Cloud Services Discovery and Selection Assistant

Hamdi Gabsi, Rim Drira and Henda Hajjami Ben Ghezala
RIADI Laboratory, National School of Computer Sciences, University of Manouba, La Manouba, Tunisia

Keywords: Cloud Services Discovery, Cloud Services Selection, Cloud-based Software Development, Cloud Data-set, Natural Language Processing, Cloud Services Clustering.

Abstract: The surging popularity of cloud services has led to the emergence of numerous cloud providers who offer various services. The great variety and the exponential proliferation of cloud services over the Web introduce several functionally similar offers with heterogeneous descriptions and contrasting APIs (Application Programming Interfaces). Due to this heterogeneity, efficient and accurate service discovery and selection, based on developers-specific requirements and terminology, have become a significant challenge that requires a high level of expertise and a steep documentation curve. In order to assist developers in handling these issues, first, we propose a Cloud Services Discovery and Selection Assistant (DESCA) based on a developer's query expressed in natural language. Second, we offer to the developers a cloud data-set, named ULID (Unified cLoud servIces Data-set), where services offered by different cloud providers are collected, unified and classified based on their functional features. The effectiveness of our contributions and their valuable insights to improve cloud services discovery and selection are demonstrated through evaluation experimentation.

1 INTRODUCTION

Cloud services are becoming crucial building blocks for developing scalable applications, given their important impact to reduce the management cost and the time to market. The increasing interest in cloud services has led to an expeditious diversity of services to cover all developers' requirements. Nonetheless, this diversity brings several challenges, essentially, during the discovery and selection process. The discovery process is defined as the process of detecting automatically or semi-automatically services responding to functional requirements. The selection process consists of retrieving the suitable service among discovered services given a specific required Quality of Service (QoS) (Sun et al., 2014) .

Cloud services discovery and selection is considered as a challenging task for different reasons. First, cloud providers often publish their services descriptions, pricing policies, and Service Level Agreement (SLA) rules on their portals in various and heterogeneous formats. Therefore, most of the available cloud services come with non-standard format (e.g., HTML documentation and textual descriptions). To deal with this heterogeneity, a steep documentation curve is required to clearly identify relevant services' functional features and compare several offers. Second, cloud services are continuously evolving (the update of existing services and the emergence of new services), handling this evolution during the discovery process is known to be a challenging task for cloud applications' developers.

Several studies have been carried out to address these challenges using different approaches such as semantic-based approaches (Martino et al., 2018), syntactic-based approaches (Lizarralde et al., 2018), and registry-based approaches (Abdul Quadir Md and Mandal, 2019). Even though these approaches are scientifically interesting, their main limitation is the discovery scope, which is often limited to some services that are published in a specific description standard. In fact, these approaches are based on either semantic description standard such as OWL-S (Martino et al., 2018) or Web Services Description Language (WSDL) (Lizarralde et al., 2018), (Bey et al., 2017). This limitation is impractical since it expects available cloud services to have semantic tagged descriptions or WSDL describing files, which is not the case in a real-world scenario.

In order to address the cited limitations, we propose a cloud services discovery and selection approach that does not make any assumptions about the standard description languages of the provided cloud services. Indeed, we assume, in our approach,

Gabsi, H., Drira, R. and Ben Ghezala, H.
Cloud Services Discovery and Selection Assistant.
DOI: 10.5220/0009403801580169
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 158-169
ISBN: 978-989-758-421-3

that services descriptions are provided in natural language and developer's requirement are expressed in keyword-based queries to offer a simple syntax in terms of open vocabularies wherein the developers can use their own terminology to express their needs. In that respect, we define an automatic process for services' functional features extraction from natural language descriptions, which deals with the heterogeneous nature of cloud service. Furthermore, we propose a publicly available cloud data-set where services are unified based on their functional features and regularly updated. Thus, we manage the diversity and the dynamic evolution characteristics of cloud services that practically complicate the discovery process. The proposed data-set contains more than 500 services including IaaS, PaaS and SaaS services and provided by Amazon, Google, and IBM. It was reviewed and accepted by Elsevier Mendeley Data (Elsevier Mendeley Data , 2019) and available on (ULID , 2019).

We detail, in this paper, our proposed discovery and selection process. We go in-depth demonstrating how cloud services capabilities are collected, stored, matched with developer request and finally ranked. The main contributions of this paper are as the following:

- Proposing a Unified cLoud servIces data-set (ULID) which can automatically manage the heterogeneity and the evolving nature of cloud services.
- Putting forward a unification strategy of scattered cloud services based on functional clustering using a modified K-means algorithm.
- Proposing a Cloud Services Discovery and Selection Assistant (DESCA) based on natural language query aiming to meet developers' functional requirements.
- Demonstrating the effectiveness of DESCA through experimental evaluation.

The remainder of this paper is organized as follows: Section 2 presents a motivating scenario. In Section 3, we discuss the related work. Section 4 details our discovery and selection assistant DESCA. Section 5 illustrates the evaluation of DESCA. Section 6 concludes the paper and outlines our ongoing works.

2 MOTIVATING SCENARIO

Let consider a scenario where a company decides to develop its e-commerce web site using cloud services. After analyzing the specification, the development team decides to use cloud services as much as possible and looks for the following services: relational database, a load balancer, data analysis service, a high performance computing resource and a PHP web development tool. To find candidate services for each requirement, the first solution to think about is to review manually several cloud provider web portals or to use a search engine (such as Google). For sure, the search result enables to find various services of different providers. However, from one side, the obtained results always contain irrelevant information that must be manually discarded. From another side, the content of each provider web portal needs to be perused manually to identify services features because their services descriptions are often scattered in HTML pages. Then, comparing services manually and selecting the most appropriate ones with regard to the development team's needs (i.e functional, budget and QoS requirements) is a tedious and time consuming task.

Actually, there are several commercial cross-platforms' services search, however most of them are paying. Furthermore, the main important issue of these platforms remains the neutrality and detachment regarding commercial cloud providers.

This simple scenario sheds the light on the importance of assisting developers in performing seamless cloud services discovery process. To do so, we need to reference scattered cloud services regardless of their providers and their heterogeneous descriptions in unified data-set.

3 RELATED WORK

Many research efforts have been made in order to assist the cloud services discovery and selection process. They can be presented in two different points of view, namely: architecture view and matchmaking view (Martino et al., 2018).

From one side, the architecture view is divided into centralized and decentralized. The centralized architecture depends on one central node that provides a complete view of all cloud services being offered in the market. This architecture can be achieved by proposing a cloud services registry or using cloud broker platforms.

Actually, several cloud broker platforms have been proposed. (Jrad et al., 2015) developed a cloud broker system to select cloud services based on the user QoS requirements and SLA attributes. The authors developed a utility-based algorithm for matching user functional and non-functional requirements to SLA attributes of cloud providers. (Rajganesh Nagarajan and Selvamuthukumaran, 2018) proposed a broker based cloud computing framework for enabling the users to specify their services requirements in terms of numerical representation. With respect to the user specification, the proposed broker constructs the cloud ontology to represent the available services from the service repository. The appropriate services are represented using semantic network which enables the user to know about the available services as per their posted requirements.

It is worth pointing that, even though cloud broker platforms can provide assistance in the discovery process, most of them are based on an additional layer between the provider and the final cloud user which can complicate the service's delivery chain and certainly increase the services' cost. In our work, we aim to propose an assistance framework which allow developers to fulfill seamless discovery process without any intermediaries, so that, they can better assume their decisions, and control their budgets.

The centralized architecture can be achieved using public registries such as (Asma Musabah Alkalbani and Kim, 2019) who provided a centralized cloud service repository. The authors propose a harvesting module to extract data from the web and make it available to different file format. The harvesting module uses an algorithm for learning the HTML structure of a web page. This work requires the user to determine specific control parameters such as targeted web page URL and the required information from in the targeted web page. Moreover, the collected data sets lack main service information such as services' description and operations.

From another side, the matchmaking view is divided into syntactic-based and semantic-based. The semantic-based matchmaking approaches are based on semantic description to automate the discovery and selection process. (Martino et al., 2018) proposed a cloud services ontology with automated reasoning to support services discovery and selection. However, the discovery scope, in this work, is depending on the pre-existence of providers specific ontologies (OWL-S services description files) that require mapping techniques to coordinate the difference between agnostic (abstract) and vendor dependent concepts to support interoperability. Even though many semantic approaches are scientifically interesting (Martino et al., 2018), they require that the developers have intimate knowledge of semantic services and related description and implementation details which makes their usage difficult. Moreover, from the service requestor's perspective, the requestor may not be aware of all the knowledge that constitutes the domain ontology. Specifically, the service requestor may not be aware of all the terms related to the service request. As a result of which many services relevant to the request may not be considered in the service discovery process.

The syntactic-based approaches are, generally, based on WSDL description of cloud services. (Bey et al., 2017) proposed a clustering algorithm based on similarity between users query concepts and functional description parameters of cloud services expressed in a WSDL document. Despite the high precision values found in this work, generally assuming that the candidate cloud services are described using WSDL files, is considered as impractical limitation.

The analysis of several research studies illustrates the main motivations of our proposal which are:

- First, the need for an efficient syntactic-matching approach which does not make any assumptions, such as particular standard or specific semantic representation, about the description language of available cloud services.
- Second, the relevance of a centralized architecture that references scattered cloud services regardless of their providers and their heterogeneous descriptions in unified data-set. This fact can practically assist the developers in a seamless search for relevant cloud services.

4 DESCA: CLOUD SERVICES DISCOVERY AND SELECTION ASSISTANT

In order to practically assist developers in the discovery and selection process, we need to efficiently deal with several cloud services proprieties.

First, we manage the heterogeneous nature of cloud service by properly extracting relevant services capabilities from ambiguous services' descriptions. To do so, we propose an automated process for services 'capability extraction in order to identify services' functional features. Based on these features, we define a services' functional clustering to unify functionally similar services. Thus, we can reduce the search scope and we improve the response time of the discovery process.

Second, we deal with the huge diversity of scattered cloud services by proposing a structured main source that provides to the developer relevant services meta-data and avoids laborious documentation task in several providers web portals. In that respect, we introduce ULID which presents a centralized cloud service data-set to which the developer has access to.

Last but not least, we manage the dynamic evolution of cloud services by regularly updating our data-set. Based on each commitment presented previously, we propose our discovery and selection assistant (DESCA). Indeed, DESCA is mainly composed of two components: ULID Construction Component (UCC) and Discovery & Selection Component (DSC). The input of our approach is keyword-based queries presenting developers functional requirements. The output of DESCA is a set of cloud services meeting developers needs. Ranking services according to QoS and budget is addressed in later steps outside the scope of this paper. Figure 1 presents the architecture of DESCA.

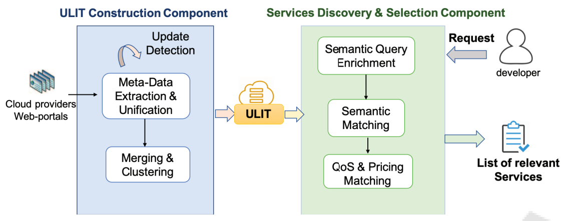

ULIT Construction Component
Services Discovery & Selection Component
Update Detection
Cloud providers Web-portals
Meta-Data Extraction & Unification
Merging & Clustering
ULIT
Semantic Query Enrichment
Semantic Matching
QoS & Pricing Matching
Request
developer
List of relevant Services

Figure 1: DESCA overview.

4.1 ULID Construction Component (UCC)

Our data-set is based on three main steps which are: Meta-data Extraction & Unification, Merging & Clustering and Update detection. We give further details about these steps in the next sections.

4.1.1 Meta-data Extraction & Unification

This step aims to collect cloud services meta-data which are scattered in HTML pages of cloud providers web portals. To do so, we use a HTML parser to harvest relevant cloud services meta-data from these portals. The collected meta-data is stored in conformance with a unified meta-model of services description given in figure. 2. Our meta-model defines relevant services meta-data that can facilitate services discovery and assist selection decisions.

The class ”Service” provides the name, the description, the endpoint and the available regions of the service. Each cloud service has a category and offers multiple products, a service product is defined as a business operation characterized by a price per unit (for example 0.01$ per GB for a storage service), a product family (such as data request), a begin range and an end range for payment. The class ”Provider” presents information related to the service's provider. The class ”QoS” presents the QoS which are; the response time, availability and reliability.

Identifying services functional features is considered as an important pillar in cloud services discovery.

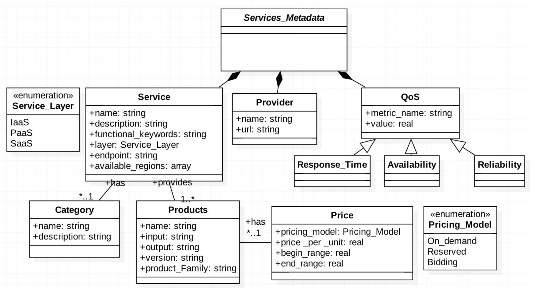


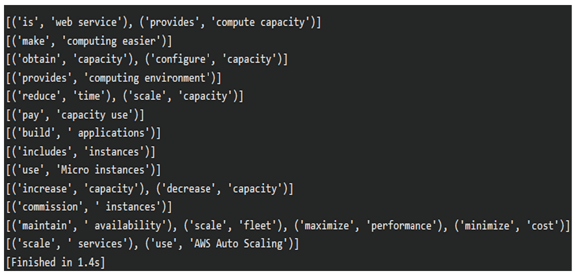
Figure 2: Meta-Model of cloud services meta-data.

In order to automatically extract relevant keywords presenting the functional features of cloud services, we use a natural language processing tool named Stanford Parser (Marneffe and Manning, 2015). The Stanford Parser can identify the grammatical structure of each sentence of service descriptions by creating grammatical relations or type dependencies among elements in the sentence. These dependencies are called Stanford Dependencies SDs (Marneffe and Manning, 2015). In our work, we model keywords as a set of binary relations $< action, object >$, where action denotes the functional feature of the service. Object denotes the entities affected by the action. Then, we use the SD sets to properly identify the grammatical relations between $< action, object >$. In fact, each SD is a binary relation between a governor (also known as a regent or a head) and a dependent (Marneffe and Manning, 2015).

To illustrate, let's suppose this sentence ” This service offers compute capacity in order to deploy workloads in a public cloud.”. We use the following SD:

- Relation direct object: dobj(governor, dependent) generally appears in the active voice, in which the governor is a verb, and the dependent is a noun or noun phrase as the direct object of governor. In our case, we obtain dobj(offers, capacity) and dobj(deploy, workloads) .
- Relation adverbial clause modifier: advcl(governor, dependent), an adverbial clause modifier of a verb phrase or sentence is a clause modifying the verb (consequence, conditional clause, purpose clause, etc.). For instance, advcl(offers, deploy).
- Relation prepositional modifier: prep(governor, dependent), is any prepositional phrase that serves to modify the meaning of the verb, adjective, noun, or even another preposition. In our case, we obtain prep_in(deploy, cloud).

The basic keywords extracted above may not present relevant functional features semantics. For example, the pair {offers, capacity}, we expect that it is {offers, compute capacity}. Likewise, the pair



```
[('is', 'web service'), ('provides', 'compute capacity')]
[('make', 'computing easier')]
[('obtain', 'capacity'), ('configure', 'capacity')]
[('provides', 'computing environment')]
[('reduce', 'time'), ('scale', 'capacity')]
[('pay', 'capacity use')]
[('build', ' applications')]
[('includes', 'instances')]
[('use', 'Micro instances')]
[('increase', 'capacity'), ('decrease', 'capacity')]
[('commission', ' instances')]
[('maintain', ' availability'), ('scale', 'fleet'), ('maximize', 'performance'), ('minimize', 'cost')]
[('scale', ' services'), ('use', 'AWS Auto Scaling')]
[Finished in 1.4s]
```

Figure 3: Functional keywords extraction.

{deploy, cloud} was expected to be {deploy, public cloud}. Therefore, the semantic extension of the basic extracted keywords is necessary. The semantic extension mainly refers to the noun part of the service functional feature. In fact, the semantic extension of nouns mainly includes qualifiers, adjectives, nouns, gerunds, and adverbs. Through the analysis of the text description of cloud services, we note that qualifiers, adverbs do not provide the semantic information, we need only a few adjectives to contain useful business semantic information. Therefore, we consider nouns and gerunds as modifiers for semantically extending the keywords. In the Stanford Parser, we mainly consider the noun compound modifier; nn(governor, dependent) relationship, which indicates that both the governor and dependent are nouns, and dependent is treated as a modifier to modify the governor.

Finally, we created a stop-word list to remove the meaningless functional features which contain verbs such as "allow, get, can, helps, etc.". Figure 3 illustrates a real example tested on a text description of a compute service offered by AWS (Amazon compute service description, 2019).

4.1.2 Merging & Clustering

We classify services referenced in our data-set in unified categories. Our main purpose, in this step, is to provide to the developers unified services categories regardless of cloud providers. Several cloud providers organize their offered services in different categories. We take advantage of the disposed categories in our unification process. indeed, common categories over different providers can be easily unified, while others are merged using a clustering algorithm based on the textual description of services categories. To do so, first of all, we extract appropriate keywords expressing the functional features of services categories as explained previously for services description. Second, we calculate the semantic similarity between each pair of categories based on extracted keywords. Finally, we gather categories into clusters using a modified K-means clustering algorithm. We detail each step as the following:

Classes Similarity Computation. We base our clustering approach on the following heuristic: *Services' categories over different providers tend to respond to the same business requirements if they share the same or similar keywords describing their functional features.*

After identifying categories' functional features which are the set of pairs $< action, object >$, we calculate the similarity $SC_{(C_1,C_2)}$ between services categories over different providers. We denote $C_i = \{p_{i1}, p_{i2}, p_{i3}, ..., p_{in}\}$ a category's functional features, where p_{in} is the pair $< action, object >$ and $|C_i|$ the cardinality of C_i, (the number of pairs p_i).

The similarity $SC_{(C_1,C_2)}$ is inspired from (Lin, 1998). Indeed, the authors prove the relevance of the proposed similarity formula in the context of words pairs similarity. The main asset of this work is defining similarity in information theoretic terms which ensure the universality of the similarity measure. The main issue of many similarity measures is that each of them is tied to a particular application or assumes a particular domain model. Dealing with this issue, the proposed similarity measure has significantly improved words' pairs similarity. The similarity formula $SC_{(C_1,C_2)}$ presents the sum of the similarities between each pair p_{1i} of C_1, and the pairs $\{p_{2_1}, p_{2_2}, p_{2_3}, ..., p_{2_n}\}$ of C_2, normalized by the cardinality of C_1. Formally, we obtain;

$$SC_{(C_1,C_2)} = \frac{\sum_{i=1}^{|C_1|}\left(\frac{\sum_{j=1}^{|C_2|} S(p_{1i}, p_{2j})}{|C_2|}\right)}{|C_1|} \tag{1}$$

where $S(p_{1i}, p_{2j})$ is the pairs similarity.

We define the pairs similarity as follows:

- Let A_1 and A_2 respectively denote the actions in the pair p_1 and p_2.
- O_{i1} and O_{i2} respectively denote objects in the pair p_1 and p_2.
- w_1, w_2 denote the weight of the action part and the object part. We suppose that some predefined action has a higher weight such as; offer, provide, deliver, etc. These weights are defined by the developers.
- m is the minimum number of objects of the pair p_1 and p_2 ($min\_number\_objects(p_1, p_2)$) whereas n is the maximum number of objects of the pair p_1 and p_2 ($max\_number\_objects(p_1, p_2)$).

The pair similarity is calculated as:

$$S_{(p_1,p_2)} = w_1 S_{(A_1,A_2)} + w_2 \frac{\sum_{i=1}^{m} S_{(O_{i1},O_{i2})}}{n} \tag{2}$$

where $S_{(O_{i1},O_{i2})}$ is words similarity. We use Jacard Similarity Coefficient to calculate the word similarity.

In fact, the Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the union divided by the size of the intersection of the sample sets.

$$J(E,F) = \frac{E \cup F}{E \cap F} \quad (3)$$

where E and F are two given sets of words. We create a feature set F(w) for each word 'w' (i.e for each object which is presented by a word in our context) containing the synonym set, generic word and interpretation of the word w. F(w) is created using BabelNet (Pamungkas et al., 2017). Based on (Lin, 1998), we define the similarity between the two words as follows:

$$S_{(w_1,w_2)} = \frac{2 \times I(F(w_1) \cap F(w_2))}{I(F(w_1)) + I(F(w_2))} \quad (4)$$

where $I(S)$ represents the amount of information contained in a set of features S. $I(S)$ is calculated as:

$$I(S) = -\sum_{f \in S} logP(f) \quad (5)$$

The probability $P(f)$ can be estimated by the percentage of words that have the feature f among the set of words that have the same part of speech in the entire BabelNet library database. When two words have the same feature set, then the maximum similarity is 1. The minimum similarity is 0 when the intersection of two words' features is empty.

Modified K-means Clustering Algorithm. The K-means algorithm is an algorithm widely used in the field of data mining. It aims to partition n elements (observations) into k clusters presented by k data centroids. Usually, k-means uses the Euclidean distance or Manhattan distance to assign each observation to the nearest cluster. We propose a modified K-means algorithm in order to have meaningful clusters and enhance the basic K-means algorithm results.

To do so, first, we left frequent and rare words unclustered. This fact is approved by the (Information Retrieval) IR community in order to have the best performance in automatic query expansion and avoids over-fitting.

Second, we enhance the cohesion and correlation conditions defined in the basic K-means algorithm by quantifying the cohesion and correlation of clusters based on our semantic functional similarity instead of Euclidean distances used in the basic K-means algorithm. The Euclidean distance is not consistent in our context because it does not provide meaningful information related to semantic similarity.

Third, to ensure that we obtain clusters with high cohesion, we only add an item (in our case a services' category) to a cluster if it satisfies a stricter condition, called cohesion condition. Given a cluster C, an item 'i' is called a kernel item if it is closely similar to at least half of the remaining items in C. Our cohesion condition requires that all the items in the cluster be kernel items. Formally;

$$i \in C \Rightarrow ||\forall j \in C, i \neq j, Sim(i,j) \geq \frac{||C||-1}{2} \quad (6)$$

We illustrate the major steps of the modified K-means algorithm as follows.

```
Modified K-means Algorithm
Input: classes scopes set (S = s1, ..., sn )
k the number of clusters
Output: k clusters
Let sim(s1, s2) be the similarity function
C = {c1, c2,... , ck }
(set of cluster centroids)
L = {L(s_i)|i = 1, 2, ..., n}
(set of cluster labels)
for all c_i in C do
ci <-- sj {Initialize Centroid (ci) }
end for
for all s_i in S do
l(s_i) <--  index_max_Sim(s_i,c_j)
end for
Centroid_Change <--False
cohesion_condition() <--True
repeat
for all c_i in C do UpdateCentroid(c_i)
end for
for all s_i in S do
M <-- index_max_Sim(s_i,c_j)
if M  /= l(s_i) then
l(s_i) <-- M
Centroid_Change <-- True
end if
Verify(cohesion_condition(s_i, s_l))
l in (1, 2, ..., n) l /= i
end for
until (Centroid_Change == False and
Verify(cohesion_condition()))
```

To keep our data-set up-to-date and manage the dynamic evolution of cloud services, we verify, first, if a cloud provider offers a services update's detection API. In this case, we take advantage of this API by using an agent-based system able to execute existing update's APIs. Otherwise we revisit providers' web portal periodically using the web scraper to detect new cloud services or those frequently updated. After updating ULID, we apply the clustering algorithm in order to assign each new service to the suitable cluster.

4.2 Discovery & Selection Component (DSC)

The DSC is responsible for discovering and selecting cloud services that meet the developer's requirements. It is composed of three modules: 1) the semantic query enrichment module, 2) the semantic matching module, and 3) the QoS & pricing matching module.

4.2.1 Semantic Query Enrichment

The Semantic Query Enrichment (SQE) aims to semantically enrich the keywords introduced by the developer using BabelNet (Pamungkas et al., 2017). Indeed, as a dictionary, BabelNet covers some specific terms from every word related to their terms. It maps all the stemmed words into their lexical categories. BabelNet groups nouns, verbs, adjectives, and adverbs into sets of synonyms called synsets. The synsets are organized into senses, giving, thus, the synonyms of each word, and also into hyponym/hypernym (i.e. Is-A), and meronym/holonym (i.e. Part-Of) relationships, providing a hierarchical tree-like structure for each word. In order to have meaningful enrichment, we filter BabelNet response by choosing the synsets which are part of the concept ”Cloud Computing” having the id ” bn:01225375n”.

4.2.2 Semantic Matching

The semantic matching module is responsible for identifying cloud services that best match the extended developer’s keywords. In fact, after specifying the cluster of cloud services meeting developer’s requirements, we use the ranking function ”Okapi BM25” (BM stands for Best Matching) (Whissel and Clarke, 2011) which is basically a TF-IDF enhanced function. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document. A document, in our case, is defined by the service’s name and its description, the search query is the introduced keywords by the developer. One of the most prominent instantiations of the function is as follows;

Given a query Q, containing keywords $q_1,...,q_n$, the BM25 score of a document ”D” is:

$$Score(D,Q)=\sum_{i=1}^{n} IDF(q_i)\frac{f(q_i,D)(K_1+1)}{f(q_i,D)+K_1(1-b+b\frac{|D|}{Avgdl})} \quad (7)$$

- $f(q_i,D)$: is q_i’s term frequency in D
- $|D|$: is the length of the document D in words
- Avgdl: is the average document length in ULID
- k_1 and b are free parameters, usually chosen, in absence of an advanced optimization, as $k_1 \in [1.2,2.0]$ and $b=0.75$
- $IDF(q_i)$: is the Inverse Document Frequency weight of the query term q_i. It is usually computed as:

$$\mathrm{IDF}(q_i)=\log\frac{N-n(q_i)+0.5}{n(q_i)+0.5} \quad (8)$$

- N is the total number of documents in ULID
- $n(q_i)$: is the number of documents containing q_i

In order to enhance ”Okapi BM25 ” results, we take into account the location of the keywords in the service description text. Indeed, if keywords are located at the beginning or in the middle of the service description text, probably, the main service’s purpose is responding to the functional requirements designed by these keywords. Consequently, this service is more important than others where keywords are located in the end. To do so, we propose a multiplicative factor:

$$M_f=1+\frac{N-i}{N} \quad (9)$$

where N is number of sentences in the service description text and i is the position of the sentence containing the keyword in the description text (i.e $i=1$ for the first sentence, etc.), we multiply the rating given by ”Okapi BM25 by M_f. We demonstrate the effectiveness of our proposition in the experimental evaluation.

4.2.3 QoS & Pricing Matching

After discovering the cloud services meeting the developers’ functional requirements, the QoS & Pricing Matching module is responsible of identifying services which respond to the developers’ budget and QoS constraints. We proposed in our previous works (Gabsi et al., 2018), (Gabsi et al., 2019) a personalized Simple Additive Weighting (SAW) algorithm aiming to identify suitable IaaS services meeting developers non-functional requirements. The elaboration of the QoS & Pricing Matching module to take into consideration other cloud services models specificities (PaaS & SaaS services) is the subject of our ongoing work.

5 RESULTS AND ANALYSIS

In order to illustrate DESCA, we set up, firstly, by evaluating each step of the UCC as well as the modified K-means clustering algorithm. Then, we proceed by evaluating the overall performance of the DSC.

5.1 ULID Construction Component Evaluation

To ensure a proper evaluation of the UCC, essentially, the merging & clustering step, we proceed by an external evaluation, namely, the clustering results are evaluated based on data that was not used for the clustering, such as known class labels and external benchmarks. These types of evaluation methods measure how close the clustering is to the predetermined benchmark classes (Rendan et al., 2011). To do so, we test the UCC on a real data-set presenting Azure Microsoft cloud services. The test-set is composed of 205 cloud services offered by Azure Microsoft (Microsoft Azure Services, 2019). It is created by parsing Azure Microsoft web portal (Microsoft Azure Services, 2019) and collecting services meta-data.

5.1.1 Services Functional Keywords Extraction

By analyzing the description of 205 services using the Stanford Parser, it is worth pointing that the functional keywords extraction highly depends on the terms used by the service providers in describing their services. In some cases, we obtain non-meaningful pairs $< action, object >$ due to the abundant use of adjectives for commercial purpose. In our case, we obtained 14 non-meaningful pairs. In order to verify the effectiveness of our method of extracting functional keywords (pairs $< action, object >$), we randomly selected 50 services as experimental data. We ask five developers to manually extract the sets of functional keywords pairs for each service, and compare them to the sets of functional keywords pairs automatically extracted. We evaluate the experimental results by calculating the precision and recall rate. The formula of the precision and recall rates are defined as follows:

$$Precision = \frac{S_A \cap S_M}{S_A} \qquad Recall = \frac{S_A \cap S_M}{S_M} \tag{10}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{11}$$

Where S_M represents the set of functional keywords pairs that is extracted manually, S_A denotes the set of functional keywords pairs that is automatically extracted. The Table 1 shows the experimental results (for reason of space restraint, we display a sample of five sets). For the extraction result, we have 0,7 as precision average, 0,98 as recall average and 0.817 as F- Measure average.

From Table 1, it can be concluded that the results of the functional keywords extracted by developers are different. The reason is that each developer has a different understanding of services description. The recall rate of the extracted results is almost close to 1.0. This shows that the automatically extracted sets of functional keywords pairs can cover all the keywords sets of each service. The precision average of the extraction results is 0,7, lower than the recall rate. The F-measure average is 0.817. This means that the automatically extracted functional keywords pairs provides acceptable results, but it leaves scope for enhancements due to the non- standardized description used by services providers.

5.1.2 Modified K-means Clustering Algorithm

While presenting our approach, we mention that we take advantage of the proposed services categories already offered by services providers. Thus, we applied our clustering algorithm on services' categories to unify them over different providers. Our approach remains accurate if we apply it directly on cloud services, namely we cluster services instead of services' categories. In that respect, we test our clustering algorithm on the test-set composed of 205 services offered by Azure Microsoft (Microsoft Azure Services, 2019). These services are functionally clustered, by the provider, in 18 categories (Machine learning, Analytics, Compute, Management services, etc) which present the predetermined benchmark classes.

We use the Purity of the Cluster as a metric to analyze the effectiveness of the modified K-means algorithm (Rendan et al., 2011). The following is the definition of cluster purity: suppose that D is the set of services to be clustered and C is the result of a clustering on D. $C_i \in C$ denotes a cluster in C, whereas S denotes the standard classification result on D, $s \in S$ denotes a class s in S, p_i denotes the largest number of services in the cluster C_i which are in common with s. The cluster purity CP (C_i) is defined as:

$$CP(C_i) = \frac{1}{|C_i|} max(p_i) \tag{12}$$

The clustering purity of the whole set of service to be clustered is defined as:

$$CP(C) = \sum_{i=1}^{C} \frac{|C_i|}{|D|} CP(C_i) \tag{13}$$

We set up our modified K-means algorithm on k=18. Figure 4 presents the clusters returned by the algorithm. To correctly interpret our results, we compare the purity values given by our algorithm to those given by he basic K-means algorithm. Table 2 shows the clustering results.

Using the modified K-means algorithm, the number of services in each cluster is slightly different

Table 1: Extracted functional keywords results.

Automatically extracted set of functional keywords pairs	Manually Extracted set of functional keywords pairs	Precision	Recall	F-Measure
$S_1 : \{< Create, Virtual\_machine >\}$	$S_1 : \{< Create, Virtual\_machine >\}$	1	1	1
$S_2 : \{< Create, Mobile\_application >, < Build, Mobile\_application >, < Deploy, Mobile\_application >\}$	$S_2 : \{< Create, Mobile\_application >, < Deploy, Mobile\_application >\}$	0,67	1	0,8
$S_3 : \{< Detect, Human\_faces >, < Identify, People >, < Organize, images >, < Compare, Image >, < Verify, Features >, < Provide, Face\_algorithm >\}$	$S_3 : \{< Detect, Human\_faces >, < Identify, People >, < Compare, Image >, < Verify, Features >, < Provide, Face\_algorithm >\}$	0,83	1	0,9
$S_4 : \{< Create, Data\_pipelines >, < Monitor, Data\_pipelines >, < Orchestrate, workflows >\}$	$S_4 : \{< Create, Data\_pipelines >, < Monitor, Data\_pipelines >, < Accelerate, Data\_integration >, < Orchestrate, workflows >\}$	0,75	1	0,86
$S_5 : \{< Protect, Data >, < Provide, Backup >\}$	$S_5 : \{< Protect, Data >, < Provide, Backup >\}$	1	1	1

Table 2: Cluster purity results.

Cluster Results	Central Service of Cluster	Number of Services	Purity values	Basic K-means purity
Cluster 1: Analytics	Azure Analysis Services	15	0,87	0,67
Cluster 2: Compute	Virtual Machines	13	0,85	0,62
Cluster 3: Containers	Container Registry	7	1	0,57
Cluster 4: Databases	Azure Database	11	0,91	0,72
Cluster 5: AI + Machine Learning	Machine Learning Services	31	0,90	0,65
Cluster 6: Developer Tools	Azure Lab Services	9	1	0,67
Cluster 7: DevOps	Azure DevOps Projects	8	0,88	0,63
Cluster 8: Identity	Azure Active Directory	6	0,67	0,50
Cluster 9: Integration	Event Grid	5	0,60	0,40
Cluster 10: Web	Web Apps	7	0,86	0,57
Cluster 11: Storage	Storage	13	0,92	0,69
Cluster 12: Security	Security Center	11	0,55	0,42
Cluster 13: Networking	Virtual Network	13	0,85	0,62
Cluster 14: Mobile	Mobile Apps	8	0,88	0,50
Cluster 15: Migration	Azure Migrate	5	1	0,60
Cluster 16: Media	Media Services	8	1	0,75
Cluster 17: Management Tools	Azure Managed Application	21	0,67	0,48
Cluster 18 : Internet of Things	IoT Central	14	0,92	0,64
		Clustering Purity	0,78	0,60

from that obtained according to the pre-classified services. This difference is due to some confusing descriptions basically for services in the integration and management tools clusters as well for the security and identity clusters. This explains the reason why the CP value is not very high for these clusters.

In light of the clustering results, we have 0,78 as purity value. It is worth mentioning that the proposed

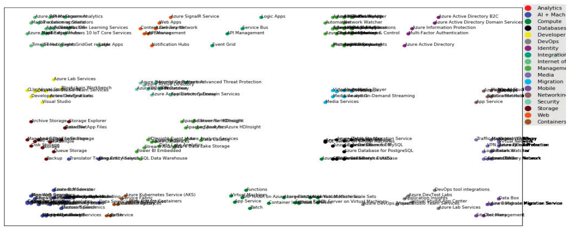


Figure 4: Services clustering results.

modifications (the cohesion condition, the weak correlation and the semantic similarity function) made on the basic K-means algorithm have contributed to better purity values (0,78 instead of 0,60).

5.2 Discovery & Selection Component Evaluation

We evaluate the DSC using the recall (R), the precision (P), the Top-k precision (P_k) and the R- precision (P_r) metrics. In this context, the precision evaluates the capability of the system to retrieve top-ranked services that are most relevant to the developers need, and it is defined to be the percentage of the retrieved services that are truly relevant to the developers' query. The recall evaluates the capability of the system to get all the relevant services in the data-set. It is defined as the percentage of the services that are relevant to the developer need.

Formally, we have;

$$P = \frac{|S_{Rel}|}{|S_{Ret}|} \qquad R = \frac{|S_{Rel}|}{|Rel|} \tag{14}$$

$$P_k = \frac{|S_{Rel,k}|}{k} \qquad P_r = P_{|Rel|} = \frac{|S_{Rel,|Rel|}|}{|Rel|} \tag{15}$$

where Rel denotes the set of relevant services, S_{Ret} is the set of returned services, S_{Rel} is the set of returned relevant services and $S_{Rel,k}$ is the set of relevant services in the top k returned services. Among the above metrics, P_r is considered to most precisely capture the precision and ranking quality of the system. We also plotted the recall/precision curve (R-P curve). An ideal discovery and selection assistant has a horizontal curve with a high precision value; an inappropriate assistant has a horizontal curve with a low precision value. The R-P curve is considered by the (Information Retrieval) IR community as the most informative graph showing the effectiveness of a discovery system (Davis and Goadrich, 2011).

In order to evaluate DSC, we present in Table 3 the results of some queries (for reason of space restraint, we display the top-4 retrieved services). The region "Ireland" is chosen by default for all the queries. We evaluated the precision of the retrieved services for different queries, and report the average top-2, top-5, and top-10 precision. To ensure the top-10 precision is meaningful, we selected queries which return more than 15 relevant services. DSC returns a total of 20 services per query. Figure 6 illustrates the results. The top-2, top-5, and top-10 precisions of DESCA using the enhanced BM25 function are 98%, 87%, 74% respectively, while the basic BM 25 results are 89%, 72%, 59% respectively. This demonstrates that taking into account the location of the keywords in the services text description (the enhanced BM25 function) can effectively improve the precision of the system. We plot the average R-P curves to illustrate the overall performance of DESCA. As mentioned previously, an ideal discovery assistant has a horizontal curve with a high precision value. Typically, precision and recall are inversely related, ie. as precision increases, recall falls and vice-versa. A balance between these two needs to be achieved by a discovery assistant. As illustrated by figure 5 and figure 6, for a recall average equals to 0,65 we have 0,87 as precision value. In fact, as an example, for the query containing {Virtual Machine, Compute Capacity, Server} as keywords, we have 30 services considered as relevant in ULID i.e $|Rel| = 30$, DESCA returns a total of 20 services per query i.e $|S_{Ret}| = 20$, among them 18 services are considered relevant i.e $|S_{Rel}| = 18$. We obtain a precision value $P = 18/20 = 0,9$ and a recall value $R = 18/30 = 0,6$.

It is worth pointing out that in some cases, depending on particular requirements, high precision at the cost of a recall or high recall with lower precision can be chosen. Thus evaluating a discovery and selection assistant must be related to the purpose of the discovery and the search process. In our case a compromise between the recall and the precision values is necessary. Therefore, we can announce that DESCA provides accurate results for cloud services discovery and selection.

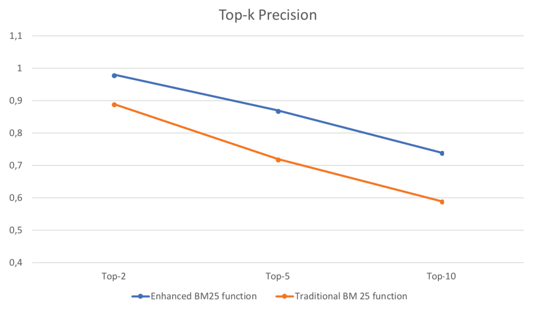


Figure 5: Top-k precision for retrieved services.

We present in figure 7 an extract of the returned result for our motivating scenario presented in Section 2. We suppose that the e-commerce website devel-

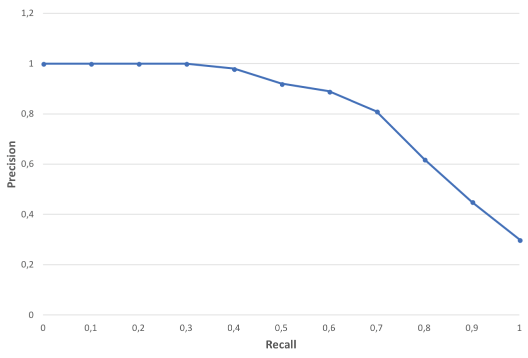
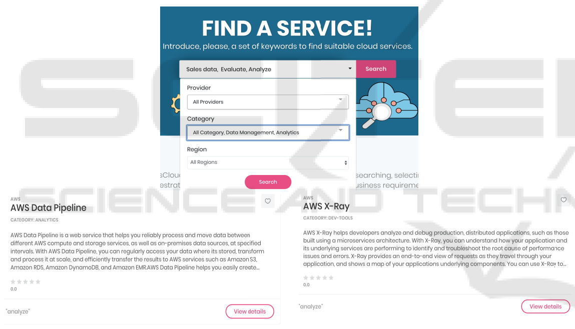


Figure 6: R-P curves of DESCA.

oper look for a data analysis service to evaluate the sales data. He/She may introduce the followings set of keywords: " Sales data, Evaluate, Analyze". The result is presented by DESCA as a set of services (and their short descriptions). The developer can access to all the details about each service. Moreover, we propose for each service a link of the related cloud patterns (Cloud Patterns, 2019) to explain and illustrate its relevant development use cases.



Sales data, Evaluate, Analyze

All Providers

All Category, Data Management, Analytics

All Regions

AWS
AWS Data Pipeline
CATEGORY: ANALYTICS
AWS Data Pipeline is a web service that helps you reliably process and move data between different AWS compute and storage services, as well as on-premises data sources, at specified intervals. With AWS Data Pipeline, you can regularly access your data where its stored, transform and process it at scale, and efficiently transfer the results to AWS services such as Amazon S3, Amazon RDS, Amazon DynamoDB, and Amazon EMR.AWS Data Pipeline helps you easily create...
0.0
"analyze"

AWS
AWS X-Ray
CATEGORY: DEV-TOOLS
AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors. X-Ray provides an end-to-end view of requests as they travel through your application, and shows a map of your applications underlying components. You can use X-Ray to...
0.0
"analyze"

Figure 7: Services discovery results.

6 CONCLUSIONS

Cloud computing offers several services that change the way applications are developed. The key pillar in cloud applications development is an efficient and accurate cloud service discovery and selection meeting developers requirements. After conducting a comprehensive investigation of cloud services discovery approaches, we concluded that proposing a discovery and selection assistant based on a developer's query expressed in natural language can greatly simplify and expedite the discovery and selection process. Our discovery assistant DESCA is based on a public cloud service data-set named ULID which is available on (ULID , 2019). ULID services are classified according to their functional features using our clustering algorithm. Indeed, we enhance the basic K-means algorithm to ensure a better cohesion of clusters and a meaningful semantic similarity between them. Our enhancements have contributed to better clusters purity values.

Our ongoing research includes further investigations on supporting more QoS metrics. The crucial challenge is to take into consideration the fluctuation of some QoS metrics between runtime and design time such as throughput and bandwidth. In addition, we plan to extend DESCA to support more cloud providers and propose relevant recommendations based on developers previous searches.

REFERENCES

Abdul Quadir Md, V. V. and Mandal, K. (2019). Efficient algorithm for identification and cache based discovery of cloud services. In *Journal of Mobile Networks and Applications*, pages 1181–1197.

Amazon compute service description (2019). Amazon compute service description. URL: https://aws.amazon.com/ec2/?nc1=h_ls [accessed: 05-01-2020].

Asma Musabah Alkalbani, W. H. and Kim, J. Y. (2019). A centralised cloud services repository (ccsr) framework for optimal cloud service advertisement discovery from heterogenous web portals. In *IEEE Access*, volume 7, pages 128213 – 128223.

Bey, K. B., Nacer, H., Boudaren, M. E. Y., and Benhammadi, F. (2017). A novel clustering-based approach for saas services discovery in cloud environment. In *Proceedings of the 19th International Conference on Enterprise Information Systems*, volume 1, pages 546–553. SciTePress.

Cloud Patterns (2019). Cloud Patterns. URL: http://www.cloudpatterns.org/ [accessed: 05-01-2020].

Davis, J. and Goadrich, M. (2011). The relationship between precision-recall and roc curves. In *Information Retrieval*, pages 233–240.

Elsevier Mendeley Data (2019). Elsevier Mendeley Data . URL: https://www.elsevier.com/authors/author-services/research-data/mendeley-data-for-journals [accessed: 05-01-2020].

Gabsi, H., Drira, R., and Ghezala, H. H. B. (2018). Personalized iaas services selection based on multi-criteria decision making approach and recommender systems. In *International Conference on Internet and Web Applications and Services*, pages 5–12.

Gabsi, H., Drira, R., and Ghezala, H. H. B. (2019). A hybrid approach for personalized and optimized iaas services selection. In *International Journal on Advances in Intelligent Systems*.

Jrad, F., Tao, J., Streit, A., Knapper, R., and Flath, C. (2015). A utility based approach for customised cloud service selection. In *International Journal of Computational Science and Engineering*, volume 10, pages 32–44.

Lin, D. (1998). An information-theoretic definition of similarity. In *International Conference on Machine Learning*, volume 1, pages 296–304.

Lizarralde, I., Mateos, C., Rodriguez, J. M., and Zunino, A. (2018). Exploiting named entity recognition for improving syntactic-based web service discovery. In *Journal of Information Science*, volume 45, pages 9 –12.

Marneffe, M.-C. and Manning, C. D. (2015). The stanford typed dependencies representation. In *Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

Martino, B. D., Pascarella, J., Nacchia, S., Maisto, S. A., Iannucci, P., and Cerr, F. (2018). Cloud services categories identification from requirements specifications. In *International Conference on Advanced Information Networking and Applications Workshop*, volume 1, pages 436–441.

Microsoft Azure Services (2019). Microsoft Azure Services. URL: https://azure.microsoft.com/en-us/services/ [accessed: 05-01-2020].

Pamungkas, E. W., Sarno, R., and Munif, A. (2017). B-babelnet: Business-specific lexical database for improving semantic analysis of business process models. In *Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, volume 15, pages 407–414.

Rajganesh Nagarajan, R. T. and Selvamuthukumaran (2018). A cloud broker framework for infrastructure service discovery using semantic network. In *International Journal of Intelligent Engineering and Systems*, volume 11, pages 11–19.

Rendan, E., Abundez, I., Arizmendi, A., and Quiroz, E. M. (2011). Internal versus external cluster validation indexes. In *International Journal on Advances in Intelligent Systems*, volume 1.

Sun, L., Dong, H., Khadeer, F., Hussain, Hussain, O. K., and Chang, E. (2014). Cloud service selection: State-of-the-art and future research directions. In *Journal of Network and Computer Applications*, volume 45, pages 134–150.

ULID (2019). ULID . URL: http://dx.doi.org/10.17632/7cy9zb9wtp.2 [accessed: 05-01-2020].

Whissel, J. S. and Clarke, C. L. A. (2011). Improving document clustering using okapi bm25 feature weighting. In *Information Retrieval*, volume 14, pages 466–487.

Table 3: Top-4 retrieved services.

Developer queries	Top-4 retrieved services (enhanced BM25)		Top-4 retrieved services (basic BM25)
	Services	Prince per Unit	
Storage capacity	Google Cloud Storage	$0,02 per GB	Oracle Storage IBM EVault
Backup	IBM Object Storage	$0,02 per GB	Google Cloud Storage
Object storage	Oracle Storage	$0.025 per GB	AWS EBS
Data security	AWS S3	$0,04 per GB	IBM EVault
Face recognition	Amazon Rekognition	$0.10 per 1 Minute 1000 face	Oracle Face Detection
Face API	IBM Watson Visual Recognition	$0.002 per General Tagging	Google Api Cloud vision
Compare images	Google Api Cloud vision	$1,50 per 1000 units	Amazon Lex
	Oracle Face Detection	-	IBM Watson Tone Analyzer
Video on-demand streaming	Amazon Kinesis	$0.00944 per GB data ingested	Google Cloud CDN
Live dynamic streaming	IBM Live Event Streaming	$99 per Month	Oracle Digital Convergence
Video encoding	Google Video Intelligence	$0,10 per Minute	AWS Elemental
	Oracle Event Hub	$0.2723 per Minute	IBM Multiscreen streaming
Security identification	Amazon Cognito	$0.00550 per Monthly active users	IBM Activity Tracker
Authentification	IBM App ID	$0.004 USD/Authentication event	Amazon Cloud Directory
Data encryption	Google Cloud IAM	Free	Google Cloud platform security
Access control	Amazon GuardDuty	$1.00 per GB	Oracle Identity