www.ijiarec.com

ISSN:2348-2079

Volume-6 Issue-1

International Journal of Intellectual Advancements and Research in Engineering Computations

Cloud service mentor system using ambiguous connectives and refined user request handling methodology – a survey

[1]M.Beemajan Shaheen, [2]M.Ramu

[1]Asst. Prof/CSE, SVCET, Puliyangudi

[2]Asst Prof & Head, MCA/SVCET, Puliyangudi

ABSTRACT

Cloud service selection under diverse nature of the cloud services is still an open issue. Unlike web services which are all selected based on WSDL description, cloud services cannot be selected as there is no standard representation of cloud services as most of the cloud services are described in natural language. In this paper we proposed a cloud service mentor system which is built upon semantic technologies and has three ontology structures for cloud services namely SaaS, PaaS, and IaaS ontology. This system parses the cloud service description documents through natural language handling method and major concept of the cloud service is identified. Based on the major concept of the cloud service the cloud service ontology is updated. The user request is processed using natural language handling methods and ambiguous connectives (AND, OR) are used to refine the request based on first order horn clause logic and DNF. Then the SPARQL queries are generated to retrieve the corresponding cloud services from ontology repository. The performance evaluation of the system shows that this system outperforms other methods and provides better results.

Keywords: cloud service ontology, cloud service concept, service mentor, horn clause logic, SPARQL

INTRODUCTION

Cloud computing is a paradigm which evolved from the technologies like grid computing and service oriented architecture (SOA). Cloud computing is an elastic on demand self-service that are provided from a shared pool of resources. Every technology like software, infrastructure, security, platform are provided as services through cloud computing. The user can request a service, make use of them and pay according to their usage time and amount of usage. The advantage of cloud computing is, the user don't need to know about the technical details like platform, software, etc. to use a cloud service, where every kind of issue is taken care by the cloud service providers at their end. Thus it helps the small and medium sized organizations to get profit over the resource by using it from the cloud instead of investing huge amount on those resources by themselves. Before cloud computing, web services played a major role in the information technology domain. Web Services provide a means to invoke services across the web which is similar to remote procedural calls and to support interoperable machine-to-machine interaction over the network. WSDL (Web Services Description Language) is a standardized way to describe service structures using XML [1]. Web Services are used in the applications of Grid Computing, Enterprise System based on SOA (Service-Oriented Architecture) and Electronic Commerce. Web Services lack semantically interesting properties that are needed for dynamic service discovery and invocation. It provides only syntactic description of the services.

Many cloud service providers are evolving tremendously today and it's the user job to select the most appropriate service among them according to his requirements. Cloud service providers

Author for correspondence:
Asst. Prof/CSE, SVCET, Puliyangudi
Email: shaheenzahir21@gmail.com

provide their services under three categories – Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) [27]. Web services like e-mail, file conversion, word processor etc are provided as a Software as a Service (SaaS) through different cloud service providers. PaaS services provide support for constructing and delivering web applications and internet available services. IaaS provides computing infrastructure like virtual machine, computer hardware and database as its service. Skilled users of cloud computing can easily select the service they need by analyzing the cloud service properties through the cloud service description documents, service level agreements which are all provided by the cloud service provider itself. But for unskilled users it is a difficult task and still an open issue of discovering a cloud service. It will be a better solution if the user selects a cloud service based on the semantics of the service instead of depending on syntax of the service like web services. The cloud service providers normally describe about their services in the form of service description documents, service level agreements and the technical details in WSDL format. To provide a meaningful way to parse those documents which are in natural language, semantic technologies could be used.

Semantic Web can be described as "knowledge representation meets the Web" to enable automatic handling and composition of information in the Web by means of ontology [8]. Semantic Web describes the domain information that can be processed by machine using Ontologies – set of conceptual terms labeled by URLs that are used in XML documents to give XML structures the semantics required by automatic reasoning. Semantic Web services are used extensively in health care, life sciences, clinical research and translational medicine domain to gain remarkable benefit as they depend on the interoperability of information from many disciplines. Semantic Web services are technology to facilitate dynamic service discovery, composition and invocation of Web Services. It combines the technologies from the field of Web Services, Semantic Web and automatic reasoning. Major representation framework used for Semantic Web Services are OWL-S (Web Ontology Language for Services) [31], WSMO (Web Service Modelling Ontology) [12] and SWSF (Semantic Web services Framework) [5]. Semantic web service discovery approaches reduces cost and time and improves the precision and recall parameters.

By utilizing semantic web technologies on cloud service discovery an unskilled user will be able to retrieve the cloud services which are all described in natural language. By considering the above said ideology, we proposed a semantic technology based system which uses an ontology structure for the standard cloud services namely PaaS, IaaS and SaaS. This ontology structure represents the knowledge explicitly about the cloud services based on the concepts of the cloud service by parsing the cloud service description documents through natural language handling method. Based on the ontology structure, we frame a service concept matrix whose column corresponds to a semantic concept of cloud service and row corresponds to the cloud service provider. A cloud service requester can provide his request in natural language through the user request interface which in turn parses the request for request concept. Then the requirement is refined by request refiner using ambiguous connectives (AND, OR) technique and SPARQL queries are generated to retrieve the matched cloud services according to the user requirements from the cloud service ontology repository. The reminder of this paper is organized as follows. Section II describes about few related works that are available under semantic web service discovery and composition. Section III provides information regarding the cloud service ontology which is populated by parsing the service description documents from cloud service providers. Section IV describes about service ontology population and service discovery methodology which is based on four algorithms described under sub sections. Section V describes about the experimental analysis of the system followed by conclusion under section VI.

RELATED WORK

The problem of web based service selection involves multi criteria decision making scenario and there has been many research methods proposed by many researchers, and still is an open issue under academic and industrial sectors. In [28], the authors proposed an ontology

bootstrapping process for web services. Web services are described using two components namely WSDL and textual descriptions. The proposed system employs methods like Term Frequency/Inverse Document Frequency (TF/IDF), web context generation and Free Text Description verification on these two components to provide accurate ontology by resolving inconsistency. The above said bootstrapping method for ontology construction requires human intervention for handling a predefined set of web services. The performance of the approach is comparatively low and it does not automatically verify the new concepts and relations to identify any redundancy. In [17], the authors presented a composition technique for service composition. WordNet is used for identifying the sense of words in the WSDL and disambiguate to get the concept specific to the context. These concepts are then inserted into the QSQL which is a data structure containing two parts: a domain link that avoids reasoning a concept that already exists and a domain data that records the service information in Input/Output data vectors. If there is no match between the user request and sub goal, then search the EQSQL and retrieve the related service model. This EQSQL based automatic service composition technique is evaluated only for specific domains and the effectiveness of the approach is not satisfactory. The QoS properties of the services are also not considered which an important factor for service composition and discovery.

In [20], the authors proposed a multimedia service selection method based on weighted Principal Component Analysis (PCA). PCA is used to transform a set of correlated QoS criteria to a set of independent components. For each candidate multimedia service overall utility value is calculated and service with highest utility value is recommended to the user. The proposed PCA method is based on user assigned weight values for QoS and it becomes difficult to assign appropriate weight values for dozens of QoS criteria. This approach is not effective if there are only few multimedia services. In [19], the authors proposed a formal definition of context-independent similarity and show that a Web service can be substituted by an alternative web service. Petri nets are used to model web services in such a way to concentrate on a particular property namely well structuredness. An algorithm has been designed to check the similarity between two web services by searching the communicating reachability graphs (CRG) and verifying the similarity. CRG is used to analyze the behavior of SWN based on the state analysis. A tool has been developed using this algorithm to automate the verification of similarity by integrating with Platform-Independent PN Editor. The approach is not efficient particularly with the increase of the state space. Complexity of the problem is not much reduced.

In [23], the authors proposed a service discovery platform that retrieves the most related semantic cloud service from a set of key word based retrieved cloud services. The cloud resources and concepts described in natural language are automatically annotated with the semantic concepts using GATE framework. Using classic vector space model the semantic index is framed from the annotated resources. Finally the most matched services between request and particular cloud service were retrieved using the cosine formula. In [22], the authors proposed the cloud service discovery through semantic knowledge by creating a semantic repository and service extraction for the requirement of the user. Semantic repository consists of semantic descriptions of cloud service. Each service is represented as a vector in which each dimension corresponds to a separate concept of the domain ontology. The semantic search engine then calculates a similarity value between the request q and each service s. This is done by using the cosine similarity. Despite of the vast research efforts on web services, there is a need to develop a standard model for defining the cloud service structure and to retrieve the cloud services according to cloud user requirements. In this paper we are addressing this issue using natural language handling techniques by parsing the cloud service description documents and updating the cloud service ontology accordingly with the cloud service concept. The user request is also parsed for concept matching and the corresponding cloud services are retrieved accordingly by refining the user request based on ambiguous connectives.

CLOUD SERVICE ONTOLOGY STRUCTURE

Cloud service providers describe their services in the documents like Service Level Agreement (SLA), Service Description Document (SDC) and WSDL documents. Technical information such as hardware and software are described in the SDC documents. SLA provides information such as service time and trading parties. This information is well understood by the technical users but the naive users can't understand them. Hence a knowledge structure must be provided to understand their service information. So we develop an ontology structure for the cloud service as our first step. Ontology is the means of explicitly expressing the concepts. We define three ontology structures for the three standard cloud service categories namely, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). On the whole we had defined 32 service concepts for 3 cloud service category. The services provided by each category, number of concepts, classes and individuals defined are shown in the table I.

Table I. Ontology services and concepts

Ontology Structure	Service	Concept	Class	Individual
SaaS	Web Service like e-mail and file conversion	16	820	245
PaaS	Computing Platform such as middleware and deployment environment	5	470	52
IaaS	Infrastructure such as storage, virtual machine and database	11	520	126

ONTOLOGY POPULATION AND SERVICE DISCOVERY

System architecture is shown is figure I. The work flow of the system is divided into two phases namely ontology generation and service discovery. There are 5 components. They are

1. Service Crawler
2. Service Concept Extractor
3. Service Ontology Repository
4. User Request Interface
5. Service Retriever

Service Crawler

The Service Crawler has been developed with Java and is responsible for traversing the web and retrieving the documents such as WSDL, SLA and SDC documents. The retrieved service lists are based on [33] that provide information regarding different cloud service providers under various categories. These service documents are the input for the concept extractor. Apart from these service descriptions, web services dataset from [2] have also been used. The WSDL files of various web services are then converted in to OWL files using CODE [29] plug-in. An example of WSDL file and the corresponding OWL file of the service is given table II.

Service Concept Extractor

Concept definer comprises of two subcomponents. They are PoS tagger and the Service annotator. Natural Language text document such as SLA and SDC are retrieved through service crawler and parsed using Stanford log linear parser [18]. Nouns [NN, NNP, NNS and NNPS] represent the resources and classes of cloud service whereas the verbs (VB, VBZ and VBN) represent the properties that define the relationship among the resources of the cloud service. The nouns that are identified by the PoS tagger are then given to the Service Annotator which is based on WordNet [9]. Word Net provides the lexical concepts of cloud service through the synsets. By considering weight of each concept identified through PoS tagger in the cloud service documents it helps us to define the major concept C of the cloud services. Weight of the concept can be defined on the basis of its frequency of occurrences in the document. The concept with the maximum weight/frequency is darkened as the major concept C of that particular cloud service. The weight/frequency of the particular concept is shrewd by frequency-inverse document (tf-idf) algorithm [16]. It is calculated as

$$W(c)=(freq_{t,d} / freq_{j,d}) * \log(N_{d,s} / N_{d,c}) \quad \text{----- (1)}$$

Where w(c) represents the weight of each concept, $freq_{t,d}$ represents the concept c in the

current document, $freq_{j,d}$ represents the total number of terms in the current document d, $N_{d,s}$ represents the total number of retrieved document for particular cloud service s and $N_{d,c}$ represents the number of documents that represent the concept c.

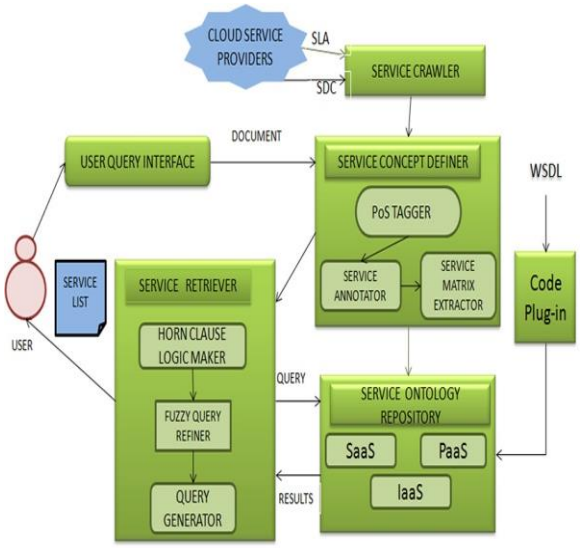

Figure I. System Architecture

Ontology Formation

Once the major concept had been identified, the similarity relation among the concepts has to be identified. The similarity among the concepts can be well identified using the joint probability distribution. If A and B are two concepts then the four joint probability distributions will be P(A,B), P(A',B), P(A,B') and P(A',B'). One such joint probability distribution based similarity matching was given by Paul Jaccard known as Jaccard coefficient [13]. According to Jaccard coefficient if A and B are two concepts then similarity matching S_J is given as

$$S_J = \frac{P(A \cap B)}{P(A \cup B)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$= \frac{P(A,B)}{P(A,B)+P(A,B')+P(A',B)}$$

The value of S_J is zero (lowest value) when A and B concepts are mutually exclusive concepts (eg: e-mail and virtual deployment) and one (highest value) when A and B points to the same concept (eg: e-mail and net mail). If S_J value is one, then the concepts are made as equivalent classes. If it is below 0.5 then they are assumed to be disjoint else they are given parent child relationship (eg e-mail and inbox) in the ontology structure. An example of Amazon S3 cloud service ontology structure and the OWL file of the service are given in table III.

Service Concept Matrix Formation

The major concept C will be the service type provided by the cloud service provider according to their service description documents such as SLA and SDC. The concept service matrix is generated for each identified service provided by various cloud service providers. The concept service matrix is of order m*n where m corresponds to the number of service providers providing the identified major concept and n corresponds to the number of sub concepts identified for the major concept. For instance if database is the major concept identified

and virtual machine, No SQL, My SQL database, big data analytics, Java, PHP, Ruby, JRuby, Python and GO are the sub concepts of IaaS, then the database service concept matrix would have above identified sub components as its column. Each row corresponds to a particular cloud service provider. For instance one row may indicate Amazon Cloud while other may point to IBM and so on. Algorithm I illustrates the service concept matrix formation. If the matrix entry value is one, it indicates the presence of specific component from a particular cloud service provider. The value zero indicates the absence of specific component from a particular cloud service provider.

Input: Concept (Service, C), m sub concepts (SC) and n service providers(S).
Output: Service Concept Matrix, M_{n*m}
Begin
Step 1: $M_{n*m} = \{0\}$.
Step 2: for each S_i , i=1 to n.
for each SC_j , ,j=1 to m
if $SC_j \in S_i$, then $M_{ij}=1$
Step 3: Return matrix M_{n*m} .
End

Algorithm I: Service Concept Matrix Formation

Example of PaaS service concept matrix and SaaS service concept matrix is given in table IV. This service concept matrix is used for retrieving services based on the user request in which all the requirements of a user irrespective of the ambiguous connectives condition is matched with service concept matrix and retrieved services are added with the resultant service list.

Table IV. Service Concept matrix for PaaS and SaaS

PaaS Service Concept Matrix							SaaS Service Concept Matrix					
	Java	PHP	Ruby	...	JRuby	GO		24*7 Support	Load Balancing	...	Python	DownTime
Google App Engine	1	1	0	...	0	1	Amazon EC2	0	1	...	0	1
Heroku	1	0	1	...	0	0	Live2support	1	0	...	0	0
.							.					
.							.					
Engine Yard	0	1	1	...	1	0	Fi Cloud	0	0	...	1	0
AppFog	1	1	1	...	0	0	Five 9	0	0	...	0	1

Service Ontology Repository

This component is the knowledge base of the three categorized services which are all developed based on OWL [34]. Once the major concepts and resources are extracted from the documents by concept definer, they are passed to the service ontology repository and the OWL ontology structure is generated and updated using Jena semantic web framework [15] with the arrived concepts and their relationships.

User Request Interface

This component is the real user interactive interface. User can submit their request in natural language format through this interface. The request will be processed with PoS tagger as mentioned above and major concept of the request will be defined. In addition to the concepts identified as

noun, we are also interested in extracting the conjunctions (only 'and' and 'or') tagged as CC and prepositions (like 'with', 'besides', 'along with' and so on) tagged as JJ by the parser which are all treated as ambiguous connectives (AND, OR) and will be used while searching for cloud services. It is inferred that the concepts stated with preposition implies the 'and' conjunction. For instance the statement 'I need a virtual machine *along with* MySQL database' implies 'I need virtual machine *and* MySQL database'. This document is then passed to the concept definer to proceed with our proposed approach for service discovery. The retrieved services are then given to the user as recommended service.

Service Retriever

The service retriever selects the most appropriate cloud services related to that of user request. It selects the services by transforming the user request into modified first order horn clause logic program by horn clause logic maker. The user compatible services are then identified by parsing the service concept matrix from the first order horn clause logic program as in equation 2 and then the request will be refined by request refiner to find the appropriate services from ontology. The complete request generation process has been explained in the following sub sections.

Ambiguous Horn Clause Logic

The ambiguous requirement of user is transformed into a first order horn clause logic program. First order logic programs can effectively provide solution to the ambiguous logic. The first order logic program is slightly modified by us according to the user's requirement. The modified logic program can actually reflect the exact requirement of the user. The first order horn clause logic is of the form,

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow B \quad \text{---------- (3)}$$

Where $A_1, A_2, .., A_n$ are called premises and B is called conclusion i.e, if all of the premises are true then a conclusion B can be drawn to be true. Our modified logic is of the form,

$$A_1 \# A_2 \# \ldots \# A_n \rightarrow B \quad \text{---------- (4)}$$

Where $\# \in \{\vee, \wedge\}$ ie # corresponds to Logical connective either AND or OR, $A_1, A_2, .., A_n$ are called premises and B is called conclusion i.e, if the computation of the premises with its respective logical operations are true then a conclusion B can be drawn to be true.

Horn clause logic can be categorized [4] into

1. Rule: Both premises and the conclusion exists.
2. Fact : Premise does not exits.i.e, $\rightarrow Q$
3. Goal: Conclusion does not exist. i.e, $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow$. It is also called as objective.

Our work focuses on the evaluation of goal statement which is the request received from the user for a cloud service. Our ultimate aim lies in providing the conclusion for the goal statement as a cloud service. For instance consider the user request 'I need a virtual machine that have NoSQL, MySQL and big data analytics. The service should support java or Go or Ruby'. As explained the request will be transformed into first order logic goal statement as follows

Virtual machine $\wedge$ NoSQL $\wedge$ MySQL $\wedge$ big data analytics $\wedge$ java $\vee$ Go $\vee$ Ruby $\rightarrow$ …… (5)

Request Refiner

The number of 'or' conjunction between the concepts in the user's request document is counted during parsing. If there are n 'or' count in the user request document, then the request refiner will populate 2^{n+1}-1 queries to match with the service matrix. Algorithm II gives the population of 2^n queries for a given horn clause logic program with n OR ($\vee$) connectives.

Input: horn clause logic program (P), count of OR connectives in P (n)
Output: Set of G[2^n], Possible additional queries (goal statement) for user requirement (P).
Step 1: G[2^n] = {0}, j = 0
Step 2: for i = n down to 1
Q={x/x ∈ Convert i OR(∨) connectives in P to AND(∧) connectives in all possible ways}.
For each element, e ∈ Q
F = Deduce e to Disjunctive Normal Form (DNF).
G[j] = F
j = j +1
Step 3: Return G.

Algorithm II: Population of additional queries.

Algorithm II populates all the possible queries for the given user's request. The queries are populated in such a manner that each OR connective in the logic program, P is replaced with that of the AND connective in all possible ways. The translated logic program is then transformed into Disjunctive Normal Form (DNF) [30]. DNF is of theform, $A_1 \vee A_2 \vee \ldots \vee A_n$. A_1, A_2 ... A_n are the product of literals. DNF is a Sum of Product (SoP) expression. In DNF the OR operation is at the outermost level and each term that is ORed will be product of literals. Logic gate for DNF $(a \wedge b \wedge c) \vee (d \wedge e \wedge f)$ is shown in figure II. DNF is then converted into a standard first order horn clause logic program as in equation 1.

Algorithm III converts DNF into first order logic program by parsing the DNF with the delimiter V (OR) and eliminate the redundant elements. Hence each product term from the sum of products are extracted. The number of product terms obtained will be 2^{n+1}-1 where n is the count of OR terms in the request.

Input: DNF request set G[2^n]
Output: Set of product terms, PR.
Begin
Step 1: $\forall x \in$ G
PR = {y/ y ∈ Parsed x with the delimiter, ∨ }.
Step 2: $\forall x \in$ PR
If x is redundant eliminate it.
Step 3: Return PR.
End

Algorithm III: Conversion of DNF into horn clause logic set

Request Generator

Request generator generates the SPARQL queries and interacts with the cloud service repository to retrieve the most suited cloud services according to the user requirements. The selected list of service is then suggested to the user as the cloud services recommendation. Algorithm IV guide the generation of SPARQL request.

```
Input: Horn Clause logic program set PL[ ]; Concept C of cloud service; Service S[ ]={x/ x∈ services from
       ontology}; Resources R[ ]={x/x∈ resources from request}; Property P[ ]={x/x∈ properties from request}
Output: ServiceList[ ]
Begin
Step 1: ∀ pl ∈ PL
Step 2:          ∀ s ∈ S
Step 3:                  ∀ r ∈ R && ∀ p ∈ P
Step 4:                          If {
Step 5:                                  There exists more than one predicate in pl
                                                 Generate union subrequest for value attributes
Step 6:  SELECT ?s WHERE{
         ObjectPropertyAssertion ( :p ?c ?r)}
                                 }
Step 7:                  Else repeat step 3
Step 8:  ServiceList[ ] = ServiceList[ ] ∪ s
Step 9:  Return ServiceList[ ]
End
```

Algorithm IV: SPARQL request generation

Algorithm IV starts with concepts, services, resources and properties identified in ontology and user request. For each logic program, each identified services along with identified resources and properties, the algorithm checks for number of predicates in the logic program. If more than one predicate exists in the logic program then a union sub request is generated. The service s is selected based on the Object c; Property p and Assertion r.

EXPERIMENTAL ANALYSIS OF THE SYSTEM

Our proposed system has been implemented over the Java Agent Development Framework (JADE) [14]. Our system consists of independent components namely Stanford log linear parser, user request interface etc. These components have to be connected with each other for their communication. JADE allows independent components for peer-peer communication. Three types of cloud service ontologies were defined using protégé [25] and has been hosted using Apache Tomcat [35]. Jena semantic web framework and SPARQL [26] are used for requesting the cloud service ontology repository through ARQ. In addition to the dataset generated based on [33], set of WSDL files from [2] have also been converted to OWL files using CODE plug-in. CODE (CMU'S OWL-S Development Environment) is an Integrated Development Environment that supports the developer through the whole process from the Java generation, to the compilation of OWL-S descriptions to the deployment and registration with UDDI. It guarantees the syntactic correctness of the service description and allows the developer to use the SPIN model checker to verify correctness claims about the control flow of the OWL-S Process Model. Using these datasets, SaaS, IaaS and PaaS ontology have been generated with 32 concepts and experimented with 450 cloud service providers. Precision (P), Recall (R) [7] and F-measure (F) [11] are the three metrics that we chose to evaluate our system's (FC-S) service retrieval performance and the system is compared with (OR-S) [22] and OWLS-M4 [21]. OWLS-MX, utilizes both logic based reasoning and content based information retrieval techniques for service retrieval of the services specified in OWL-S. The three metrics are defined as

$$P=\frac{Number\ of\ Relevant\ services\ \cap Number\ of\ Retrieved\ services}{Number\ of\ Retrieved\ services} \quad \text{---------- (6)}$$

$$R=\frac{Number\ of\ Relevant\ services \cap Number\ of\ Retrived\ services}{Total\ Number\ of\ Relevant\ services} \quad \text{---------- (7)}$$

$$F = \frac{2(P*R)}{P+R} \quad \text{---------- (8)}$$

In order to evaluate the performance of our system with other two systems, 25 queries have been chosen randomly, each with various requirements. The queries are transformed into the required format as needed by these three systems. The number of requirements and ambiguity is increased in each request and the performance is measured. Figures III, IV and V provide performance measure in terms of precision, recall and F-measure respectively.

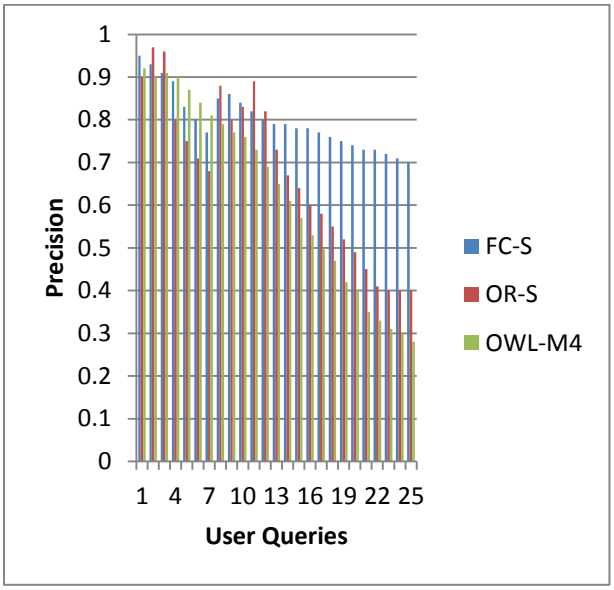

Figure III: Precision Graph

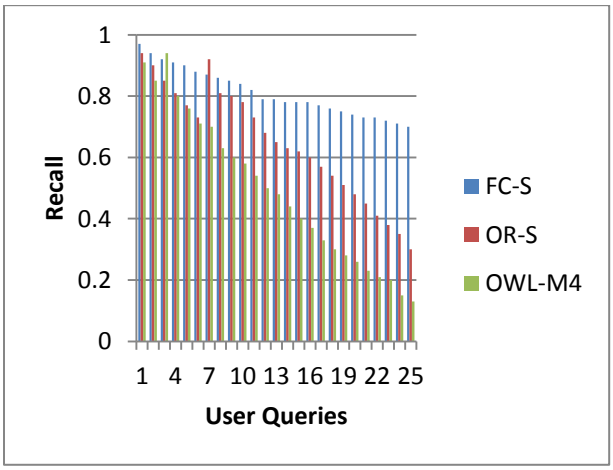

Figure IV: Recall Graph

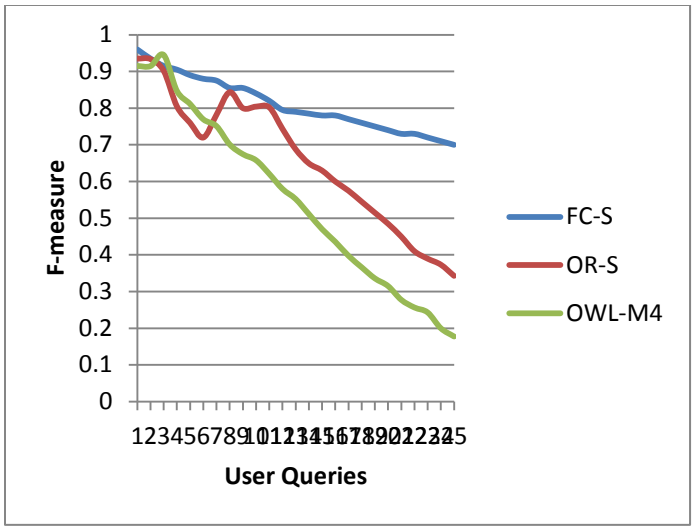

Figure V: F-measure Graph

CONCLUSION

Currently user relies on keyword based search engine for cloud service retrieval. It does not provide the correct solution as current keyword based techniques is having their own disadvantages. As there is no standard ontology structure for cloud services we had proposed and generated three ontology structures namely SaaS, IaaS and PaaS ontology. The horn clause logic is framed from these ontology structures and transformed into service concept matrix. The user queries which are in natural language are parsed to identify the user request concept. Then the user request is further refined into subqueries using ambiguous connectives (AND, OR) as specified by user and SPARQL queries are generated accordingly to retrieve the relevant services from the cloud service ontology repository. The experimental analysis of the system shows that it can provide better results than the other two methods compared under user request fuzziness. As we use service documents from Cloud service providers to generate the ontology, it may not contain entire technical attributes of the service. Further cloud service comes up with new technical terms often. Hence our ontology has to be updated whenever the changes arrive from cloud service providers. Thus we are in need of an ontology which supports new services. In near future we are in a plan to develop the cloud service discovery methodology which takes the previous user experiences with the cloud service providers for recommending the cloud services according to user requirements.

REFERENCES

[1]. Aaron E. Walsh, “Uddi, Soap, and Wsdl: The Web Services Specification Reference Book”, Prentice Hall Professional Technical Reference, 2002

[2]. Al-Masri, E., and Mahmooud, Q.H.: Investigating Web Services on the World Wide Web, 17th International Conference on World Wide Web (WWW), Beijing, 2008, 795-804

[3]. Andrés García García, Ignacio Blanquer, “Cloud Services Representation using SLA Composition”, Journal of Grid Computing, 2014

[4]. Antoniou, Grigoris, and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[5]. Battle, Steve, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer et al. "Semantic web services framework (SWSF) overview." World Wide Web Consortium, Member Submission SUBM-SWSF-20050909 (2005).

[6]. Chen, Wuhui, and Incheon Paik. "Improving efficiency of service discovery using Linked data-based service publication." *Information Systems Frontiers* 15(4), 2013, 613-625

[7]. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, "Introduction to Information Retrieval", Cambridge University Press, 2009.
[8]. Dieter Fensel, Frank van Harmelen, Vrije Universiteit, Amsterdam, "An Ontology Infrastructure for the Semantic Web", IEEE Transactions on Knowledge and Data Engineering, 19(2), 2001, 261-272
[9]. G.A. Miller, WordNet: a lexical database for English, Commun. ACM 38(11), 1995, 39–41.
[10]. Garg, Saurabh Kumar, Steve Versteeg, and Rajkumar Buyya. "A framework for ranking of cloud computing services." Future Generation Computer Systems 29(4), 2013, 1012-1023.
[11]. H. Do, S. Melnik, E. Rahm, Comparison of schema matching evaluations, in:Proc. Workshop on Web Databases, German Informatics Society, Bonn, 2, 2002.
[12]. J. D. Bruijn, C. Bussler, J. Domingue et al., "Web Service Modeling Ontology (WSMO)," http://www.w3.org/Submission/WSMO
[13]. Jaccard, Paul, "The distribution of the flora in the alpine zone", New Phytologist 11, 1912, 37–50
[14]. JADE, Java Agent Development Framework, http://jade.tilab.com/.
[15]. Jena - A Semantic Web Framework for Java, http://incubator.apache.org/jena/
[16]. Jones KS, "A statistical interpretation of term specificity and its application in retrieval", Journal of Documentation 28(1), 1972, 11–21.
[17]. Kaijun Ren; Nong Xiao; Jinjun Chen, "Building Quick Service Request List Using WordNet and Multiple Heterogeneous Ontologies toward More Realistic Service Composition," *Services Computing, IEEE Transactions on* , 4(3), 2011, 216,229.
[18]. Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network", in Proceedings of HLT-NAACL 2003, 252-259.
[19]. Li, Xitong, et al. "A petri net approach to analyzing behavioral compatibility and similarity of web services." *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 41(3), 2011, 510-521
[20]. Lianyong Qi, Wanchun Dou, Jinjun Chen, "Weighted principal component analysis-based service selection method for multimedia services in cloud", Computing, 2014.
[21]. M. Klusch, B. Fries, K. P. Sycara, OWLS-MX: a hybrid Semantic Web service matchmaker for OWL-S services, Journal of Web Semantics 7(2), 2009, 121-133.
[22]. Miguel Ángel Rodríguez-García, Rafael Valencia-García, Francisco García-Sánchez, J. Javier Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, Knowledge-Based Systems, 56, 2014, 15-25
[23]. Miguel Ángel Rodríguez-García, Rafael Valencia-García, Francisco García-Sánchez, J. Javier Samper-Zapater, Creating a semantically-enhanced cloud services environment through ontology evolution, Future Generation Computer Systems, Volume 32, 2014, 295-306.
[24]. Paliwal, Aabhas V., Shafiq, B., Vaidya, J., Xiong, H., & Adam, N. "Semantics-based automated service discovery." *Services Computing, IEEE Transactions on* 5(2), 2012, 260-275.
[25]. Prote´ge´, http://protege.stanford.edu/
[26]. Prudhommeaux, E., and Seaborne, A. (2008). SPARQL request language for RDF, W3C recommendation, W3C, available at http://www.w3.org/TR/2008/REC-rdf-sparql-request-20080115
[27]. Rittinghouse, John W., and James F. Ransome., "Cloud computing: implementation, management, and security", CRC press, 2009.
[28]. Segev, A; Sheng, Q.Z., "Bootstrapping Ontologies for Web Services," *Services Computing, IEEE Transactions on*, 5(1), 2012, 33, 44.
[29]. Srinivasan, Naveen, Massimo Paolucci, and Katia Sycara. "CODE: a development environment for OWL-S Web services." *Robotics Institute, Carnegie Mellon University* 2005.
[30]. Weisstein, Eric W. "Disjunctive Normal Form." From MathWorld—A Wolfram Web Resource.http://mathworld.wolfram.com/DisjunctiveNormalForm.html
[31]. World Wide Web Consortium, "OWL-S: Semantic Markup for Web Services", World Wide Web Consortium (W3C) Member Submission, 2004, available at http://www.w3.org/submission/owl-s
[32]. Yu, Qi. "CloudRec: a framework for personalized service Recommendation in the Cloud." *Knowledge and Information Systems* 2014, 1-27
[33]. http://www.cloudbook.net/directory/product-services/cloud-computing-directory

[34]. http://www.w3c.org/standards/techs/owl
[35]. http://tomcat.apache.org/

Currently working as Assistant Professor, Computer Science and Engineering Department in S.Veerasamy Chettiar College of Engineering and Technology, Puliyangudi. Having nearly 10 years of teaching experience and having familiarity in networking field.

Currently working as a Head, Master of Computer Applications Department in S.Veerasamy Chettiar College of Engineering and Technology, Puliyangudi. Having nearly 9 years of teaching experience with vast knowledge in networking field.