing attributes and feature group cardinality. FaMa is implemented as an Eclipse plug-in, which supports the following analysis operations: verifying a FM, product number analysis, verifying configuration, product searching, and so forth. FaMa can accept two forms of FM file formats, namely, XML and plain text. The plain-text format is the simplest and clearest among all the formats, so we select it to express the IaaS FM. The syntax of the plain-text format is defined as follows.

As illustrated in Figure 7, a FM is expressed as three sections: the "%Relationships" section, the "%Attributes" section, and the "%Constraints" section. The hierarchical features and the vertical relationships among features are defined in the "%Relationships" section. The features after the colon are the children of the feature before the colon. By default, features are of mandatory type, while optional features are presented in the form of [feature name]. The feature group is represented by cardinality and a set of features in the form: [min, max]{FeatA, FeatB, ...}. The feature attributes are defined in the "%Attributes" section; the definition of each attribute contains attribute type, range, default value, and null value. The attributes can be an enumeration type and defined in the form "FeatName.AttName:[val1, val2, ..., val$N$], defaultVal,

```
%Relationships
LinodeServer: Virtualmachine OS Network [Loadbalancer] Purchase Location;
Virtualmachine: [1,1]{Linode1G, Linode2G, Linode4G, Linode8G, Linode16G,
Linode32G, Linode48G, Linode64G}
OS: Linux;
Linux: [1,1]{Arch, Debian, centos, Ubuntu, OpenSUSE, gentoo, Slackware,
Fedora};
...
%Attributes
LinodeServer.periodcost:real[0 to 10000000],0,0; % Period cost ($)
LinodeServer.availability:real[0 to 1],0,0;
LinodeServer.Securitylevel:real[0 to 1],0.1396,0;
Virtualmachine.HDD: [SSD, NoSSD],SSD,0;
...
%Constraints
Linode1GB AND Month AND (Network.estimatetraffic > Freetraffic.freelimit)
IMPLIES
(Virtualmachine.RAM == 1 GB) AND (Virtualmachine.vCPU == 1) AND (Virtualmachine.HDDcapabiltiy == 24 GB
AND (Freetraffic.freelimit == 204 8G) AND
(Virtualmachine.costMONTh == 10$) AND
(LinodeServer.cost.month = VM.costmonth + Freetraffic.extraprice × (Network.estimatetraffic − Freetraffic.freelimit)); ...
```

Figure 7: Linode service model in plain-text format.

nullVal"; the horizontal relationships among features and the COMPLEX CONSTRAINTS among multiple features and attributes are defined in the "%Constraints" section.

7.2. Creation and Matching of the Service and Demand Models. According to the requirement description in this experiment and the collected related information about the three IaaS services, we have created the demand model as illustrated in Figure 4, the DigitalOcean service model as illustrated in Figure 3, the Linode service model as illustrated in Figure 8, and the plain-text format of the Linode service model as illustrated in Figure 7. Owing to space limitations, the RackSpace service model is omitted.

We take Linode as an example to demonstrate the concept mapping between the service and demand models, along with the supply and demand matching.

First, according to the concept ontology of IaaS FMs, the concept "VM" in the demand model is mapped into the concept "Virtualmachine" in the service model, "Location" is mapped into "Region," "CentOS" is mapped into "centos," and so forth. Therefore, the input configuration for the Linode service model can be easily generated by the demand model, including features such as Virtualmachine, CentOS, Network, Loadbalancer, Month, and Asia, as well as attributes such as Virtualmachine.memory = 2 G, Virtualmachine.vCPU = 2. Second, the plain-text format of the Linode service model and the input configuration are submitted to the FaMa platform; the analysis operation code is presented in Figure 9. FaMa first verify the validity of the Linode service model and the input configuration, and then the best matching instance is obtained as follows by searching the Linode service model:

Linode2GB AND LoadBalancer AND Month AND Singapore AND (Network.estimatetraffic <= Freetraffic. Freelimit × VM.num) IMPLIES (Virtualmachine.RAM == 2 GB) AND (Virtualmachine.vCPU == 2) AND (Virtualmachine.HDDcapabiltiy == 48 GB) AND (Freetraffic.freelimit == 3072 G) AND (Virtualmachine.costmonth == $20) AND (LinodeServer.periodcost = Virtualmachine.costmonth × Virtualmachine.num + LoadBalancer.monthcost × LoadBalancer.num);

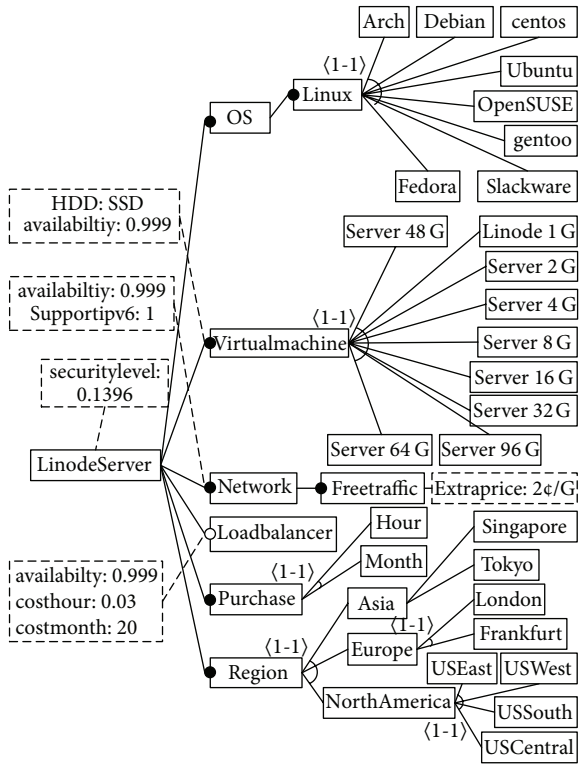

Figure 8: Major structure of the Linode service model.

LinodeServer.periodcost = 20 × 2 + 20 = $60.

We have verified the validity of the DigitalOcean service model, the RackSpace service model, and their input configurations using similar methods and obtained the best matching

```
import es.us.isa.FAMA.Reasoner.QuestionTrader; . . .
import IaaSCustomOperation.ProductSetOpt; //Matching instances are optimized in it
Public class AttributedFMoperation {
public static void main(String[] args) {
QuestionTrader qt = new QuestionTrader ();
GenericAttributedFeatureModel afm = (GenericAttributedFeatureModel)
qt.openFile ("IaaSFM/Linode.afm");
qt.setVariabilityModel (afm);
ValidQuestion vq= (ValidQuestion) qt.createQuestion ("Valid"); qt.ask (vq);
System.out.println ("The validity of the model is" + vq.isValid());
if (vq.isValid())
{ Configuration p = new Configuration();
GeneriAttributedFeature f1 = afm.searchFeatureByName ("VM"); . . .
f1.addAttribute(f1.searchAttributeByName("memory"), "2");       . . .
p.addElement (f1,1);                                                    . . .
ValidConfigurationQuestion vcq= (ValidConfigurationQuestion)
qt.createQuestion ("ValidConfiguration");
vcq.setConfiguration (p);   qt.ask (vcq);
System.out.println ("The validity of the configurationis" + vcq.isValid());
if (vcq.isValid()) { qt.addStagedConfiguration (p);
Questionq = qt.createQuestion ("Products"); qt.ask(q);
ProductsQuestion pq= (ProductQuestion) q;
if (pq.getNumberOfProcucts() == 0) system.out.println ("\n No valid product");
else
   { Iterator<? extends GenericProduct> it = pq.getAllProducts().iterator();
      if (it.hasNext()) { ProductSetOpt po = new ProductSetOpt(it);
        po.findBestProductAndprint (); }
   }
} } } }
```

Figure 9: Source code of verifications of the Linode service model and the input configuration and searching for the best instance.

Table 3: Main nonfunctional metrics of the best matching instances.

Metric name	RackSpace	DigitalOcean	Linode
a_1: cost	107.44	55	60
a_2: security index	0.1478	0.1307	0.1396
a_3: SPECCPU2006	111	101	118
a_4: availability	0.9989	0.9980	0.9998
a_5: scaling latency	123	55	118
a_6: network latency	109 ms	398 ms	260 ms
a_7: market share	5%	1%	2%

instances. The costs of the best instances from DigitalOcean and RackSpace are \$55 and \$107.44, respectively.

7.3. Ranking of the Best Matching Instances. The steps in ranking the best matching instances are as follows.

(1) Measurement of the Main Nonfunctional Metrics for the Best Matching Instances. According to Section 7.1.2, the main nonfunctional metrics are obtained as shown in Table 3.

Accordingly, the following decision matrix $\mathbf{X}$ is created:

$$\mathbf{X} = \begin{bmatrix} 107.44 & 0.1478 & 111 & 0.9989 & 123 & 109 & 0.05 \\ 55 & 0.1307 & 101 & 0.9980 & 55 & 398 & 0.01 \\ 60 & 0.1396 & 118 & 0.9998 & 118 & 260 & 0.02 \end{bmatrix}. \quad (9)$$

In order to facilitate comparison of the above metrics of these instances, we utilize the method given in [45] to standardize all evaluation metrics through the formula "$x_{ij} = (\max x_{ij} - x_{ij}) + \min x_{ij}$, $i = 1 \rightarrow m$." All evaluation metrics are converted into profit metrics by the formula "$\max x_{ij} - x_{ij}$", and then a coordinate translation is carried out by "$+\min x_{ij}$," so that the values of these metrics can be restored to their original size level. Second, we normalize the converted matrix by the formula $y_{ij} = x_{ij} / \sum_{i=1}^{m} x_{ij}$, $1 \le i \le m$, $1 \le j \le n$, where m is the number of alternative service instances and n is the number of evaluation metrics. Finally, the standardized decision matrix $\mathbf{R}$ is obtained:

$$\mathbf{R} = \begin{bmatrix} 0.208 & 0.353 & 0.336 & 0.333 & 0.231 & 0.528 & 0.625 \\ 0.405 & 0.313 & 0.306 & 0.333 & 0.517 & 0.145 & 0.125 \\ 0.387 & 0.334 & 0.358 & 0.334 & 0.252 & 0.327 & 0.250 \end{bmatrix}. \quad (10)$$

(2) Determination of the Subjective Weights. According to Section 6.2.1, we first divide all evaluation metrics into four groups and sort them by ascending importance. The metric sequence a_5, (a_4, a_6), (a_2, a_7), and (a_1, a_3) is obtained. The importance of each metric in the sequence is greater than or equal to its left-hand closest metric. Next, the metric weight distribution is carried out, and the results are shown in Table 4.

According to the formula $w_i = 1.5^{\beta_i} / \sum_{j=1}^{7} 1.5^{\beta_j}$, $\sum_{j=1}^{7} 1.5^{\beta_j} = 15.25$, the subjective weight vector $\mathbf{w}_s$ is calculated:

$$\mathbf{w}_s = (0.221, 0.148, 0.221, 0.098, 0.066, 0.098, 0.148)^T. \quad (11)$$

Table 4: Distribution of the metric relative importance degree.

β	0	0.5	1	1.5	2	2.5	3	3.5	4
$n = 1$	a_5	—	a_4	—	a_2	—	a_1	—	—
$n = 2$	—	—	a_6	—	a_7	—	a_3	—	—

(3) Calculation of the Objective Weights. According to Section 6.2.2, the entropy of an evaluation metric is obtained by formula (4), and the results are as follows:

$$\begin{aligned} e_1 &= 0.96494, \\ e_2 &= 0.9989, \\ e_3 &= 0.99813, \\ e_4 &= 0.99999, \\ e_5 &= 0.93473, \\ e_6 &= 0.89452, \\ e_7 &= 0.81945. \end{aligned} \tag{12}$$

The metric variation coefficients are obtained by the formula $d_j = 1 - e_j$, and the results are as follows:

$$\begin{aligned} d_1 &= 0.03506, \\ d_2 &= 0.0011, \\ d_3 &= 0.00187, \\ d_4 &= 0.00001, \\ d_5 &= 0.06527, \\ d_6 &= 0.10548, \\ d_7 &= 0.18055. \end{aligned} \tag{13}$$

The objective weight vector is obtained as follows by the formula $w_j = d_j / \sum_{j=1}^{n} d_j$:

$$\begin{aligned} &\mathbf{w}_o \\ &= (0.090, 0.003, 0.005, 0.001, 0.168, 0.270, 0.463)^T. \end{aligned} \tag{14}$$

(4) Combination Weighting. After the completion of the subjective weighting, metrics are sorted by ascending importance as follows:

$$(a_5), (a_4, a_6), (a_2, a_7), (a_1, a_3). \tag{15}$$

From the results of the subjective weighting, a_4 and a_6, a_2 and a_7, and a_1 and a_3 are equally important; therefore, there are some equivalent results of sorting metrics:

$$\begin{aligned} &(a_5), (a_4, a_6), (a_2, a_7), (a_3, a_1); \\ &(a_5), (a_4, a_6), (a_7, a_2), (a_1, a_3); \\ &(a_5), (a_4, a_6), (a_7, a_2), (a_3, a_1); \\ &(a_5), (a_6, a_4), (a_2, a_7), (a_1, a_3); \\ &(a_5), (a_6, a_4), (a_2, a_7), (a_3, a_1); \\ &(a_5), (a_6, a_4), (a_7, a_2), (a_1, a_3); \\ &(a_5), (a_6, a_4), (a_7, a_2), (a_3, a_1). \end{aligned} \tag{16}$$

After completion of the objective weighting, metrics are sorted by ascending importance as follows.

$$a_4, a_2, a_3, a_1, a_5, a_6, a_7. \tag{17}$$

The results show that the sorting results of metrics obtained by the two weighting methods are not exactly the same; the sorting results of metric groups are also not the same. Therefore, we take the linear combination coefficient $a = 1$. According to formula (7), the combination weight vector is calculated as $\mathbf{w} = \mathbf{w}_s = (0.221, 0.148, 0.221, 0.098, 0.066, 0.098, 0.148)^T$. The result can be explained as follows: The metric weight sorting obtained by the objective weighting methods deviates markedly from the subjective preferences of the user. Therefore, the result obtained by the objective weighting method has no reference value. Based on the main nonfunctional metrics and their weights, we have obtained the comprehensive evaluations of the three best instances from the RackSpace, DigitalOcean, and Linode service models as follows:

$$\begin{aligned} U_{\text{Best-instance-from-RackSpace}} &= 0.364592, \\ U_{\text{Best-instance-from-DigitalOcean}} &= 0.302921, \\ U_{\text{Best-instance-from-Linode}} &= 0.332487. \end{aligned} \tag{18}$$

Thus, the best matching instances from the three providers are sorted as follows:

$$\begin{aligned} &U_{\text{Best-instance-from-RackSpace}} > U_{\text{Best-instance-from-Linode}} \\ &\quad > U_{\text{Best-instance-from-DigitalOcean}}. \end{aligned} \tag{19}$$

7.4. Results Analysis. First, the IaaS discovery process implements the matching of supply and demand on IaaSs with a configuration instance granularity, in which the search for the matching instances covers the complete configuration space for each IaaS. Therefore, the QoS metrics of all matching instances can be obtained accurately so that the QoS of these instances can be objectively evaluated and compared, which ensures that the decision results are accurate. Second, the IaaS discovery and selection process realized in this paper is systematic and automatic. After

the user submits the demand of an IaaS and the subjective weight vector of the decision metrics, the entire process will be completed automatically. Thus the efficiency of IaaS discovery and selection is greatly improved. Finally, the performance of our methods is good. The repository of IaaS service models is updated and maintained periodically by the system. Thus, the main processing part of our methods lies in the matching of supply and demand on IaaS FMs. This processing part takes out the IaaS service model from the repository sequentially and then searches the best matching instances from the IaaS configuration space represented by the service mode. Assuming there are m IaaS services, each provides about n configuration instances, and the COMPLEX CONSTRAINTS representation of each instance involves about l conditional options; the time complexity is then $O(n \times m \times l)$. The experimental result shows that the searches for the best matching instances from the three IaaS service models are accomplished within 1 s.

8. Conclusions and Future Work

This paper systematically studied the description, discovery, and selection methods of IaaSs in cloud computing. First, we designed a standard description method for IaaSs that uses a fine-grained and structured way to describe the entire configuration space of an IaaS. The method overcomes the shortcomings of existing methods, which can only describe the macro information of IaaSs and provides the conditions for the automated discovery and selection of IaaSs. Second, we proposed a systematic discovery and selection method for IaaSs, which introduces the automated analysis techniques of FMs to analyze the service model and search for the best matching instance from each IaaS. The discovery efficiency is greatly improved. Moreover, due to the fine-grained matching of supply and demand on IaaSs, the matching results directly point to the specific configuration instances, so the accuracy of service discovery is effectively improved. Finally, we unified the decision metrics for service selection, and presented effective and feasible metric measurement methods. We also presented a combination weighting method for decision metrics based on the subjective and objective weighting methods. As such, the ranking result of alternative IaaS instances is more objective and reliable.

In reality, a combination of multiple IaaSs can sometimes meet the needs of some applications better, such as global e-commerce. If a combination of multiple IaaSs from different regions is used, the user requests will be effectively dispersed so that the system pressure can be greatly reduced and user experience can be improved. In addition, some applications have to be deployed on the combination of multiple IaaSs, such as a medical-related and big-data-analysis application, which needs to collect the original data from multiple countries for analysis and processing. If the laws of the relevant countries strictly prevent such original data flowing out of their territories, then users will have to buy a combination of multiple IaaSs located in the licensed areas. Therefore, there is a strong realistic significance in studying how to discover and select the best IaaS composition based on user needs.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the Nature Science Foundation of Jiangsu, China, under Grant no. BK20131069.

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] Cloudharmony, 2015, https://cloudharmony.com.

[3] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '10)*, pp. 104–112, May 2010.

[4] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, and J. J. Samper-Zapater, "Ontology-based annotation and retrieval of services in the cloud," *Knowledge-Based Systems*, vol. 56, no. 3, pp. 15–25, 2014.

[5] J. Kang and K. M. Sim, "Ontology and search engine for cloud computing system," in *Proceedings of the International Conference on System Science and Engineering (ICSSE '11)*, pp. 276–281, Macau, China, June 2011.

[6] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.

[7] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a catalogue of metrics for evaluating commercial cloud services," in *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing (GRID '12)*, pp. 164–173, Beijing, China, November 2012.

[8] E. Wittern, J. Kuhlenkamp, and M. Menzel, "Cloud service selection based on variability modeling," in *Service-Oriented Computing*, pp. 127–141, Springer, Berlin, Germany, 2012.

[9] J. García-Galán, P. Trinidad, O. F. Rana, and A. Ruiz-Cortés, "Automated configuration support for infrastructure migration to the cloud," *Future Generation Computer Systems*, vol. 55, pp. 200–212, 2016.

[10] J. García-Galán, O. F. Rana, P. Trinidad, and A. Ruiz-Cortés, "Migrating to the cloud: a software product line based analysis," in *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER '13)*, pp. 416–426, Aachen, Germany, May 2013.

[11] Amazon, 2015, http://aws.amazon.com.

[12] K. Kang, S. Cohen, J. Hess et al., "Feature-oriented domain analysis feasibility study," Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[13] G. Shen, W. Zhang, Z. Huang et al., "Description-logic-based feature modeling and verification," *Journal of Computer Research and Development*, vol. 50, no. 7, pp. 1501–1512, 2013.

[14] C. Quinton, D. Romero, and L. Duchien, "Automated selection and configuration of cloud environments using software product lines principles," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD '14)*, pp. 144–151, Anchorage, Alaska, USA, June 2014.

[15] J. Zhou, D. Zhao, and J. Liu, "TEFL: a textual feature modeling language," *Journal of Chinese Computer Systems*, vol. 33, no. 10, pp. 2133–2140, 2012.

[16] Amazon, "AWS Total Cost of Ownership (TCO) Calculator," 2015, https://awstcocalculator.com.

[17] Rackspace, 2015, http://www.rackspace.co.uk/solutions-configurator.

[18] Cloudscreener, 2015, http://www.cloudscreener.com.

[19] Cloudorado, 2015, https://www.cloudorado.com.

[20] Planforcloud, 2015, http://www.planforcloud.com.

[21] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based genetic optimization for deployment and reconfiguration of software in the cloud," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 512–521, San Francisco, Calif, USA, May 2013.

[22] M. Zhang, R. Ranjan, A. Haller et al., "An ontology-based system for cloud infrastructure services' discovery," in *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 524–530, Wuhan, China, November 2012.

[23] T. Han and K. M. Sim, "An ontology-enhanced cloud service discovery system," in *Proceedings of the International Multi Conference of Engineers and Computer Scientists (IMECS '10)*, vol. 2180, HongKong, March 2010.

[24] L. Liu, X. Yao, L. Qin, and M. Zhang, "Ontology-based service matching in cloud computing," in *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '14)*, pp. 2544–2550, IEEE, Beijing, China, July 2014.

[25] Cloud Service Measurement Index Consortium (CSMIC), "SMI framework," 2015, http://www.csmic.org/.

[26] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: a framework for comparing and ranking cloud services," in *Proceedings of the IEEE International Conference on Utility and Cloud Computing*, pp. 210–218, Melbourne, Australia, December 2011.

[27] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371–378, 2012.

[28] E. Wittern and C. Zirpins, "Service feature modeling: modeling and participatory ranking of service design alternatives," *Software & Systems Modeling*, vol. 15, no. 2, pp. 553–578, 2016.

[29] DigitalOcean, 2015, https://www.digitalocean.com.

[30] CSA, "CSA Security, Trust and Assurance Registry (STAR)," 2015, https://cloudsecurityalliance.org/star/.

[31] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[32] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*, pp. 1–14, ACM, Melbourne, Australia, November 2010.

[33] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12)*, pp. 85–96, Boston, Mass, USA, April 2012.

[34] Spec, 2015, http://www.spec.org.

[35] H. Zhu, L. Wu, and H. Kang, "Research of cloud provider selection method based on SecLA," *Computer Science*, vol. 43, no. 5, pp. 100–108, 2016.

[36] M. Wang, "A comprehensive analysis method for determinating the weight coefficients in comprehensive evaluation of multiple indexes," *Systems Engineering*, vol. 17, no. 2, pp. 56–61, 1999.

[37] J.-X. Liu, Y.-J. Tan, and H.-P. Cai, "Study of the methods of the linear combination weighting for multiple attribute decision-making," *Journal of National University of Defense Technology*, vol. 27, no. 4, pp. 121–124, 2005.

[38] H. Chen, "Combination determining weights method for multiple attribute decision making based on maximizing deviations," *Systems Engineering and Electronics*, vol. 26, no. 2, pp. 194–197, 2004.

[39] D. Song, C. Liu, C. Shen et al., "Multiple objective and attribute decision making based on the subjective and objective weighting," *Journal of Shandong University*, vol. 45, no. 4, pp. 1–9, 2015.

[40] Y. Li and J. Wang, "Method for deriving AHP weight based on sorting," *Ordnance Industry Automation*, no. 11, pp. 42–44, 2013.

[41] W. Chen and J. Xia, "Optimal combined weighting method based on the subjective and objective weights," *Mathematics in Practice and Theory*, vol. 37, no. 1, pp. 17–22, 2007.

[42] RackSpace, 2015, http://www.rackspace.com.

[43] Linode, 2015, https://www.linode.com.

[44] ISA Research Group, "FAMA tool suite," 2015, http://www.isa.us.es/fama/.

[45] M. Li, G. Chen, and Y. Chen, "Research on the method of index standardization in comprehensive evaluation," *Chinese Journal of Management Science*, no. 12, pp. 45–48, 2004.