Enhanced Cloud Service Discovery for Naïve users with Ontology based Representation

Viji Rajendran V[1] and Swamynathan S[2]
[1] Research Scholar, Department of Information Science and Technology,
CEG Campus, Anna University,
Chennai 600025, India
[E-mail: vijirajv@gmail.com]
[2] Associate Professor, Department of Information Science and Technology,
CEG Campus, Anna University,
Chennai 600025, India
[E-mail: swamyns@annauniv.edu]
*Corresponding author: Viji Rajendran.V

Received July 23, 2015; revised October 5, 2015; accepted October 27, 2015; published January 31, 2016

Abstract

Service discovery is one of the major challenges in cloud computing environment with a large number of service providers and heterogeneous services. Non-uniform naming conventions, varied types and features of services make cloud service discovery a grueling problem. With the proliferation of cloud services, it has been laborious to find services, especially from Internet-based service repositories. To address this issue, services are crawled and clustered according to their similarity. The clustered services are maintained as a catalogue in which the data published on the cloud provider's website are stored in a standard format. As there is no standard specification and a description language for cloud services, new efficient and intelligent mechanisms to discover cloud services are strongly required and desired. This paper also proposes a key-value representation to describe cloud services in a formal way and to facilitate matching between offered services and demand. Since naïve users prefer to have a query in natural language, semantic approaches are used to close the gap between the ambiguous user requirements and the service specifications. Experimental evaluation measured in terms of precision and recall of retrieved services shows that the proposed approach outperforms existing methods.

Keywords: Natural language query, cloud services, ontology, clustering

1. Introduction

Cloud computing is a new trend of distributed computing where scalable computing resources are exposed as services over the Internet. The perception of cloud computing has seen a significant growth as it provides "everything as a service". Most companies farm out large parts of their resource requirements to cloud service providers owing to the ample available services offered. Cloud service is virtually any business or consumer service that is delivered and consumed over the World Wide Web in real time [1]. Examples of cloud services include web-based email services, online data storage and backup solutions, document management and collaboration services, database processing management and more. One of the demanding issues in cloud computing is the selection of cloud service and the service provider over the web. With the boom of the Internet, cloud service providers offer lots of services to end users in various formats. Unlike web services, cloud services do not have unified standard description.

Due to the various service descriptions, non-uniform naming conventions, heterogeneous types and features of services, it is difficult for the user to find the cloud services that fulfill their requirements [2]. Though providers like Amazon S3 and EC2 describe their services using Web Service Definition Language (WSDL), it could not solely meet the requirements of cloud service description. Furthermore, WSDL fails to cover the unique features of cloud services. Efforts like USDL (Unified Service Description Language) [3] and TOSCA (Topology and Orchestration Specification for Cloud Applications) [4] have been made to provide standards for cloud computing. But their levels of interoperability and the degree to which they can be integrated are faltering. Generally, cloud services are described in plain text in the service provider's website. Adoption of a standardized format to describe the requirements and service offerings will bolster the trade of cloud services. However, research work on cloud services description language is still in its early stages.

Contemporary service discovery techniques may not be suitable to be used in the Internet-scale environment [2]. Hence, a marketplace that contains a catalogue for all the cloud services under a single URL is infeasible. According to Gartner [5], strong demands are anticipated for all types of cloud service offerings even if there is wide variation between cloud services market sub-segments. With the growth of public cloud offerings, a cloud service registry is urgently required to connect the cloud providers and users [6]. In order to speed up the process of service discovery, catalogues of existing services along with their service descriptions and pricing concerns have to be created. To realize this, the existing services are crawled using cloud service crawler. Cloud service crawler [7] is a program which fetches as many relevant services as possible for the specific users.

Cloud services are dynamic entities that evolve at rapid rates. New services are added and old ones are removed or modified over time. As a result, the crawling should be done periodically in order to keep the cloud services repository up-to-date. Multi-threaded priority based crawler [8] is used to crawl cloud specifications and pricing details from the provider's website. A repository of cloud services is created by categorizing the crawled services based on the functionalities of the cloud services by using a clustering algorithm. When services are grouped into different clusters, services within the same cluster provide similar matches with respect to a service request. If the services are classified well, searching through predefined groups may provide better results for service discovery [9]. This paper suggests clustering of services based on similarity and search service in the closest similar group, which has fewer

services. Its main benefit is that it reduces the time needed for discovering services.

To have an effective search results, the service consumer must specify the outputs he/she requires and the inputs he/she can provide for the service. But most of the service requesters do not have a clear idea of their request, and hence, cannot convey their requests accurately. Users who are technically illiterate depend on Natural Language Queries (NLQ) for retrieving information about cloud services. NLQ to cloud services will be a key technology to guarantee effortless access to services. NLQ processing facilitates conveying without resorting to memorization of complex procedures for discovering services. This involves resolving the complex problem of identifying relevant services given an ambiguous natural language query. There may be a mismatch between the vocabulary used to specify the user request and that used to describe service descriptions. This leads to poor service discovery and hence, to low precision and recall [10].

Consequently, discovering existing cloud services according to user's request is a tricky task. Hence, to mitigate the burden of users, a novel approach is proposed in this paper to convert natural language query to a standard format. Thus, end users must be able to discover services based on a query written in natural language. The limitations of the conventional discovery process and the difficulty of matching between user demands and service descriptions advertised by providers can be overcome through ontologies and semantic technologies. As an initial step toward this goal, the potential terms in the natural language query are identified and enriched with further relevant information to route them to suitable services. This can be realized by integrating semantics to service through ontologies. Hence, this paper recommends an enhanced cloud service discovery mechanism based on ontology for naïve user requests. The discovery mechanism creates a bridge between user requirements written in natural language and service descriptions on the provider's site, using semantically enhanced standard format.

The rest of the paper is structured as follows. The related research work is described in Section 2. Problem definition and uniform representation format for services, Key – Value (KV) representation, are discussed in Section 3. The architecture of Semantic-based Service Discovery with Clustering for Naïve users is detailed in Section 4. Results are analyzed and efficiency of the system is discussed in Section 5 and Section 7 presents conclusions and future directions.

2. Related Work

Cloud adoption is, without a doubt, on the rise. Recent IDC report [11] predict that cloud services will remain a hotbed of activity in 2015 with $118 billion in the global cloud market. Sun et.al., conducted an exhaustive state-of-the-art survey of existing cloud service selection approaches to evaluate and compare current cloud service publication platforms, modeling languages and ranking methods [12]. As cloud computing becomes more popular, the number of services offered by providers increases tremendously day by day. In the previous work [8], 6743 unique cloud services were crawled from the web. Among them, 1922 services are infrastructure related services including storage, backup, virtualization and networking and 3814 are software services like accounting, business management and security as a service. 517 services focus on PaaS services. Rest 490 services offer more than one cloud service model. Major players such as Microsoft, Amazon, Google, AT&T Inc, IBM, Oracle Corp and Dell offer more than one cloud service.

While cloud services offer evident benefits in terms of cost reduction and increased performance, searching and selecting them is in flux. To find a set of ranked services, Service

Provider Search Engine (SPSE) was recommended which supports user personalization in service selection [13]. It uses SAW-based service searching and scheduling algorithm to locate the suitable service by considering the user's multiple QoS (Quality of Service) needs. But a number of cloud services exist without proper semantic descriptions. Providers use a different vocabulary to describe similar services. Due to this many services which are most relevant to the user request is left undiscoverable. Hence, discovering cloud services have more hurdles compared with web service discovery from WSDL files and UDDI registry [14]. Cloud service description languages are restricted to a certain cloud layer and support only the specification of some particular aspects of a cloud service [15]. Open Virtual Format is a standard for denoting the packaging and distribution information of an IaaS [16], and Open Cloud Computing Interface is a standard for specifying the IaaS management API.

A few proprietary specification languages maintained by the commercial cloud vendors, such as adhoc XML format for Microsoft Azure, exist in PaaS layer. Standards in Web services such as WSDL can be reused for specifying services in SaaS layer. TOSCA [4] supports provisioning of SaaS applications. These languages do not help to publish, query, and compose cloud services across providers. The Blueprint Specification Language (BSL) [15] provides a specification language for cloud service providers to specify their services. But it allows for specifying only certain information sets of a cloud service and lacks support to Service-level Agreement policy, the pricing policy, and compensation policy of a cloud service. Hence, there is a great demand for a uniform specification language. Discovering cloud services based on user request from internet repository is a time consuming process. Hence, to find the best suitable service for a certain demand, service registries has been established as fundamental systems between providers and consumers.

The search facility is limited to the service name and/or category in service directories like CloudBook [17]. Hence, to create a repository of cloud services, focused crawlers are required to gather cloud service descriptions from the provider's website. Algorithms based on clusters can extract information from large volumes of data by dividing it into different groups based on certain similarities [9]. The access time can be reduced by bringing in efficient clustering and matchmaking algorithms. Operation parameter clustering techniques are also used in Seekda! [18], where information about services is gathered from various sources like Web pages and blogs. Some researchers propose using semantic approaches for cloud service discovery. The Mosaic ontology [19] developed in OWL is used for semantic retrieval and composition of cloud services in the Mosaic project. Semantic service annotation enables end users to search existing web services using keywords [20].

Sim [21] propose an Agent based Cloud Service Discovery System (CSDS) which interact with ontology to find cloud services that are closer to users' requirements. Unified business service and cloud ontology [22] captures the required business services in an organization and provides a mapping between business functions and the offered services in the cloud landscape. The ontology serves as a repository for cloud services and providers. Users can query the ontology to discover the services that match their requirements. In most of the ontology based discovery systems, the query is expressed in SPARQL language [23], confining the use of ontology to experienced users only. Hence, a system that accepts a natural language query is greatly demanded. In Pythia, natural language input is converted into a formal query by means of a linguistic analysis driven by an ontology-based grammar [24]. It is also observed that burst path losses have become a pressing concern in real time video conferencing services [34-36]. Hence, the efficiency of service discovery becomes a significant issue for cloud computing enabled applications.

3. Background

3.1 Problem Definition

Naïve users rely on conventional search engines in pursuing cloud services. There are no specialized search engines for discovering cloud services. Traditional search hunt offers immeasurable solutions. Moreover, searching the web is a lingering process. To speed up the service discovery process, an up-to-date repository of services is essential. This can be accomplished with a multi-threaded priority based incremental crawler and similarity based clustering. Services are crawled from the web and these *M* services are clustered into *N* categories (i.e. C_1, C_2… C_N) based on their similarity by a clustering algorithm. The selection order of cloud services does not influence the accuracy of the clustering process. However, users may not know which cloud services to use, and they need to keep perusing the service description before selecting an appropriate one. Henceforth, the search engines are not suitable for finding services that address the user's prerequisites.

Most of the cloud providers describe the services in their websites in various formats. Hence, the complexity involved in matching user demand with the service description is very high. One of the critical issues addressed in this paper is the formal representation of services in a standard format. Service description is an abstract specification of operation it supports and can be expressed as a quadruple $CS_{des} = (S_{name}, S_{id}, S_A, S_D)$, where S_{name} represents the title of the service, S_{id} is the unique identification of the service, $S_A = \{A_1, A_2,…., A_i\}$ are the attributes of the service including the functional and non-functional specifications and $S_D = \{D_1, D_2,…, D_i\}$ are the corresponding data values of the attributes. The service request is a description of user requirements. The request in natural language is processed and expressed as a triple $CS_{req} = (S_K, S_V, S_C)$, where $S_K = \{K_1, K_2,…, K_n\}$ represents the functional and non-functional requirements, $S_V = \{V_1, V_2,…, V_n\}$ is the user's expectation on the attributes and S_C is a set of QoS constraints.

Further, the difference in the vocabulary used by the providers and users makes the searching process more complex. Hence, semantics needs to be integrated into the services through ontology. Ontology-based service matchmaking method is used to find the most suitable services (M_{CS}) from a cluster of services whose attribute values are closest to the service requested CS_{req}. The absence of technical skills of naïve users demands an efficient strategy for ranking M_{CS}. Service ranking outputs a ranked list using a scoring mechanism that depends on five factors and can be expressed formally as a quintuple $CS_{rank} = (SS, UP, UF, AV, SC)$, $\forall_{service} \in M_{CS}$. Here SS, UP, UF, AV and SC corresponds to similarity score, user preference, user feedback, availability and service cost respectively. The proposed system addresses the problem of naïve users being not able to discover pertinent cloud services that meet their requirements. The system requires little human interaction and can be accessed via a web based user interface.

3.2 KV Representation

Although semantic technology is the main research direction in service matchmaking, less research has been focused on the semantic representation of cloud services [26]. A precise service specification model based on ontology is vital when developing specification models for cloud services. In general, providers describe information about the cloud services in their web pages in natural language. The pricing rules and service offerings are frequently modified and published on the cloud providers' websites. The descriptors in these web pages have no

proper machine interpretable structure and henceforth, cannot be utilized to process information about cloud services automatically. Consequently, many questions arise about how to manage the inconsistencies in knowledge representation and standardization of the description. Hence, to formalize this issue, Key-Value (KV) representation is used to store cloud service description, where key stands for an attribute of a service. A separate KV template is used for a different category of services. Here the key can take only string form whereas the value can be categorical (string or Boolean) or numerical (integer or float). For example, the KV representation for storage services is shown in **Table 1**.

Table 1. Storage service specification in K-V representation

<service_type = “storage”>: { "title" : "Storage services",
"description": “Specifications of storage services”,
"type": "object", An ordered collection of key/value pairs.
["properties": { The property specifier holds the description of the key/value pairs.
"Provider": {"type": "string", "category": "MANDATORY"},
"display_name": {"type": "string", "category": "OPTIONAL"},
"url" : {"type": "string", "category": " MANDATORY"},
"billing_time": {"type": "string", "category": "OPTIONAL"},
"price": {"type": "float", "minimum": 0, "exclusiveMin": true, "category": " MANDATORY"},
"storage_capacity" : {"type" : "integer", "category" : "MANDATORY"},
"free_plan": {"type": "integer", "category": "OPTIONAL"},
"unlimited_plan": {"type": "boolean", "category": "OPTIONAL"},
"data_center": {"type": "string", "category": "OPTIONAL"},
"hasLocation": {"type": "string", "category": "OPTIONAL"},
"api": {"type": "string", "category": "OPTIONAL"},
"hasBlock_level_replication": {"type": "boolean", "category": "OPTIONAL"},
"hasRedundancy": {"type": "integer", "category": "OPTIONAL"},
"hasDesktop_file_sync": {"type": "boolean", "category": "OPTIONAL"},
"hasAvailability": {"type": "float", "category": "OPTIONAL"},
"hasRobustness": {"type": "float", "category": "OPTIONAL"},
"os_platform": {"type": "string", "category": "OPTIONAL"}]}}

4. System Architecture

A convenient natural language interface is essential to discover cloud services. The user may pose imperative questions, or wh-questions. The user can express a query using a set of keywords, simple or compound sentences. The proposed system translates the natural language question Q_N to a structured formal query Q_F that focuses on the concept articulated by Q_N. The architecture of Semantic based Service Discovery with Clustering for Naïve users (SSDCN) is shown in **Fig. 1**.

SSDCN is divided into two sections query rewriting and service matching. In query rewriting, the user request is parsed and the parsed phrases are mapped to the cloud ontology concepts. In service matching, the mapping is performed between the user request and categorized services which are stored as a catalogue. In order to fasten the process of service discovery, a catalogue of services is created by making use of cloud service crawler and clustering algorithm. The catalogue contains a set of cloud services that an end user can request, including pricing and the terms and conditions for service provisioning. The process of service discovery is to seek suitable service which could satisfy the user’s requirements from a pool of services. Reasoning based on ontology increases the chance of finding relevant

alternatives of a service [21]. Hence, in SSDCN, intelligent service discovery platform based on ontology is employed for finding suitable services precisely.

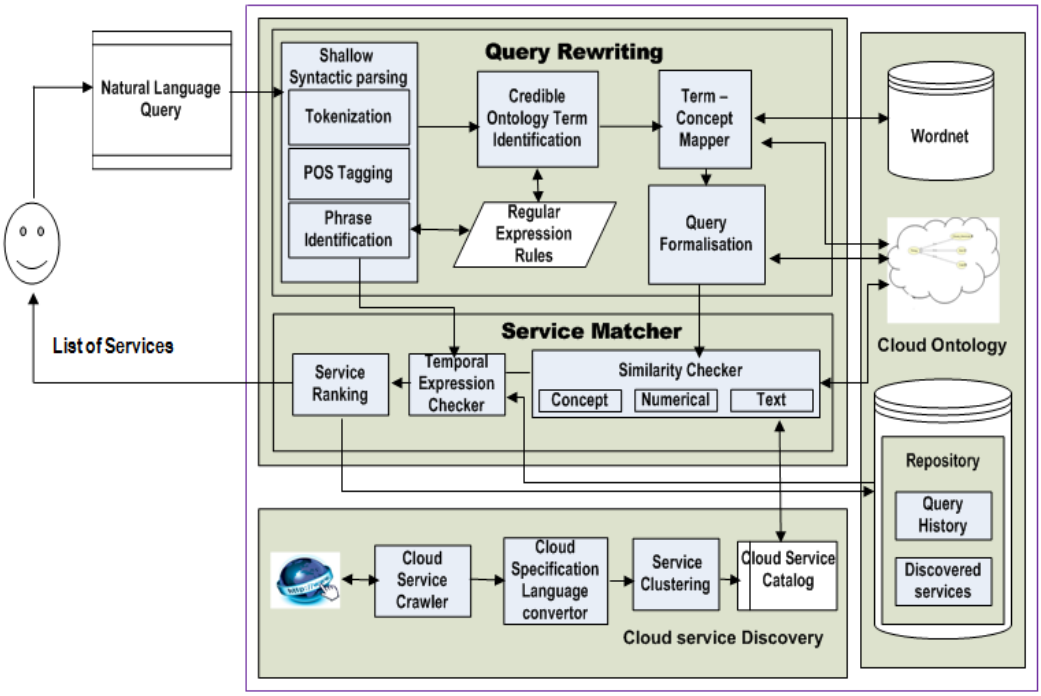

Fig. 1. Architecture of SSDCN

4.1 Cloud Ontology

Ontology is defined as a formal and explicit specification of a shared conceptualization [25]. An ontology O is a quintuple, $O = (O_C, O_P, O_I, O_V, O_A)$ where O_C, O_P, O_I, O_V and O_A are the sets of classes, properties, individuals, property values and constraint axioms respectively. To facilitate machine readable description of cloud services, a cataloging of services has been accomplished through Web Ontology Language (OWL) ontology, which recognizes the service types and their specifications. Ontology specifies how a concept is related to linguistic structures such as regular expressions and lexicons. Discovery of services represented in a heterogeneous format is a complex task. However, it has been proved that ontology has been useful for semantic annotation and discovery of cloud services [19] [21] [27] [28]. Hence, it can be used by Natural Language Processing (NLP) to improve expressiveness and to resolve the ambiguity of NL queries. Ontologies are employed to map the concepts from different providers to a unique formal representation. These ontologies are also used to establish mappings at the time of query rewriting. Thus, ontology plays a critical role in query reformulation process.

4.2 Creation of cloud services catalogue

Discovering and ranking services from the web may not be proper, particularly from the performance point of view. To create a catalogue of cloud services, multi-threaded priority based crawler is employed to crawl the service descriptions from the provider's site. Good choices of relevant seed URL certainly influence the results of a crawler. **Table 2** shows the

initial seed URL used for crawling.

Grouping of services based on similarity of service description substantially reduces the number of comparisons required for service matching. Clustering brings identical services together and this scale well even if the number of services increases exponentially. However, clustering algorithms work effectively either on pure numeric data or of pure categorical data, most of them perform poorly on mixed data types. Cloud services have both numeric and categorical attributes. Hence, it is difficult for applying conventional clustering algorithm directly into these kinds of data. Based on the type of attribute value, the attributes of the service $s_i = (a_1, a_2, \ldots, a_m)$ are split into numerical (s^{num}) and categorical (s^{cat}) attributes. A distance measure is needed for grouping services into clusters.

Table 2. Initial seed URL list

http://www.dmoz.org/Computers/Internet/Cloud_Computing/Service_Providers/
http://www.cloudreviews.com/
http://www.cloudservicemarket.info/services/servicesBrowse.aspx
http://cloudshowplace.com
http://www.cloudxl.com/category
http://talkincloud.com/tc100
http://atechjourney.com/list-of-free-cloud-storage-services.html/
http://compixels.com/2303/list-of-top-free-cloud-based-services
http://en.wikipedia.org/wiki/Category:Cloud_computing_providers

4.2.1 Similarity Calculation for numerical attributes

Consider two services $S_1 = (X, A)$ and $S_2 = (Y, B)$. Attributes of S_1 are split into numerical $S_1^{num} = \{x_1, x_2, \ldots, x_n\}$ and categorical $S_1^{cat} = \{a_1, a_2, \ldots, a_m\}$. Similarly S_2 is split into $S_2^{num} = \{y_1, y_2, \ldots, y_n\}$ and $S_2^{cat} = \{b_1, b_2, \ldots, b_n\}$. For numerical attributes, Pearson Correlation Coefficient is used to determine the similarity between feature vectors. This metric is measured from -1 to +1 and it measures how highly correlated are two services. Pearson Correlation Coefficient of 1 indicates that the data attributes are perfectly correlated and a score of -1 means that the data attributes are not correlated. In the mathematical form, the score can be described as:

$$r_{xy} = \frac{n\sum_i x_i y_i - \sum_i x_i \sum_i y_i}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}} \quad (1)$$

In this equation, *(x, y)* refers to the data objects and *'n'* is the total number of attributes. A distance metric for two variables *'x'* and *'y'* known as Pearson's distance can be defined from their correlation coefficient as

$$Sim(x, y)^{num} = 1 - r_{xy} \quad (2)$$

and lies between 0 (when correlation coefficient is +1, i.e. the two samples are most similar) and 2 (when correlation coefficient is -1). For instance, for the data provided in **Table 3**, the numerical attribute corresponds to *{hasHourlyPrice, hasVirtualCores, hasComputeUnits, hasMemory, hasDiskSpace}*. Here, X= {0.15, 2, 4, 7.5, 840} and Y= {0.11, 2, 2, 7.5, 738}, and hence, correlation coefficient r_{xy} = 0.99999636298309 and Sim (x, y) = 3.64 *10^{-6}, which shows that there is a strong positive correlation among services S_1 and S_2.

4.2.2 Similarity Calculation for categorical attributes

For categorical attributes, the Jaccard/Tanimoto Coefficient [29] is employed for measuring

similarity. It uses the ratio of the intersecting set to the union set as the measure of similarity, and can be mathematically represented as:

$$T(A,B) = \frac{M_c}{M_a + M_b - M_c} \tag{3}$$

Table 3. Two services S_1 and S_2 in KV representation

<service_type = “compute”>:{ "title": "compute services", "description":“Specifications of compute services”, "id": "C070", ["properties": {"Provider": "Amazon", "display_name": "AWS", "billing_time": "hourly", "hasHourlyPrice": "$0.15", "hasInstance Type": "m1.large", "hasVirtualCores": "2", "hasCompute Units": "4", "hasMemory": "7.5 GB", "hasDiskSpace": "840 GB", "supportOS": "linux", "supportDatabase": "MySQL", "hasDataCentreAt": "Europe" }]}	<service_type = “compute”>:{ "title": "compute services", "description": “Specifications of compute services”, "id": "C095", ["properties": {"Provider" : "Joyent", "display_name": "Joyent medium", "billing_time": "hourly", "hasHourlyPrice": "$0.11", "hasInstance Type": "medium", "hasVirtualCores": "2", "hasCompute Units": "2", "hasMemory": "7.5 GB", "hasDiskSpace": "738 GB", "supportOS": "linux", "supportDatabase": "MongoDB", "hasDataCentreAt": "Europe"}]}

In this equation, *‘M’* represents the number of categorical attributes in *(A, B)* and *‘c’* in this case is the intersection set.

$$Sim(A,B)^{cat} = 1 - T(A,B) \tag{4}$$

From the data provided in **Table 3**, *A = {“Amazon”,“AWS”,“hourly”,“m1.large”, “Linux”, “MySQL”,“Europe”}* and *B = {“Joyent”,“Joyent medium”,“hourly”,“medium”, “Linux”, “MongoDB”,“Europe”}*. Here, $M_a = 7$, $M_b = 7$, $M_c = 3$ and hence, $T(A, B) = 0.27$ and $Sim(A, B)$ is 0.73. The value of $Sim(A, B)^{cat}$ is in the range of [0,1]. When the value of $Sim(A, B)^{cat}$ is larger, the correlation between *A* and *B* is lesser. Consider the services has *m* attributes, with m_c categorical and m_n numeric attributes, where $m_c + m_n = m$. Also, assume that each attribute has equal importance, i.e. all attributes have equal weight and no biased treatment of any attributes. The similarity between two mixed-type services S_1 and S_2 can be represented as:

$$Sim(S_1,S_2) = \frac{m_n}{2m} Sim(X,Y)^{num} + \frac{m_c}{m} Sim(A,B)^{cat} \tag{5}$$

Since the Pearson coefficient is used as a similarity metric for numerical data and its range is [0,2], $Sim(X, Y)^{num}$ is divided by 2.

Algorithm 1: Finding similarities of services

Input: Set of services ,S = {s_1, s_2, s_3,…., s_n}

Output: Similarity between service

Steps:

1. Split $s_i = (a_1, a_2, ..., a_m)$ into $s_i = (s^{num}, s^{cat})$, where s^{num} and s^{cat} represent the numerical and categorical attributes of s_i.
2. Find distance metric for the numerical attribute using Pearson’s distance formula as explained in equation 2.
3. Employ Tanimoto coefficient for calculating similarity of categorical attributes (explained in equation 4)
4. Use equation 5 to find the similarity between a pair of services.

4.2.3 Service clustering

An improved k-means approach is used for clustering of cloud services so that the most similar services are grouped together. The steps for implementing the modified K-means algorithm are:

1. Generate K cluster centers as per the algorithm 2.
2. Compute the proximity of each service to each cluster centre and assign each service to the nearest cluster centre.
3. Re-compute the cluster centers by taking the mean of the member services in each cluster; for the categorical attribute, most repeated values in the cluster is taken as the attribute mean.
4. Stop if there is no or minimal change in the cluster centers; else go back to step 2.

Thus, the services are clustered based on their similarity and a catalogue is maintained for the further discovery process. Two major modules of SSDCN are query rewriting and service matching.

Algorithm 2: Finding initial centroids
Input: $S = \{s_1, s_2, s_3, \ldots\ldots, s_n\}$ // set of services
k = number of desired clusters;
Output: Set initial centroids K.
Steps:
1. Calculate the Weighted Score (WS) of each service $S_n = \{a_1, a_2, a_3, \ldots., a_m\}$
   Weighted Score (WS) of $S_n = \sum_{i=1}^{m} \frac{a_i}{a_i(\max)}$, where a = the attribute's value, m = number of attributes and $a_i(max)$=Maximum value of attribute a_i
   //numerical values are assigned to each categorical attribute based on their number of occurrences, most repeated value is given the higher number
2. Sort the services in ascending order based on weighted score
3. Divide the datasets into k subsets
4. Calculate the average of each group
5. Select services as initial centroids whose weighted score is closest to the average value of groups

4.3 Query rewriting

Query rewriting operates by carrying out the following steps as 1) Read in a natural language query and split it into words/tokens. 2) Annotate each token with POS tags. 3) Identify the potential concepts in the query by applying regular expression chunk parsers. 4) Map the extracted phrases with ontology concepts

4.3.1 Shallow syntactic parsing

Query rewriting starts with shallow syntactic processing which consists of tokenization and Part Of Speech (POS) tagging, chunking and extraction of key concepts. Natural Language Toolkit (NLTK) with Python is used in the preprocessing. The Natural Language Query (NLQ) posed by the user is a sequence of tokens, $Q_N = \{T0, T1, \ldots, Tn\}$. The tagger with the necessary linguistic knowledge reads the NLQ and assigns part of speech category to each word in an input query. The tag set is based on the *Penn Treebank* Tagging Guidelines [30] with 36 *POS tags*. For example, the query "*List the providers who offer free storage services*" can be tagged as follows: *[List/NNP] [the/DT] [providers/NNS] [who/WP] [offer/NN] [free/JJ] [storage/NN] [services/NNS]*

After having POS for each word, the query is given to the phrase identification parser which analyzes the syntax of given NLQ and finds the relation between words. A phrase is a subsequence of tokens which exemplifies the needed information. The parser uses a context-free grammar to group the word as constituents like a noun phrase, verb phrase, adjectival phrase and prepositional phrase. Rule-based chunker receives a sequence of tagged words and then divides the NLQ into relevant phrases. Chunk parsing extracts syntactically associated fragments, in agreement with regular expression grammar, which define well-grounded sequences of POS tags. NN.* could be one or more nouns where NN: noun, singular or mass; NNS: noun, plural; NNP: proper noun, singular; or NNPS: proper noun, plural. CP is a comparative phrase, could be either adjective comparative or adjective superlative (cheaper, most popular). IN could be a preposition or subordinating conjunction (in, of, like, after, that). CD could be any cardinal number (10, two). The output of the parser for the query "*Name the providers who offer ERP services with a cost less than $10 per month and have data centers in Asia*" is shown below.

Rules used to identify a NP	Output
NBAR: {<NN.*|JJ>*<NN.*>} {<DT|PRP$>?<JJ>*<NN.*>*} CP: {<JJR|JJS>} NP: <NBAR><CP><IN><CD><IN><NBAR>} {<CP>?<NBAR><IN>+<NBAR>+} {<NBAR><CD><NBAR>?} {<NBAR>}	(S (NP (NBAR Name/NN)) the/DT (NP (NBAR providers/NNS)) who/WP (NP (NBAR offer/NN ERP/NN services/NNS)) with/IN (NP (NBAR cost/NN) (CP less/JJR) than/IN 10/CD per/IN (NBAR month/NN)) and/CC have/VBP (NP (NBAR datacenters/NN) in/IN (NBAR Asia/NNP)))

4.3.2 Identification of Credible Ontological terms

The major goal of this component is to identify a syntactic structure of a NLQ and adapt it to a formal semantic representation. Heuristic rules are used to identify Credible Ontological terms (COT) from natural language query. COTs refer to query terms that could be associated with ontology concepts. Each NP (noun phrase) is identified as a base to COT. To identify the COT of noun phrases, following rules are used:

• For phrases with of the form: "NN.+" the COT is the last noun keyword and all other keywords are treated as attributes/modifiers.

• For the phrases of the form "<CP>? <NBAR> <IN> <NBAR>", the COT is the last noun keyword before the preposition. For example, "datacenter" in "List the largest datacenter in Asia".

A manually built synonym table is used to identify common abbreviations and their original forms in the noun phrases. For example, the original form of *'ERP'* is 'Enterprise Resource Planning' and *'OS'* is 'Operating System'. NLTK Stopwords Corpus is used to remove default English stopword from these phrases.

4.3.3 Term Concept mapping and formal specification

This module maps query phrases with ontology concepts via WordNet [31] synsets (synonym sets) by assigning the structured meanings of the ontology to plain text. Ontologies are used as background knowledge to capture the semantic features of the key phrases in the query. In

order to have the best opportunity to discover the requested service, the credible query terms are expanded with the concepts from ontologies. The tokens in the query are linked to ontological concepts. COTs of noun phrases identify the class concepts in the ontology. The object type property identifier is extracted using a verb phrase. The datatype property identifier is extracted using the regular expression <CD> + <NN.* > + e.g., *'five VMs'*.

The term-concept mapper retrieves the concepts from the ontology repository and matches them with the query terms. Initially, the query words are matched directly with the subjects, predicates, and objects of the ontology models. If a direct mapping fails, the stripped mapping is done between query words and the ontology concepts. If no match is found, two possible relations of interest are taken; the synonyms or equivalent terms and hypernyms where ontology concept is more general than WordNet synset of query terms. WordNet is used to identify the synonyms of the concepts and relationships in the ontology. This will help to query a service even if the user is unaware of the exact terminology used in the service description. Once the corresponding ontology terms are identified, the query is rewritten to KV representation.

Algorithm 3: Concept Mapping
Input: COT Terms, Ontology
Output: KV representation of query
T: {t | t in COT} // list of COTs in NLQ
for i = 1 to |T| //loop through all indexed tokens in NLQ
// Loop through ontology O, and create a list with all concepts c,
// where the stem of the concept name or the stem of a synonym of the concept equals the stem of the
// COT, and where the POS equals the POS of the COT
L :={c| c ε O}
for m = 1 to |L|
if stem(t) = stem(name(c)) OR stem(t) = stem(synonyms(c)) **then**
if POS(t) = POS(c) **then** call replace(c, t, corres- noun-phrase) **end for**
if c is a subclass of concept "service" **then** assign "title" as <name(c)> in KV format
if c is object type property or c is-a member or instance **then** assign superclass(c) as <t> in KV format
if c is a datatype property **then** assign name(c) as <t> in KV format **end for**

The general representation of the formal query has the following format.

<service_type = < X > > : { "title" : < Y > , "temporal expr" : < Z > [<Concept name> : < A > , <Object Property name> : < B > , <Data Property name> : < C >] }

For example, the formal representation of the query *"List the popular storage service providers from North America"* is <service_type = "storage">:
{"title": "Storage service", "temporal expr": "popular" [hasLocation: North_America]}, where the datatype property hasLocation in ontology is used to enhance the query.

4.4 Service matching

Given a user's service request, the system should be able to narrow down the search within a small group of eligible providers rather than checking all providers against the user's request. The ontology based matchmaking of services appears in this perspective as a promising solution, allowing the efficient discovery and selection of services, adapted to the constraints

of the user requirement. When a service query requirement comes, we should find the cluster of services whose attribute values are closest to the service requested. For this, compare feature vector of service query with the feature vector of the centre of each cluster. After that, in the closest cluster, ontology-based service matchmaking method can be used to find the suitable service of choice. With the concept ontology, the similarity between service and service query is calculated. If the similarity degree is larger than the threshold, then the service is matched, or else the service is discarded. Thus, this approach simplifies the search process by doing calculations only within the most similar cluster, rather than with all the cloud service entries. Due to a limited number of services in each group, the time taken for service searching is low.

The most relevant cloud services with its details are presented as a list of suitable services to the user. The service request *q* can be expressed as $Q_a = < Q_a^n, Q_a^c >$ where numeric attributes $Q_a^n =< q_{a1}^n, q_{a2}^n, \dots, q_{ai}^n >$, categorical attributes $Q_a^c =< q_{a1}^c, q_{a2}^c, \dots, q_{aj}^c >$ and $0 \leq i, j \leq m$. Similarly, centre of cluster $Xa =< X_a^n, X_a^c >$ where numeric attributes $X_a^n =< x_{a1}^n, x_{a2}^n, \dots, x_{ai}^n >$ and categorical attributes $X_a^c =< x_{a1}^c, x_{a2}^c, \dots, x_{aj}^c >$. Similarity between the query *q* and the center of the cluster *x* can be found as in equation 5:

$$Sim(q,x) = \frac{i}{m} * q^n + \frac{j}{m} * q^c \quad where\ q^n = Sim^{num}(q_{ap}^n, x_{ap}^n), \text{ and } 1 \leq p \leq i$$
$$q^c = Sim^{cat}(q_{aq}^c, x_{aq}^c), \quad \text{and } 1 \leq q \leq j$$

Algorithm 4: Discovery of services
Input: Query service, Closest cluster centroid
Output: Similar services list
Steps
1. Compare the similarity of Query service (Q_s) with each service $s_i \in$ Closest cluster (CC) using equation 6.
2. If the similarity score ≥ threshold τ , append the service to the list; else discard the service.

Once the nearest cluster is found, we apply ontology based matchmaking to compare the services within the cluster.
The similarity is calculated using the equation proposed by Kolodner and Simpson [32]:

$$S(t,r) = \frac{\sum_{i=1}^{n} w_i * sim(a_i^t, a_i^r)}{\sum_{i=1}^{n} w_i} \quad (6)$$

Where S (t, r) is the global similarity between the target '*t*' and the source '*r*'; 'w_i' is the weight of the attribute '*i*'; a_i^t and a_i^r are the value of attribute '*i*' of target '*t*' and source '*r*' respectively. Since all attributes are given equal weights the equation (6) reduces to:

$$S(t,r) = \frac{1}{n}\sum_{i=1}^{n} sim(a_i^t, a_i^r) \quad (7)$$

$sim(a_i^t, a_i^r)$ is calculated according to overlap coefficient and similarity for numerical attributes.
The similarity measure for categorical attributes can be represented by

$$Sim(C_i, C_j) = a\ \frac{|A_i \cap A_j|}{f(A_i, A_j)} + (1-a)\frac{|A_i' \cap A_j'|}{g(A_i', A_j')} \quad (8)$$

where A_i and A_j are the sets of attributes of classes C_i and C_j, $|A_i \cap A_j|$ is the number of common attributes shared by classes C_i and C_j and a takes the values of 0 or 1. When a=1 and

$$f(A_i, A_j) = \min(|A_i|, |A_j|), \quad (9)$$

equation (8) becomes the overlap coefficient given as:

$$sim_{overlap}(C_i^t, C_i^r) = \frac{|A_i^t \cap A_i^r|}{\min(|A_i^t|, |A_i^r|)} \quad (10)$$

where A_i^t and A_i^r are the set of formal attributes of the class specified in feature C_i^t and C_i^r. The overlap between two sets of attributes of classes are equal to the intersection between the two sets of attributes normalized by the size of the minimum number of attributes.
The similarity between two numeric values in the same domain can be calculated as the following formula.

$$sim_{num}\left(a_i^t, a_i^r, a_i\right) = 1 - \frac{|a_i^t \cap a_i^r|}{a_i^{max} - a_i^{min}} \tag{11}$$

where a_i^t *and* a_i^r are the numeric values of the attribute a_i of the target and source; a_i^{max} and a_i^{min} are the maximum and minimum numeric values of the attribute a_i.

Choosing services exclusively in the light of their functionality may bring about services with detrimental QoS. QoS-based service selection is vital in the cloud as the huge number of services will certainly bring upon competition among providers that offer similar functionality. A large number of competing cloud services and the lack of technical skills of naïve users insist an efficient methodology for ranking services [33]. Top N services, whose similarity scores are computed, are ranked based on user preferences, user feedback and QoS of cloud services like availability and cost. The time complexity of ranking is O (C N log N) where N is the number of services and C is the number of criteria chosen for service selection. Rank scores of service "i" can be mathematically given as:

$$\begin{aligned} Rank\_score_i &= Similarity\_score_i + User\ \ preferences_k * \theta_k + \\ &User\ Feedback_i * \omega_1 + Availability * \omega_2 + Cost * \omega_3 \end{aligned} \tag{12}$$

where 'i' represents the service whose rank is currently evaluated. 'k' represents various user preferences other than availability and cost and $0 \leq k \leq m$. 'ω' and 'θ' represents the corresponding weights assigned to various criteria chosen for service ranking and

$$\sum_{i=1}^{3} \omega_i + \sum_{k=1}^{m} \theta_k = 1 \tag{13}$$

The user queries and discovered services are stored in a repository which can be used as a reference whenever the user queries have temporal expression. The repository is defined as a set of entries. Each entry is composed of a set of questions and answers. **Table 4** shows the common temporal expressions used in user queries.

Table 4. Usual temporal expressions in user query

Temporal expression	**Sample Token**	**Example query**
Prepositional phrases	in a year	List the service providers which provide news from multiple sources in the last year.
Adjectival phrases	current, popular	Name the popular storage providers with on the fly encryption system.
Adverbial phrases	recently, frequently	Give frequently used service providers for photo album management.
Noun phrases	today	Fetch the services to listen online music with less cost per hour on weekends

5. Experimental Results and Analysis

All experiments are implemented with Java 6 JRE in Eclipse 4.3. Ontology was constructed using Protégé 4.3 and accessed using Jena API. Experiments are conducted on a machine with 1.90 GHz Intel Core I3-3227U CPU and 4GB RAM. The system offers a simple GUI to

facilitate the discovery process as shown in **Fig. 2**. The discovery process has as input a query expressed in plain text. The goal of the SSDCN is to automatically create structured formal queries by mapping the natural language question into KV representation. The implemented technologies are transparent to the user. As there is no established standard evaluation dataset for cloud services related queries, sample queries were generated in all possible service categories. The user's requirements may contain many features such as geographical location of the data center, deployment model, security policies or even more detailed technical aspects like a number of virtual machines needed. The user can pose simple (e.g. Q1, Q2, Q3, Q6, Q7, Q8, Q13, Q15) or complex (e.g. Q4, Q5, Q9, Q10, Q11, Q12, Q14) queries as shown in **Table 5**.

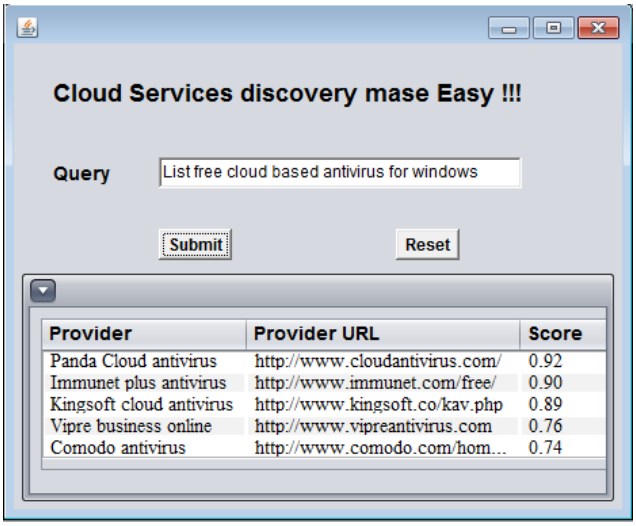

Fig. 2. User interface for service discovery

Table 5. Sample queries used for service discovery

Q1	List the payroll providers available in America.
Q2	Fetch the providers who offer free storage services of atleast 5 GB.
Q3	List document management systems with document encryption features.
Q4	Fetch free web hosting service with PHP and MySQL
Q5	Give the service providers for file sharing and syncing with minimum free storage of 1GB
Q6	List the providers who give Java web hosting platform with minimum pay per resource
Q7	Name the infrastructure providers who have a minimum of 5 instance types
Q8	List the services that can play online videos with cost less than 5 dollars per month
Q9	Name the vendors who offer Linux based web hosting platform which supports dedicated IP addresses.
Q10	List the providers who offer a Linux virtual machine with price less than $0.25 hourly.
Q11	Name the providers who offer ERP services with a cost less than 10$ per month and data center in Asia.
Q12	Fetch the services with self instantiating virtual machines on 2GHz processing and 4MB cache and not less than 8GB RAM with a cost less than 25 dollars per month.
Q13	Provide data analytic services with migrating option hosted on windows platform.
Q14	Fetch the providers who give customer relation management services on Java with 24/7 technical support.
Q15	Fetch cloud-based multi player game services with cost less than 5 dollars per month

Experiments were implemented over a collection of 6743 services covering 30 categories such as various applications, platform, storage, etc. Clustering of services was done in a reasonable time and service discovery can be entirely conducted offline. With no clustering in advance, the average search complexity to find the most similar service from N services is *O(N)*. With clustering, the service matching will first locate the appropriate cluster by comparing the distance between query and cluster centroid and then search each service in that cluster. As a result, the average search complexity will be *O (M)*, where $M = K + O(S_i)$, assuming that there are *'K'* categories, each category has *'Si'* services. This is very promising since the number of services does not exceed a few hundred in most of the service categories.

The average time taken to discover services, using and without using clusters, is calculated with varying number of services. General methods for service discovery compare the request with all of the services in the registry. **Fig. 3** shows the time taken for discovering services with and without clustering for three queries (Q1, Q2 and Q5) of simple, medium and complex constraints. Usage of service clusters improves the system response time to a great extent as depicted in **Fig. 3.** SSDCN method can reduce response time to 23-80% compared with service discovery without clustering. As the number of service increases, the response time for service discovery without clustering increases dramatically compared with the proposed method. Hence, service discovery based on clustering can effectively improve time efficiency of service discovery.

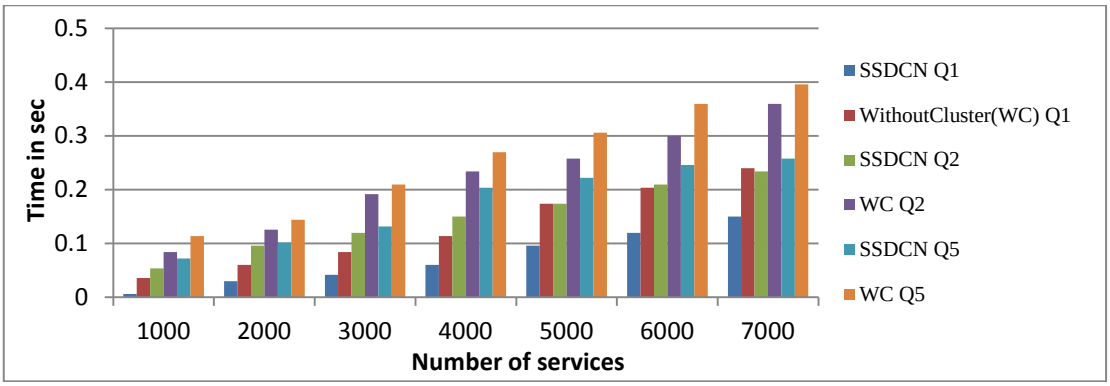

Fig. 3. Response time for Queries Q1, Q2 and Q5 with and without clustering

For each sample query, a set of services was manually chosen. SSDCN approach is compared with keyword based search, WordNet-based lexical query expansion and ontology-based semantic query expansion. The results of these approaches were compared with manually selected suitable services. The rank of 5 services found common in the top-15 list in all the four approaches, for queries with various constraint level, were used in the performance analysis. Here, ANOVA is used to analyze the performance of each service selection method. The null hypothesis for the rows states that the row mean is equal for all the rows irrespective of the service in consideration. The null hypothesis for the columns states that all the approaches perform equally well. The ranking of 5 common services for query Q4 and the results of a two-way ANOVA test is shown in **Table 6** and **Table 7**.

Table 6. Rank score of query Q4 for various methods of service discovery

	Keyword	**Lexical**	**Semantic**	**SSDCN**
000webHost	0.4	0.5	0.54	0.56
Biz.nf	0.44	0.53	0.6	0.74
ByteHost	0.3	0.48	0.65	0.63
100webspace	0.54	0.69	0.79	0.86
Awardspace	0.63	0.79	0.86	0.91

Table 7. Results of two-way ANOVA test

Source of Variation	***SS***	***df***	***MS***	***F***	***P-value***	***F critical***
Rows	0.27487	4	0.068718	36.86232	1.19E-06	3.259167
Columns	0.22228	3	0.074093	39.74609	1.65E-06	3.490295
Error	0.02237	12	0.001864			
Total	0.51952	19				

The value of F is much higher than $F_{critical}$. It means that there is a significant difference between the four service discovery approaches. Also, P- value is less than α=0.5. Hence, both the null hypothesis can be rejected.

Precision and recall are used to evaluate the effectiveness of the proposed approach. Precision is the fraction of retrieved documents that are relevant. The recall is the fraction of relevant documents retrieved. F-score is the harmonic mean of precision and recall. Mathematically, they are defined as follows:

$$Precision = \frac{\text{Number of relevant services retrieved in a service discovery}}{\text{Total number of services identified}} \tag{14}$$

$$Recall = \frac{\text{Number of relevant services retrieved in a service discovery}}{\text{Total number of relevant services available}} \tag{15}$$

$$F - score = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{16}$$

Table 8 shows the comparison of performance measurement of various approaches. The results show that the SSDCN approach outperforms the rest in terms of high precision and recall values. Keyword based search has low precision due to the difference in the terminology used by providers and naïve users. SSDCN employs the power of both lexical expansion and semantic expansion of query words. Precision has a higher margin on complex queries as they have more constraints. Ontology-based semantic query expansion and SSDCN have comparable precision. **Fig. 4** shows that the SSDCN approach outperforms other approaches in terms of F-score. The false negatives in semantic approach become less as the number of constraints in the query increases. Hence, the gap between ontology-based semantic query expansion and SSDCN in recall rate and f-measure becomes smaller. From **Table 8** it is evident that semantic based methods are much more efficient compared to traditional keyword-based methods and WordNet-based lexical expansion methods for finding similar cloud services.

Table 8. Comparison of precision and recall for various methods of service discovery

Query	Keyword based		Lexical expansion		Semantic expansion		SSDCN	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Q1	0.45	0.39	0.69	0.63	0.89	0.80	0.91	0.93
Q2	0.53	0.20	0.70	0.40	0.88	0.75	0.94	0.95
Q3	0.47	0.40	0.69	0.59	0.91	0.92	0.93	0.95
Q4	0.37	0.49	0.55	0.80	0.91	0.96	0.93	0.98
Q5	0.34	0.48	0.63	0.85	0.91	0.92	0.96	0.95

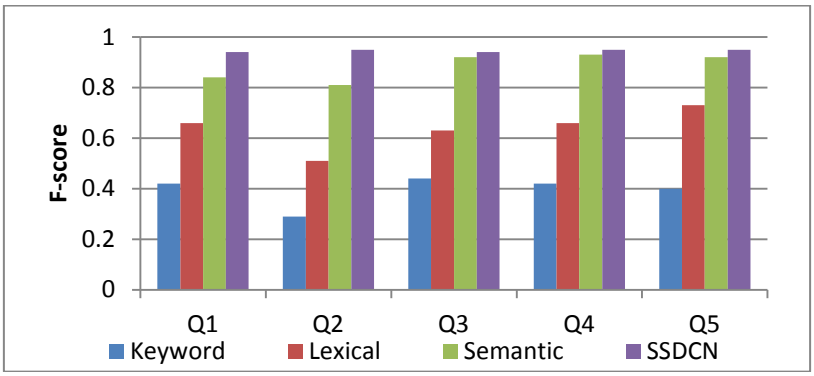

Fig. 4. Comparing F-Score values of different methods for a set of queries

6. Conclusion

This paper presented an automated cloud service discovery system which provides solutions that accurately match the suitable services in terms of the naïve user request. The experimental results demonstrate the effectiveness of natural language processing and clustering in cloud service discovery. It is not an easy task to effectively find the services that semantically match the user's requirements in free text. Ontology-based natural language processing techniques are employed to convert the user request in plain text to a formal representation. Ontology-based service discovery help to bridge the ambiguity of the user query in natural language and the service description. Based on the assumption that the efficiency of finding services can be improved if services are grouped together, enhanced k-means approach is used for clustering of services. Clustering helps in speeding up the discovery processes resulting in better efficiency. Overall, the results show that compared with existing methods, SSDCN improves recall and precision. Results show that SSDCN can effectively help an amateurish user to identify services that are closest to their preferences. In the future, we plan to consider methods to evaluate trust and reliability of providers from the user experience as an important benchmark for ranking services. The performance of SSDCN in video streaming and multimedia conferencing services will also be explored in the future.

References

[1] Chen Fei, Xiaoli Bai, and Bingbing Liu, "Efficient service discovery for cloud computing environments." *Advanced Research on Computer Science and Information Engineering*, Springer Berlin Heidelberg, pp. 443-448, 2011. Article (CrossRef Link)

[2] C.N. Höfer, and G. Karagiannis, "Cloud computing services: taxonomy and comparison," *Journal of Internet Service Applications*, issue 2, pp. 81–94, 2011. Article (CrossRef Link)

[3] Kona Srividya, Ajay Bansal, Luke Simon, Ajay Mallya, and Gopal Gupta. "USDL: a service-semantics description language for automatic service discovery and composition," *International Journal of Web Services Research (IJWSR) 6*, no. 1, pp.20-48, 2009. Article (CrossRef Link)

[4] Topology and Orchestration Specification for Cloud Applications Version 1.0. OASIS Committee Specification Draft 03, 2012.

[5] Anderson E, L. Lam, C. Eschinger, S. Cournoyer, J. M. Correia, L. F. Wurster, R. Contu et al. "Forecast overview: Public cloud services, worldwide, 2011-2016, 4Q12 Update," *Gartner Inc.*, February 2013.

[6] Afify Yasmine M, "Cloud Services Discovery and Selection: Survey and New Semantic-Based System," *Bio-inspiring Cyber Security and Cloud Services: Trends and Innovations*, Springer Berlin Heidelberg, pp. 449-477, 2014. Article (CrossRef Link)

[7] Noor T H, Sheng Q Z and Bouguettaya A. "Cloud service crawler engine," *Trust Management in Cloud Services*. Springer International Publishing, pp. 69–79, 2014. Article (CrossRef Link)

[8] Viji Rajendran V, and Swamynathan S, "Multi Threaded priority based semantic crawler for cloud services," *International conference on Intelligent Information Technologies (ICIIT)*, Chennai, pp.122-130, 2014.

[9] A.K. Jain, M.N. Murty, P.J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31,no. 3, pp. 264 – 323, 1999. Article (CrossRef Link)

[10] Studer Rudi, "Semantic Service Discovery using Natural Language Queries," *Diss. SAP Research*, 2009.

[11] Frank Gens, IDC Predictions 2015: Accelerating Innovation and Growth on the 3rd Platform, Doc # 252700, Dec 2014.

[12] Sun L, Dong H, Hussain F K, Hussain O K, and Chang E. "Cloud service selection: State-of-the-art and future research directions," *Journal of Network and Computer Applications*, vol 45, pp.134-150, 2014. Article (CrossRef Link)

[13] Zhao L, Ren Y, Li M and Sakurai K, "Flexible service selection with user-specific QoS support in service-oriented architecture," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 962- 973, 2012. Article (CrossRef Link)

[14] UDDI Technical White Paper (2001), http://www.uddi.org/pubs/lru_UDDI_Technical_Paper Article (CrossRef Link)

[15] D.K. Nguyen, F. Lelli, M.P. Papazoglou, W.-J. van den Heuvel, "Blueprinting approach in support of Cloud computing," *Future Internet. 4 (1)*, pp.322–346, 2012. Article (CrossRef Link)

[16] DMTF, Open Virtualization Format Specification Version 1.0.0, Specification DSP0243, Distributed Management Task Force, Inc., 2009.

[17] Cloudbook. The Cloud Computing & SaaS Information Resource, http://www.cloudbook.net/directories/product-services/cloud-computing-directory?category=Applications Article (CrossRef Link)

[18] Semantic technology institute. (2009). Seekda! Available from http://seekda.com/ Article (CrossRef Link)

[19] Moscato, Francesco, Rocco Aversa, Beniamino Di Martino, T. Fortis, and Victor Munteanu, "An analysis of mOSAIC ontology for Cloud resources annotation," in *Proc. of Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 973-980. IEEE, 2011.

[20] Liu X, Huang G and Mei H, "Discovering homogeneous web service community in the user centric web environment," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 167– 181, 2009. Article (CrossRef Link)

[21] Sim K. M., "Agent-based Cloud computing," *IEEE Transaction on Service Computing*, vol. 5, no. 4, pp. 564-577, 2012. Article (CrossRef Link)

[22] Tahamtan, A., Beheshti, S.A., et al., "Cloud Repository and Discovery Framework Based on a Unified Business and Cloud Service Ontology," *8th IEEE World Congress on Services, IEEE Press*, USA , pp. 203–210, 2012. Article (CrossRef Link)

[23] The W3C SPARQL Working Group, "SPARQL 1.1 Overview," *W3C recommendation* 21 March 2013.

[24] Unger, Christina, and Philipp Cimiano. “Pythia: compositional meaning construction for ontology-based question answering on the semantic web,” *Natural Language Processing and Information Systems.*, Springer Berlin Heidelberg, pp.153-160, 2011. Article (CrossRef Link)
[25] Gruber, Thomas R. "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies 43.5*, pp.907-928, 1995. Article (CrossRef Link)
[26] Sun Le, Hai Dong, and Jamshaid Ashraf., “Survey of service description languages and their issues in cloud computing,” in *Proc. of Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on*, pp. 128-135. IEEE, 2012. Article (CrossRef Link)
[27] Garcia R, Angel M, Valencia-Garcia R, Garcia- Sanchez F and Samper-Zapater J, "Ontology-based annotation and retrieval of services in the cloud," *Knowledge-Based Systems* 56, pp. 15–25, 2014. Article (CrossRef Link)
[28] Abdullah A, Noor T H, Sheng Q Z and Yong Xu, "Towards ontology-enhanced cloud services discovery," *Advanced Data Mining and Applications*, Springer International Publishing, pp. 616–629, 2014. Article (CrossRef Link)
[29] Tanimoto T, 1957 IBM Internal Report 17th Nov.1957.
[30] Santorini, Beatrice. "Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)." 1990.
[31] Miller, George A. "WordNet: a lexical database for English," *Communications of the ACM 38.11*, pp. 39-41, 1995. Article (CrossRef Link)
[32] J. Kolodner and R. Simpson. The MEDIATOR: Analysis of an early case-based problem solver. Cognitive Science, 13(4):507-549, 1989. Article (CrossRef Link)
[33] Qi Yu, “*CloudRec*: a framework for personalized service Recommendation in the Cloud,” *Knowl Inf Syst*, vol. 43, pp.417–443, 2015. Article (CrossRef Link)
[34] J. Wu, Y. Shang, C. Yuen, B. Cheng, and J. Chen, "TRADER: A Reliable Transmission Scheme to Video Conferencing Applications over the Internet," *Journal of Network and Computer Applications*, vol. 44, pp. 161-171, 2014. Article (CrossRef Link)
[35] J. Wu, B. Cheng, C. Yuen, Y. Shang, J. Chen, "Distortion Aware Concurrent Multipath Transfer for Mobile Video Streaming in Heterogeneous Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 14, no. 4, pp. 688-701, 2015. Article (CrossRef Link)
[36] J. Wu, B. Cheng, Y. Shang, C. Yuen, J. Chen, "A Novel Transmission Scheme to Inter Destination Video Synchronization," *IET Communications*, vol. 9, no. 5, pp. 603-612, 2015. Article (CrossRef Link)

Viji Rajendran V received her Bachelor’s degree in Computer Engineering from Cochin University of Science and Technology in the year 2002 and Master’s degree in Computer Science and Engineering from Anna University in the year 2008. She is currently pursuing Ph.D. in the Department of Information Science and Technology, Anna University, Chennai, India. Her research interest includes Cloud Services, Ontology and Semantic Web. She is a lifetime member of Indian Society for Technical Education (ISTE).

Swamynathan Sankaranarayanan received his Master’s degree in Computer Science and Engineering and Doctorate in Distributed Computing from Anna University, Chennai. He is currently working as an Associate Professor of Department of Information Science and Technology, College of Engineering Campus, Anna University, Chennai, India. He has more than 20 years of teaching and research experience. He has carried out various funded projects. He has published more than 60 papers in reputed journals and conference proceedings. His research interest includes Distributed Computing, Semantic Web and Web Mining.