2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Ontology-based Service Matching in Cloud Computing

Li Liu
School of Automation and Electrical Engineering
University of Science and Technology Beijing
Beijing, China
liuli@ustb.edu.cn

Xiaofen Yao
School of Automation and Electrical Engineering
University of Science and Technology Beijing
Beijing, China

Liangjuan Qin
School of Information Technology and Management
University of International Business and Economics
Beijing, China
ljqin@uibe.edu.cn

Miao Zhang
School of Automation and Electrical Engineering
University of Science and Technology Beijing
Beijing, China
s20130926@xs.ustb.edu.cn

Abstract—**This paper focuses on how to maximize accuracy of Cloud service discovery and give enough flexibility to Cloud customers to discover their best suited services from a range of Cloud providers. An ontology-based Cloud service discovery approach is proposed, which works based on modeling semantically enriched Cloud services, ontology reasoning and logic matchmaking. Cloud customers have different preferences for non-functional attributes, ranking discovered services according their preference can help select the most appropriate cloud service. Experimental results show that the discovered services not only meet customer's requirements in semantics but also satisfy QoS requirements given in the terms of SLA.**

Keywords—Cloud Computing; Ontology; Service Level Agreements (SLA); Service Discovery

I. INTRODUCTION

Cloud computing is a large scale application of distributing computing, whereby shared virtualized resources, software, and information are provided as virtualized-encapsulating services delivered on demand to customers over the Internet[1]. Cloud services have generally three different service delivery models, i.e., Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [2][3] .

Service discovery is a procedure of searching for required services which their functional and non-functional semantics satisfy a customer's goal. Cloud services are typically accessed using brokers. The broker allows customers to submit a service request to Cloud request, including required set Service Level Agreement (SLA) objectives for that service. The broker will then proceed to match available service descriptions of Cloud provider to service request description and find candidate services which can provide expected functionality. However, only functional service descriptions are not sufficient for service discovery process. Usually customers also have some non-functional QoS demands, such as a limited budget, the minimal time and cost, or a strict response time (deadline), etc. A key advantage of Cloud service is dynamically and automatically service on demand. Furthermore, as more and more Cloud services are available, there is often a case where many of them can satisfy functional demands of a service request. Therefore, it leads to the issue of quickly and efficiently matchmaking and ranking to select the best services for the requested among a list of candidate services.

This paper focuses on how to maximize accuracy of Cloud service discovery and give enough flexibility to Cloud customers to discover their best suited services from a range of Cloud providers. We use ontology to service discovery to enhance the service semantic information. We designed a Cloud ontology contained a set of concepts for similarity reasoning. These ontology concepts allows the broker have a better way to understand the meaning of a cloud service, and can further improve service matching results.

Each performance factor is defined in terms of SLA called Service Level Objectives (SLO) which are used for computing overall quality degrees of cloud services with respect to a request's QoS demands. Such SLO information can be performance (response time, latency etc.), availability, cost, security, etc., which have substantial impacts on user's expectation. Hence we use SLO as main factor to distinguish and rank cloud services. Furthermore, we propose an Ontology-based and SLA-aware service discovery algorithm, which apply ontology to match cloud services with user's requirements, as well we use SLO-based sequence vector ranking algorithm to select the fittest services. Experimental results show that the selected services not only meet customer's requirements in semantics but also satisfy the QoS demands given in the terms of SLA.

Our proposed approach brings the following two benefits: (1) In service matching phase, we consider the equivalence concepts which are same in semantics but

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

different in syntax. So our algorithm can improve accuracy of Cloud service discovery; (2) It can find the most suitable Cloud service taking into account of user QoS requirements specified in SLO.

The remainder of this paper is organized as follows: in the next section we discuss prior works related to service discovery method in cloud. In section 3 we propose SLA-based service discovery method. Then we rank the candidate services by user's non-functional property preferences in section 4. We make simulation environment for cloud via CloudSim and present evaluation results in section 5 to certify the validity of the methodology. Finally, section 6 concludes the paper with a brief summary and describes our future research directions.

II. RELATED WORK

The process of service discovery is to find out appropriate service which could satisfy the user's requirements among a range of services. Current approaches to service discovery can be divided into symmetric attribute-based and semantic-based matching between user requirements and services provider[4]. In a heterogeneous Cloud environment, it is difficult to make syntax and symmetric of QoS descriptions of services and user requirements. Therefore, building semantics descriptions can provide an inter-Cloud language which helps customers quickly and accurately discover the required service.

Existing cloud service discovery methods are as follows. Le Duy Ngan et al. [5] presented OWL-S based semantics cloud service discovery and selection system, which adopts the OWL-S (Ontology Web Language for Services) Language to describe Cloud services and user requirements.

The service discovery process supports complex constraints and makes the semantics match dynamically. Amir Vahid Dastjerdi et al. [6] proposed a WSMO-based cloud service discovery method, by using Web Service Modeling Ontology (WSMO) language. The ontology concepts of virtual units and requirements are defined in WSMO, and translate the user's requirements such as OS and hardware requirements into a standard semantics.

Then an ontology-based service discovery method is proposed to search for appropriate services. This method can help users find suitable applications effectively from different suppliers. Since most users want to find service accurately, while QoS is a critical factor for service accuracy and it is able to distinguish the web services which have similar function.

In this case, Pon Harshavardhanan et al. [7] proposed a QoS-Broker architecture for dynamic web services discovery, which the QoS properties are stored in the database by QoS Broker (as a behalf of service providers). Users can search for specific services in QoS database via service select component which can help to select appropriate services according to their functional or nonfunctional requirements and personal preferences. Jaeyong Kang et al. [8] presented a multi-criteria cloud service search engine. In order to enhance the accuracy of service discovery, the user can specify the functional, technical and cost requirements as input parameters. The system can return service list according these parameters based on the similarity reasoning between cloud services and users requirements. Michael Brock et al.[9] proposed cluster as a service mode in the cloud environment, which helps users discovery, selection and use the existing computing cluster service.

Different from existing service matching methods, the Ontology-based matching algorithm presented in this paper has considered the equivalence concept to increase the service matching success rate. In order to help user select the fittest services, we propose a service selection method which combines the analytic hierarchy process (AHP)[15] with preference ranking organization method for enrichment evaluations (PROMETHEE)[16], in which AHP is used to determine the weights of attributes according to users' preferences, and PROMETHEE is used to obtain the final rank of candidate services.

III. SLA-BASED SERVICE MATCHING ALGORITHM

The typical architecture of a Cloud services includes three roles, namely Cloud customer, Service broker and Cloud service provider. The Cloud customer sends a service request to the broker, and then the broker searches for the service repository based on required SLO; finally matching service providers should be sent back by the broker. If the broker finds a set of services satisfy the functional requirements of customer, how to select the fittest service based on the user preferences is a key issue.

A. Cloud Service Ontology

Ontology can be defined as explicating semantics of a shared concept. It also provides a shared understanding of a domain of interest to support communication among computers and human by defining shared and common domain theories [10][11]. Ontology has been developed to facilitate knowledge sharing and reuse in the field of artificial intelligence. Ontology can be used to describe properties of service demand and capabilities to enable Cloud service providers to advertise their services. Considering one type of Cloud services IaaS as an example, there are a number of computing resources, such as processing power, storage, networks, and other fundamental computing resources. The purpose of creating cloud Ontology is to enhance semantic information and let the computer understand the meaning of the concept expressed by service provider and customer, to realize semantic-based services discovery.

The domain model and vocabulary for expressing service are showed in Fig. 1. Our IaaS Cloud ontology defines the hierarchical relations of Cloud concepts. For example, there

are four different concepts (*CPU,RAM,HDD,OS*) in each service of IaaS. Each concept has several sub-concepts

Fig. 1 IaaS cloud ontology

respectively, such as *OS* has children nodes *Linux*, *Windows, Apple*. *Windows* include *WinXP*, *Win2000*, *Vista*, and so on.

Fundamentally the ontology reasoning is used to extract implicit knowledge of concept, and get the matching degree of any two concepts by reasoning and calculating the similarity between two concepts. On the base of Cloud ontology, the similarity reasoning is explained in next section.

B. Matchmaking method

S is property vector of IaaS, *R* is user's requirements vector for IaaS service . Service matching is to find the appropriate service to satisfy the user's functional goal based on the ontology concepts.

From the ontology concepts of IaaS cloud depicted in fig. 1, we know that some concepts are same in semantics but different in syntax, as well there are some inheritance relation between concepts. For the concepts with same semantics but different syntax, we defined them as equivalence matching. The inheriting concepts are defined containing matching. The four matching type include same comparison, equivalence matching, containing reasoning and similarity matching. The matching step is shown in fig. 2.

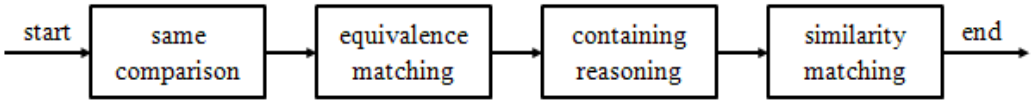

Fig. 2 the matching procedure

- same comparison

We firstly calculate the concept similarity between user's requirements and provided cloud services. Only when each concept in service S is exactly equal to corresponding concept in request R, that is service S satisfy request R . If same comparison is false, then go to the next equivalence matching step. For example, $R_i(R_i \in R)$ is Win2000, and the corresponding concept $S_i(S_i \in S)$ is Win2000, then the two concepts are same.

- equivalence matching

If two concepts are not exactly equal, then carry out the equivalent matching. If the result of matching is

equivalent, then matching the next concept in S and R until the end, else go to the next containing reasoning step. For example, R_i is Win2000 and the corresponding concept S_i is Windows2000, that is the two concepts are equivalent matching.

- containing reasoning

If two concepts are not exactly equal and equivalent, then execute containing reasoning, which is mainly to solve the inheritance relationship between concepts. If the comparison result shows that the two concepts are containing relation, then matching the next concept in S and R until the end, else go to the next similarity matching step. For example, if R_i is Win2000, and the corresponding concept S_i is Windows, that is to say S_i contains R_i, so the two concepts are containing relation.

- similarity matching

If the relationship between each concept in requirement and service is not any one of the above three cases, then the similarity matching is performed, which is to calculate the degree of similarity between two concepts. If the degree of similarity is no less than user's threshold, two concepts are defined as similarity. For example, R_i is Win2000, and the corresponding concept S_i is Linux, then the two concepts are similar.

If the relation of concept between request and services provider is not any of the above four cases, that is this service unsatisfied with the user's request. Then to find the other cloud service from the registry to match with user's requests until all services processed.

C. Similarity calculation

The similarity value between R and S is calculated by Eq.1.

$$Sim(R,S) = \sum_{i=1}^{n} \omega_i Sim(R_i, S_i) \quad (1)$$

Where $\omega_i \in [0,1]$ is weight coefficient defined the degree of influence, and $\sum_{i=1}^{n} \omega_i$=1.

$$\mathrm{Sim}(\mathrm{R_i}, \mathrm{S_i}) = \mathrm{Sim_{conp}}(\mathrm{R_i}, \mathrm{S_i}) + \mathrm{Sim_{prop}}(\mathrm{R_i}, \mathrm{S_i}) \quad (2)$$

$i = (1, \ldots, n)$, where $Sim_{conp}(R_i, S_i)$ is the concept similarity, $Sim_{prop}(R_i, S_i)$is the property similarity.

$Sim_{conp}(R_i, S_i)$ is calculated by counting common reachable nodes between two concepts [12][13]. the concept similarity is calculated as Eq.3:

$$Sim_{conp}(R_i, S_i) = \frac{|\alpha(R_i) \cap \alpha(S_i)|}{|\alpha(R_i)|} + (1-\rho)\frac{|\alpha(R_i) \cap \alpha(S_i)|}{|\alpha(S_i)|} \quad (3)$$

Where, $\rho \in [0,1]$ is the influence degree. $\alpha(R_i)$ is the number of reachable nodes from root to R_i, $\alpha(R_i) \cap \alpha(S_i)$ is the number of the reachable nodes shared by R_i and S_i, represent the commonality between concepts R_i and S_i. Such as the *CPU* concept in cloud ontology shown in fig.1, there are three reachable nodes from the concept *CPU to Pentium*, two reachable nodes from the concept *CPU to AMDCPU*. The concept *Pentium* and *AMDCPU* have only one common reachable node, i.e. $\alpha(Pentium) = 3$, $\alpha(AMDCPU) = 2$, $\alpha(Pentium \cap AMDCPU) = 1$.

If set $\rho = 0.5$, then the similarity between *Pentium* and *AMDCPU* is:

$$Sim(Pentium, AMDCPU) = 0.5 \times 1/3 + 0.5 \times 1/2$$

if the threshold $T_c \leq 0.4$, then the two concepts are similar.

The property similarity$Sim_{prop}(R_i, S_i)$ can be calculated by Eq.4.

$$Sim_{prop}(\mu(R_i), \mu(S_i), C) = 1 - \left|\frac{\mu(R_i) - \mu(S_i)}{C_{max} - C_{min}}\right| \quad (4)$$

Where $\mu(R_i)$ and $\mu(S_i)$ represent the value of property, C represents the concept that owns the property, C_{max} and C_{min} indicates the range of the property. For example, the maxmum size of hard disk is 800G, and the minimum size is 100G, then the similarity between 400G and 500G hard disk is $Sim_{prop}(400,500,HDD) = 1 - \left|\frac{400-500}{800-100}\right| = 0.86$, if the similarity value is greater than the threshold , the matching successfully.

IV. SERVICE RANKING

Most users not only have functional requirements, they also have some non-functional property requirements. For example, users require a limited cost of services or minimal response time. Based on non-functional properties preference descript in SLO, the candidate services having similar function are ranked to find appropriate service by our service selection method which combines the analytic hierarchy process (AHP)[15] with preference ranking organisation method for enrichment evaluations(PROMETHEE)[16]. AHP is used to determine the weights of user's preferences, and PROMETHEE is used to obtain the final rank of candidate services.

A. Weights of Metric

The AHP method is used to determine the weights of QoS metrics.

- Construct the hierarchy

As a example which QoS metrics including response time, cost, reliability and availability, the hierarchy is generated as shown in Fig.3. The higher level is determined or measured by lower level attributes.

- Assign weight to each metric

Users adopt a pairwise comparison mechanism to determine the relative priority of each metric, and construct pairwise comparison matrix A. For example, comparing the

relative importance between response time and cost, that is to determine which one has a greater impact to the user's preference in the case of other properties are the same, and

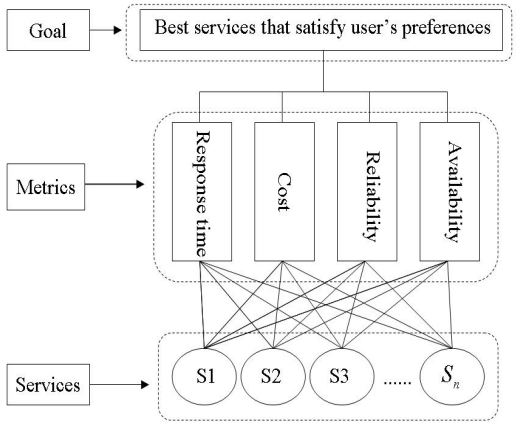

Fig. 3 The AHP hierarchy model of service selection

how much the degree of influence will be. Usually integer1-9 is used for pairwise comparison: 1 means the equal importance, 9 means the highest level of importance. After the pairwise comparison, matrix $A = (\alpha_{ij})_{m\times m}$ (m is the number of metrics) is as (5).

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} \qquad (5)$$

where $\alpha_{ii} = 1, \alpha_{ij} = 1/\alpha_{ji}$, The weights for each metric are given by the right eigenvector w corresponding to the highest eigenvalue $\lambda_{max}(A)$, The weight ω derived by matrix A, and obtained by the equation $A\omega = \lambda_{max}(A)\omega$, $\omega = (\omega_1, \ldots, \omega_m)^T$.

- Consistency check

The purpose of consistency check is for testing coordination of the important degree between each attribute. To avoid appearing the contradiction such as: A is more important than B, B is more important than C, and C is more important than A. For example, it is inconsistent in the case where a service user thinks that response time is strongly more important than reliability, and reliability is moderately more important than cost, and cost is moderately more important than availability. The consistency index (CI) is shown as Eq.6

$$CI = \frac{\lambda_{max} - n}{n - 1} \qquad (6)$$

The consistency ratio (CR) is calculated as $CR = \frac{CI}{RI}$, indicates whether the evaluations are sufficiently consistent. The RI is obtained from the random consistency index table. If $CR \leq 0.1$, the consistency rate is acceptable; otherwise, matrix A needs to be revised to calculate again. Finally, the weight of each trust metric can be obtained.

B. Service ranking

PROMETHEE is used to rank the candidate services, and the process is shown in the following:

Define the preference functions and parameters for metrics. PROMETHEE performs pairwise comparisons by considering the deviation between the evaluations of the alternatives. Two candidate services s_1, s_2 compare on the property j ,and the comparison results express with the preference function $P_j(s_1, s_2)$, the greater the preference function value, the better performance the candidate service on that attribute. The preference function is denoted as Eq.7.

$$P_j(s_1, s_2) = f_j[d_j(s_1, s_2)], \forall s_1, s_2 \in S \qquad (7)$$

where $d_j(s_1, s_2) = Q_j(s_1) - Q_j(s_2)$ is the difference between the evaluation of tow service for criterion Q_j. PROMETHEE has six pre-defined preference functions(PF) according to the inherent characteristics of the metrics. The preference function of response time is a linear criterion. When the difference is smaller than the indifference threshold it is considered as negligible (criterion preference degree is equal to zero). If the difference exceeds the preference threshold it is considered to be significant (criterion preference degree is equal to one). When the difference is between the indifference and preference thresholds, an intermediate value is computed for the preference degree using a linear interpolation. The preference function of price is a V-shaped criterion, meaning that in a certain acceptable range, preference increases linearly with the difference, out of the acceptable range, the low cost service has an absolute advantage, and preference degree is 1. The preference function of reliability and availability is a Gaussian criterion.

When a preference function has been associated to each criterion by the decision maker, all comparisons between all pairs of actions can be done for all the criteria. A multi-criteria preference degree as Eq.8 [16]:

$$\pi(s_1, s_2) = \sum_{j=1}^{m} \omega_j P_j(s_1, s_2) \quad \forall s_1, s_2 \in S \qquad (8)$$

where $\pi(s_1, s_2)$ measures how much s_1 is preferred over s_2 on all the property metrics.

The positive, negative and net preference flows[16] is as from Eq.9 to Eq.11 differently.

$$\phi^+(s) = \frac{1}{|S| - 1} \sum_{x \in (S - \{s\})} \pi(s, x) \qquad (9)$$

$$\phi^-(s) = \frac{1}{|S| - 1} \sum_{x \in (S - \{s\})} \pi(x, s) \qquad (10)$$

$$\phi(s) = \phi^+(s) - \phi^-(s) \qquad (11)$$

Where $|S|$ is the number of service S. The positive preference flow $\phi^+(s)$ quantifies how a candidate service S is preferred to all the others, while the negative preference flow $\phi^-(s)$ quantifies how a candidate service S is being preferred by other services. An ideal action would have a positive preference flow equal to 1 and a negative preference flow equal to 0. The positive and negative preference flows are aggregated into the net preference flow $\phi(s)$ calculated by equation (11), which is defined as the intersection of these two rankings.

C. *Ranking algorithm*

Step1: obtaining non-functional property from the candidate services, such as the cost, response time, reliability and availability.

Step 2: The AHP method is used to determine the weights of property metrics, first constructing the hierarchy which contains the target layer, the property layer and the solution layer. Then we use a pairwise comparison mechanism to determine the relative priority of each metric and getting comparison matrix A. Finally verify the consistency of the matrix A.

Step3: Select the preference functions base on the inherent characteristics of the property metrics. Using PROMETHEE method to calculate the positive preference flow ϕ^+, the negative preference flow ϕ^- and the net preference flow ϕ.Then ranking the candidate services based on the net preference flow ϕ.

Step4: Finally, the rank of candidate services can be generated and the service with higher preference degree will be selected.

V. EXPERIMENT RESULTS

In this section our approach is evaluated on a case study to show the effectiveness of the proposed algorithm and tested in cloud computing simulation software CloudSim[17]. First we use ontology development tool Protégé3.4.8 to build IaaS cloud ontology. Then we use Cloudsim to call ProtégéOWL-API to read into the OWL file, so the application can access ontology information which are stored in OWL file, then execute the ontology query and reasoning operations. In order to verify the effectiveness of the matching methods, we test a series of data set in different range among 10-50 services.

Our experiment includes the following three cases: A is without the cloud ontology, in this situation we can get the matching results only when the two concepts are equal exactly. B is with the cloud ontology, consider containing reasoning and similarity matching, but not consider the equivalence concept. C is with the cloud ontology, and takes into account the equivalence concept. The precision ratio of service matching is shown in fig.4, which is the ratio of the number of success matching to the number of all related candidate services. As fig.4 shown, the precision ratio increased significantly when considering the cloud ontology and the non-functional attributes of the user preferences.

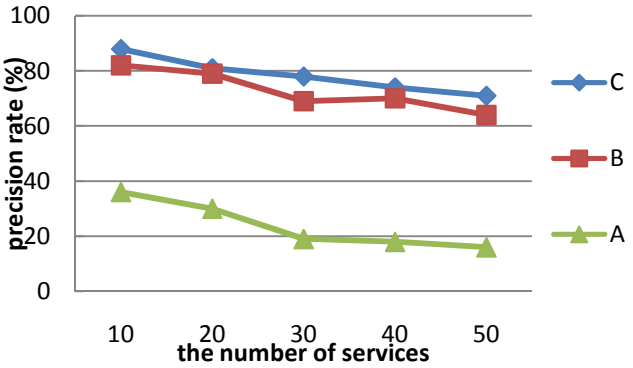

Fig. 4 precision rate of service matching for three cases

In order to verify whether the proposed service matching method can effectively express a user's personalized preference, we assume that there are two users named U_1 and U_2 needing to decide which service to select. The user's preferences are expressed as follows:

- Assume that U_1 sets response time as the most important metric, followed by cost, reliability and availability. The pairwise comparison matrix A_1 is constructed. $\lambda_{max} = 4.18$, $CR = 0.07 < 0.1$ .The weights are $\omega = [0.44,0.081,0.119,0.359]$

- Assume that U_2 sets availability as the most important trust metric, followed by response time, reliability and cost. The pairwise comparison matrix A_2 is constructed. $\lambda_{max} = 4.201$ and $CR = 0.075 < 0.1$. The weights are $\omega = [0.249,0.166,0.049,0.535]$. Then, the weights are filled into Table I, and the positive, negative and net outranking flows are calculated separately.

TABLE I **PREFERENCE FUNCTIONS AND PARAMETERS**

Parameters	Time	Price	Reliability	Availability
Min/Max	Min	Min	Max	Max
Weight	ω_1	ω_2	ω_3	ω_4
PF	Linear	V-shape	Gauss	Gauss
q	50	-	-	-
p	150	100	-	-
s	-	-	5	5

We select the top 30% services of the ranking queues, and calculate the average metric values of all candidate services. In order to show the intuitive comparison results, the response time and price are decreased 10 times. The comparison results of service selected with different QoS metric considering user's preference are shown in Fig.5.

Fig.5 show that the average values of U_1's response time and cost of the selected services are better than U_2's, for the reason of U_1 taking the response time as the most important metric. While the average values of U_2's availability and reliability of the selected services are better than U_1's according to U_2's preference. Generally, the experimental

results show that services selected by U_1 and U_2 have faster response time and higher availability respectively, i.e. our method can express users' preferences for multi metrics.

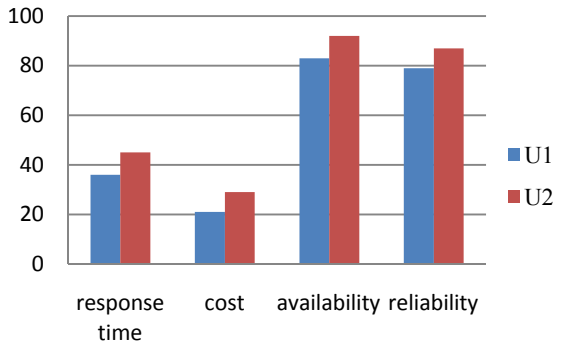

Fig. 5 metric values of selected services

VI. CONCLUSIONS

Ontology-based and SLA-aware service matching method is proposed in this paper, in which we calculate similarity between two concepts by cloud ontology, and consider the equivalence concept in service matching procedure. At the same time based on user's requirement of various non-functional properties, we proposed a service selection method which combines the analytic hierarchy process (AHP) with preference ranking organization method for enrichment evaluations (PROMETHEE) to rank the available services. Experimental results show that the proposed method can effectively find the cloud service meeting user's requirements.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China(Grant No 61370132) and the State High-Tech Development Plan (No.2013AA01A601).

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, "Cloud Computing and Grid Computing 360-Degree Compared," IEEE Grid computing Environment Workshop,2008. GCE'08. pp:1-10, Nov. 2008.

[2]R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, "Cloud computing and emerging IT platforms:Vision, hype, and reality for delivering computing as the 5th utility," Future Generation Computer Systems, vol. 25, no. 6, pp.599-616, June 2009.

[3] A. V. Mladen, "Cloud Computing-Issues,Reasearch and Implementations," Journal of Computing and Information Technology-CIT, vol. 16, no. 4, pp.235-246, June. 2008.

[4]D. Fensel, F. Facca, E. Simperl, I. Toma, "Web service modeling ontology (Book style)," Semantic Web Services Springer Berlin Heidelberg, 2011, pp. 107–129.

[5]Le Duy Ngan, R. Kanagasabai, "OWL-S Based Semantic Cloud Service Broker," 2012 IEEE 19th International Conference on Digital Object Identifier, 2012 , pp: 560 – 567.

[6]A.V. Dastjerdi, S. G. H. Tabatabaei, R. Buyya, "An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery," 2010 10th IEEE/ACM International Conference on Digital Object Identifier 2010, pp: 104-112.

[7]P. Harshavardhanan, J. Akilandeswari, R. Sarathkumar, "Dynamic Web Services Discovery and Selection Using QoS-Broker Architecture," 2012 International Conference on Digital Object Identifier, 2012, pp: 1 – 5.

[8]K. Jaeyong, M. S. Kwang, "Cloudle: A Multi-criteria Cloud Service Search Engine," 2010 IEEE Asia-Pacific Services Computing Conference, 2010, pp:339-346.

[9]M. Brock, A. Goscinki, "Toward Ease of Discovery,Selection and Use of Clusters within a Cloud," 2010 IEEE 3rd International Conference on Cloud Computing, 2010, pp:289-296.

[10]H. Stuckenschmidt, "Ontology-based information sharing in weekly structured environments," Ph.D. thesis, AI Department, Vrije University Amsterdam, 2002.

[11]K. Jaeyong, M. S. Kwang, "Cloudle:An Agent-based Cloud Search Engine that Consults a Cloud Ontology," Annual International Conference on Cloud Computing and Virtualization(CCV 2010), 2010, pp:312-318.

[12]H. Taekgyeong, M. S. Kwang, "An Ontology-enhanced Cloud Service Discovery System," Proceedings of the International MultiConference of Engineers and Computing Scientists, pp.17-19, March 2010.

[13]Liangxiu Han, Dave Berry, "Semantic-supported and agent-based decentralized grid resource discovery," Future Generation Computer Systems, vol. 24, no. 8, pp. 806-812, Oct 2008.

[14]O. S. Vaidya, S. Kumar, "Analytic hierarchy process: An overview of applications," European Journal of Operational Research, vol. 169, no. 1, pp.1-29, Feb. 2006.

[15]C. Herssens, I. J. Jureta, S. Faulkner, "Dealing with quality tradeoffs during service selection," Proceeding of the International Conference on Autonomic Computing(ICAC), 2008, pp.77-86.

[16]C Herssens, IJ Jureta, S Faulkner.Dealing with quality tradeoffs during service selection[C]//Proceeding of the International Conference on Autonomic Computing(ICAC) , Chicago, IL,USA,2008: 77-86.

[17]R.N. Calheiros, R. Ranjan, A. Beloglazov, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23-50, 2011.