A Centralised Cloud Services Repository (CCSR) Framework for Optimal Cloud Service Advertisement Discovery From Heterogenous Web Portals

ASMA MUSABAH ALKALBANI1, WALAYAT HUSSAIN2, AND JUNG YOON KIM3

1Information Technology Department, College of Applied Sciences, Ibra 511, Oman
2Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, NSW 2007, Australia
3Graduate School of Game, Gachon University, Seongnam 13120, South Korea

Corresponding authors: Asma Musabah Alkalbani (asmam.ibra@cas.edu.om) and Jung Yoon Kim (kjyoon79@gmail.com)

ABSTRACT A cloud service marketplace is the first point for a consumer to discovery, select and possible composition of different services. Although there are some private cloud service marketplaces, such as Microsoft Azure, that allow consumers to search service advertisement belonging to a given vendor. However, due to an increase in the number of cloud service advertisement, a consumer needs to find related services across the worldwide web (WWW). A consumer mostly uses a search engine such as Google, Bing, for the service advertisement discovery. However, these search engines are insufficient in retrieving related cloud services advertisements on time. There is a need for a framework that effectively and efficiently discovery of the related service advertisement for ordinary users. This paper addresses the issue by proposing a user-friendly harvester and a centralised cloud service repository framework. The proposed Centralised Cloud Service Repository (CCSR) framework has two modules - Harvesting as-a-Service (Haas) and the service repository module. The Haas module allows users to extract real-time data from the web and make it available to different file format without the need to write any code. The service repository module provides a centralised cloud service repository that enables a consumer for efficient and effective cloud service discovery. We validate and demonstrate the suitability of our framework by comparing its efficiency and feasibility with three widely used open-source harvesters. From the evaluative result, we observe that when we harvest a large number of services advertisements, the Haas is more efficient compared with the traditional harvesting tools. Our cloud services advertisements dataset is publicly available for future research at: http://cloudmarketregistry.com/cloud-market-registry/home.html.

INDEX TERMS Cloud services discovery, web harvesting, service advertisements, ontology, centralized repository, heterogeneous data.

I. INTRODUCTION

The cloud computing paradigm is a new model of delivering computing resources, such as online applications, storage and networks, as a service over the World Wide Web (WWW). It focuses on sharing IT resources over a scalable network called the cloud [1]. Cloud computing is a multi-domain environment that offers thousands of online services, which makes the discovery of cloud services a complex and multifaceted task. Given the fact that end-users’ requirements vary and that cloud service providers provide a range of cloud services with only slight variations, cloud service selection is a complex yet vital problem to address [2]. The cloud offers three types of service models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). The cloud provider offers these services through cloud service advertising. The term ‘cloud service advertising’ refers to a cloud service description that present via media, which is an essential factor in any marketplace [3], [4]. The cloud service description describes a complete service
offered by the cloud service provider to their consumer, and it includes some elements that add additional value to consumers, such as Quality of Service (QoS) values and technical support details [5]–[7]. The cloud service providers are providing their services offers via their websites, whereas these website schema and layout vary from that provider to another. For example, the service offering template in Microsoft Azure marketplace website is unlike the one presented in the Amazon cloud web marketplace website [8], [9]. Therefore, one of the challenges that the cloud consumers face is to have an optimal cloud service discovery framework to guide them in finding a suitable and trustworthy cloud service from different cloud service advertisement on the web.

Cloud service discovery (CSD) is emerging as a new trend for service discovery across distributed and heterogeneous environments online. It is a process for locating a cloud service that best matches the end-user’s requirements. Since the emergence of cloud technologies, cloud providers advertise their services online, and end-users make use of general search engines such as Bing and Google to discover cloud services [10]. The ability to explore cloud services advertisements across multiple websites becoming a challenge for cloud consumers, mainly when there is a large marketplace of those services. Many Web portals contain up-to-date cloud services advertisements such as getApp. These advertisements can be extracted and analyzed using different web harvesting technique. Cloud consumers usually get confused by the massive number of possibly irrelevant search results due to keyword-based searches.

There are many literatures [10]–[14] in which the authors tried to address the issue. Nabeeh et al. [13] suggested adding a semantic annotation to cloud service profiles online to automate the discovery of cloud services. The objective of using semantic annotation is to allow search engines (such as Google) to semantically identify and retrieve service information based on a user’s objectives [10]. A key issue with the semantic-based approach is that the semantic search could vary depending on the ontology domain and terminologies covered [15]. Alkalbani and Hussain [11] conducted a survey and found that almost all of the existing studies reuse an existing ontology. Such as a business ontology to semantically describe cloud service functions and improve query precision [12], [16]. Constructing an ontology which contains all the relevant domain concepts, such as service classification, service type, etc., is not an easy task, given the fact that cloud providers use different terminologies and vocabularies to describe their service offers, even though they have the same features [14]. Akinwunmi et al. [2] addressed the issue by proposing a decentralized agent system which acts as a consultant for cloud consumers to improve their experiences with cloud services. However, this system, like other approach makes use of a web search engine to find the services, and it is still in the conceptual phase without enough practical applications in the real environment.

Although existing literature has expended a great effort on enhancing search engines techniques with semantic annotations or by developing semantic-based systems for discovery cloud services, however, none of the existing literature discussed a centralised cloud services repository framework that has the capacity to extract, integrate and store cloud services advertisements from semi-structured data located in multiple and poorly organized Web portals into a centralized repository. The centralised repository plays a vital role in an efficient and effective service advertisement discovery in WWW that has been ignored. In this paper, we tried to address the issue by proposing the CCSR framework. The proposed framework presents how to harvest cloud services advertisements from the web portals using our developed harvesting tool - Harvesting-as-a-Service (HaaS). Using HaaS consumers crawl data from the WWW without doing any coding and retrieve relevant details in a fractional of time compared to the traditional methods. The extracted real-time data can be transferred in any format such as CSV, or pdf depending on the choice of a user. The data then analyzed and mapped into a central repository that is used for cloud service discovery.

Significance of the Paper: This study is significant from the following two perspectives. Firstly, the harvesting tool ‘HaaS ‘allows users to extract real-time data from the web without the need to write any code. Secondly, the proposed centralised cloud service repository enables a consumer for efficient and effective cloud service discovery. The centralised repository act as a knowledge source for cloud services. Thirdly, we didn’t find free real cloud dataset on cloud services. This study provides real cloud services dataset and actual cloud reviews that could be used by potential cloud consumers and for future research as well.

The rest of the paper is organized as follows: In Section 2 we discuss and critically analyse the related studies, Section 3 describes the proposed system architecture, Section 4 outlines the system workflow, Section 5 presents the system implementation, Section 6 conducts the system evaluation and presents the results and discussion, and the work concludes with a summary and description of future work in Section 7.

II. RELATED STUDIES
This section aims to provide a background to the state-of-the-art cloud service discovery approaches by studying how these approaches have been examined in the existing literature. The section analyses each of the contexts, features and methods for each of the approaches. We summaries some of the approaches and highlight gaps at the end of the section.

In the current literature, approaches in terms of service selection and composition, trust model [17] and reputation-based approaches [18] are the main approaches in the cloud computing context as well as in other domain such as Web Services and mobile services [19]–[23]. Also, machine learning technologies have been involved in the area of cloud computing to enhance the cloud service level of agreement [21], [24]–[28]. The cloud service discovery process depends on the technique and methods applied and used.
to allocate the service information online. One of the most popular methods for service discovery task across all domains and industries is Google. Google is not, however, restricted to finding service information. Therefore, it may retrieve relevant, irrelevant information depending on the descriptive information provided [29]. In addition, searching for the service information using web search engines depends on two factors: (1) the keyword that best indicates the target service which is the most important aspect in internet marketing and the web search engine’s algorithm; and (2) finding a service that is related to cloud services that best match user’s requirements. The latter can be challenging if the cloud marketplace is vast. In addition, there are a considerable number of websites relating to non-existing services. Parhi et al. [30] proposed a semantic framework for cloud service description based on a multi-agent approach to support the location of cloud services [16]. The proposed framework assist consumer for the discovery of cloud services by referring to shared cloud ontology taxonomies to allocate an appropriate service. Kang and Sim [31] developed a cloud service discovery system (CSDS) that assists cloud users in finding cloud services over the Internet. The framework comprises a user interface and three agents. The model consults a cloud service ontology, which comprises a taxonomy of cloud service concepts. The user interface allows end-users to enter a query which specifies their preferences, including a service name and service requirements. To search for the service over the Internet, the query processing agent uses the existing web search engine - Google. Although the proposed system assists the consumer in finding cloud services, however, the system does not have a centralised repository for effective result.

In another work, Sim [32] proposed a multi-agent cloud service discovery system Cloudle that focuses on matching end-users’ requirements with an advertised cloud service. The proposed system comprises of four self-organising agents that are used to assist users in finding advertised cloud services and managing cloud resources. It considers variations in the types of cloud service resources over the Internet and consults cloud ontology taxonomies in the search process to match services and requirements. One of the shortcomings of this work is that the cloud search engine agent deployed to gather cloud services information uses a web search engine, which is a keyword-based search engine that also retrieves irrelevant information. To overcome the issue of key-word based searching, Rajendran and Swamynathan [33] proposed a model that combines the advantages of a cloud service ontology technique with a multi-agent-based protocol. The approach uses cloud service ontology to discover and retrieve information about cloud service. Although the system has a user-friendly interface that supports the end-user to find appropriate cloud services. However, the approach, like other existing work, make use of search engine, such as Google, to retrieve cloud services from the Internet and lacking the concept of a centralised repository. The idea of the centralised repository proposed by Chen et al. [34]. The framework comprised of two modules - web service description language (WSDL) and web service registry (UDDI) [35]. The approach semantically annotate cloud services using WSDL extension [36] and then store the semantic annotation of the cloud service in a web service registry - UDDI [37]. Although the proposed system provides a solution for dynamic cloud services selection; however, the approach is unable to cope with the growing marketplace and how to update in a real-time manner. Building the concept of a centralised repository, Alkalbani et al. [38] proposed a centralized open-source repository for cloud services. The approach is supported by the Nutch Hadoop Crawler [39], [40] which crawls a Web portal to extract information about cloud services and stores this information in a local repository. The authors only focused on SaaS service data and based on crawling information from one web portal only. To gather a variety of cloud services detail for different services, Gong and Sim [41] developed three versions of a cloud crawler. The proposed crawler gathers cloud service information from three well-known cloud providers - Amazon Web Services [42], Rackspace and GoGrid. The gathered data has two tuples, service specification and service price. Further analysis was applied to the service specification using a K-means clustering algorithm to cluster cloud services in a different category. One of the shortcomings of the proposed approach is that the crawler needs to be customized for every website. It, therefore, failed to crawl cloud services efficiently, since customizing the crawler is a time-consuming process. Also, there are some other studies on mobile service selection discussed the importance of service selection, including quality of service parameters in selecting the mobile services [43]–[45].

The above-discussed approaches tried to enhance cloud service discovery. However, there are many shortcomings in those approaches which are discussed as follows:

- The existing literature did not give an optimal solution of how to extract, integrate and store cloud services details from different semi-structured data spread across the internet.
- The discussed approaches did not provide any solution for customised harvesting tool while collecting cloud services information from heterogeneous web sources to build a comprehensive listing of cloud services.
- The existing literature did not discuss the harvester that extracts real-time data from the web without the need to write any code.
- The discussed approaches did not provide a means for integrating all cloud services information in a central repository for efficient and effective decision making.

To overcome the discussed gaps, we present our proposed CCSR framework, as discussed in Section 3.

III. CCSR FRAMEWORK

In this section, we propose the architecture of the Centralized Cloud Services Repository (CCSR) framework, which acts as a directory for cloud services advertisements and assists in efficient finding for cloud services commercial offerings. The CCSR framework composed of two modules: Harvesting
as-a-Service (HaaS) and the Service Repository, as shown in Fig. 1. The detail of each module is discussed below:

### A. HaaS MODULE

This section presents the design of our Harvesting as a Service (HaaS) module, which harvests cloud services information from targeted Web portals. The proposed HaaS harvest the real-time data/information from the targeted website in a few minutes and make it available in one file. Depending on the choice of a consumer, the file can be available in JSON, CSV, SQL or PDF file formats. Besides, the HaaS has a user-friendly interface that makes the process of harvesting easy for the end-users. By “end-user”, we mean a cloud end-user such as a consumer, organization, or developer who has knowledge about cloud services advertisements and where to find them over the WWW. The HaaS comprised of six sub-modules - Policy Centre, Configuration Manager, Learning Agent, Harvester Agent, Semi-structured Harvested Data, Harvesting Optimizer and Cloud Services Repository.

The sub-modules are explained below:

1) **POLICY CENTRE**

The policy centre defines P1-configuration policy and P2- the polite harvesting policy for coordinating the harvesting process. Initial policies are set in P1 that outlines the boundary for carrying out the harvesting process. It provides details about the structure of the service advertisement in the webpage and the targeted service information to be harvested. The P1 work based on two rules. The first rule assists to choose sample page from the target website. The second rule then identifies the data that need to be harvested from that sample page. In our scenario, we have a list of service advertisement attributes (custom attributes), such as service name, review etc. and its organisation into object attributes such as review date, review name etc. The second policy P2 regulates the maximum time for each harvesting session. To avoid overloading, the harvesting process is divided into multiple sessions, depending on the total number of related webpages. After each session, the process is paused for a certain period before continuing a new session. In this way, the harvested targets website is not overloaded. The user indicates how many links can be harvested per session and indicate how many seconds to wait before moving on to the next session.

2) **CONFIGURATION MANAGER**

The Configuration Manager personalizes the harvesting process and provides essential guidelines for collecting service information form the targeted website. The harvesting process is comprised of two phases: phase one is the setup phase and phase two is the harvesting phase. Phase one is conducted in four steps, using the Configuration Manager user interface. During this setup phase, the Configuration Manager utilizes three levels of the configuration structure: the web page level, the custom object level, and the custom attribute level. The web page level provides a sample Uniform Resources Locator (URL) of a web page, while the custom object and the custom attribute levels define the data targeted for collection. The Configuration Manager consults the Policy Centre to ensure defined rules/policies are followed during the procedure.

3) **LEARNING STRUCTURE AGENT**

The task of the Learning Agent is to learn the web page layout/structure of the target Web portal. We propose an algorithm for learning the HTML structure of a web page, as shown in Algorithm 1. It is a learning algorithm that required the user to determine specific control parameters such as targeted web page URL and the required information from in the targeted web page. Sample metadata for a particular page in the targeted website collected and stored in JSON file format. In the sample date, we need to indicate the following: indicates the URL of the sample page, indicates list of objects of the sample page (S), only get the first 25 characters of a sample string, find all HTML tags which have a specific text, Get name and some attributes of a HTML tag (O), Get name, some attributes of a HTML tag and its position compared with other tags of the same type under a HTML parent tag (a). The output of this algorithm is the metadata structure in JSON format with the core is the configuration data with extended information to navigate the position of attributes...
Algorithm 1 Learning structure Algorithm
- Input: Configuration data in JSON format following the structure:
  - $S = \{\text{multiple, objects} = (o_1, o_2, \ldots, o_n)\}$ is the structure of the configuration, containing a sequence of configured objects, $\text{multiple} \in \{\text{yes, no}\}$ (get one or multiple HTML data for objects of the same structure)
  - $o_i (i \in N) = \{\text{attributes} = (a_1, a_2, \ldots, a_m)\}$ is a detail content of each object including a sequence of configured attributes.
  - $a_j (j \in \{1, 2, 3, \ldots, m\}) = \{\text{sample, multiple, fullText}: \text{sample} \text{ (sample text for the attribute), multiple} \in \{\text{yes, no}\}$ (get one or multiple HTML data of similar HTML sibling tags) and $\text{fullText} \in \{\text{yes, no}\}$ (sample text is in full content or partial content)

Output: Metadata structure in JSON format with the core is the configuration data with extended information to navigate the position of attributes during the harvesting process. The output for objects and attributes is as follow:
- $o_i (j \in N) = \{\text{attributes} = (a_1, a_2, \ldots, a_m), \text{parentTag} = \{\text{position, tagName, className, idName}\} \}$. Position: the position of $o_i$ $\text{tagName}$ in relation to the whole HTML document, $\text{tagName}$: HTML tag name, $\text{className}$: HTML class name, $\text{idName}$: HTML ID name.
- $a_j = \{\text{sample, multiple, fullText, filterTag} = \{\text{position, tagName, className, idName}\} \}$. Position: the position of the $\text{tagName}$ in relation to the $\text{parentTag}$ of the object containing $a_j$, $\text{tagName}$: HTML tag name, $\text{className}$: HTML class name, $\text{idName}$: HTML ID name.

Procedure: Begin Algorithm
For $i = 1$ to $n$
  Fetch the sample of the first attribute $a_1$ in $o_i$
  Compute HTML parent tag of $a_1$ and stores $\text{parentTag}$ in $o_i$ using BeautifulSoup
  For $j = 1$ to $m$
    If $\text{parentTag}$ does not contain $a_j$ then
      Compute HTML parent tag and then store $\text{parentTag}$ in $o_i$ using BeautifulSoup
      Repeat step 4
    End if
  End for
  For $j = 1$ to $m$
    Compute HTML tag of $a_j$ based on computed $\text{parentTag}$ in $o_i$ and stores $\text{filterTag}$ in $a_j$ using BeautifulSoup
  End for
End for

End Algorithm

during the harvesting process. The output for objects and attributes is as shown in the Algorithm 1 Output section.

The output also displayed to the end-user for data validation and modification, as showed in Fig 2. The end-user verification is to ensure that the sample data is correct and complete. Steps 1, 2 and 3 are a recursive process until the end-user-defined harvesting boundary has been reached.

4) WEB PAGE HARVESTER ALGORITHM
The task of this component is to extract meaningful information about cloud services from the Web, as specified by the end-user in the data configuration step. To handle the heterogeneity structured of service information when dealing with a large number of web pages, we define policies within the Policy Centre. The Harvester Agent then carries out the harvesting process based on the defined policies. The algorithm pseudocode for Harvester Agent shows in Algorithm 2. The input in this algorithm is the JSON file output from Algorithm 1, that has sample metadata. It shows a list sample metadata of all sample pages and indicates the pattern of the URL of the sample page. The output of this algorithm is the JSON file that has a list of the harvested information, as presented in Figure 2.

5) SEMI-STRUCTURED HARVESTED DATA
This component responsible for receiving the structured harvested information coming from the Web Page Harvester. This information is structured as a JSON object, using the restrictions specified by the end-user during the configuration phase. At this stage, the file includes harvested information with some redundant service attributes.

6) HARVESTING OPTIMIZER
The objective of this component is to remove redundant service attributes from the harvested information file. These attributes are useful for learning the correct HTML structure of the sample targeted web page. The redundant attributes removed after the learning process. To remove the redundant attributes the user assesses the sample harvested information file, then the user can remove any redundant attribute via pressing a delete button in the user interface. It contains all the cloud service information, including cloud services, offers details such as service Uniform Resource Locator (URL), service name, service type, service category, and details of
consumers’ reviews such as reviewer name and comments. This information is made available in different formats such as CSV, PDF or SQL.

B. SERVICES REPOSITORY MODULE

This component is responsible for storing and mapping the harvested information, which has meaningful information about the cloud services advertisement. To achieve this, we use ontology to represent the knowledge of the stored information. The reason for choosing the ontology is because it provides a shared and common understanding of a cloud services advertisement that communicates between people across different web platforms [46]. For example, Amazon (www.amazon.com) and eBay (www.ebay.com) have used the ontologies in products classification for sales and their features [47]. Therefore, we construct the cloud services advertisement ontology for the purpose of the sale by referring to the NIST classification for cloud services which includes three main categories: SaaS, PaaS, IaaS [48], as shown in Fig 3. This ontology offers a first conceptual of the knowledge of cloud services advertisement. Then, we map and store each cloud services advertisements into a concept in the ontology. We extracted meaningful information into the SQL file that represents the main attributes of each cloud service advertisement. We consider that cloud service advertisement ‘A’ is represented by service metadata M, which describes the general knowledge of the cloud service commercial offer such as service ID, service name and service details. In this work, we utilise the service metadata descriptive information from the relational database, which has the harvested information organised by attributes, such as service name, service description and service category. To identify the service concepts that are relevant to a particular service category concept, [service id, service name, service category, service description, provider link, free trial (yes, no), mobile app (yes, no), rating, starting price, year founded. Service ID: is the URI of the service, which is the reference to the semantically linked concepts. Service Name: is the name of the service, Service Category: is the category to which the service belongs, Service description: is the detailed text description of the service features and facilities. Provider link: is the URL link of the service provider, Starting Price: is the starting price of the service per month, Rating: is the score that a consumer gives to a service after purchasing and using it, Free Trial: indicates if the service is available for free a trial or not, Mobile App: indicates if the service is a mobile application or not.

IV. EVALUATION AND VALIDATION OF CCSR FRAMEWORK

To validate the proposed CCSR framework, we demonstrate a case study shows establishing of cloud services central repository using web harvesting tool (HaaS). Our intention in this case study is to show how the CCSR framework assists in establishing a central repository for cloud services advertisments, which assist in discovering cloud

Algorithm 2 Harvest URLs Algorithm

Input: The algorithm Harvest URLs has three types of input:
1. Metadata structure from the output of Algorithm 1: Structure algorithm
2. A sequence of URLs that have the same HTML structure with the configured Web page. URL = (url1, url2 . . . urlp)
3. Polite harvesting parameters {recordsPerSession, waitingTimeInterval}. recordsPerSession: the number of URLs to be harvested over one session, waitingTimeInterval: the time (seconds) the harvest process pause between each session.

Output: Dataset includes data of configured attributes for all inputted URLs. The dataset is stored in MongoDB and exported to CSV format.

Procedure: Compute the number of sessions NS based on the number of URLs and parameter recordsPerSession. NS = Number of URLs/recordsPerSession. Set session to 1
While session <= NS then
For k = 1 to p
For i = 1 to n
Compute all possible parentTag of oi inside urlk using BeautifulSoup and store these tags into ts
If multiple of oi is “yes” then
Repeat step 12 to step 18 for all parentTag inside ts
Else then
Perform step 12 to step 18 for the parentTag that has the position aligned with o1 position
End if
For j = 1 to m
If multiple of aj in oj is “yes” then
Parse the content of aj for all similar HTML siblings based on parentTag and aj filterTag (tagName, className, idName) using BeautifulSoup and store the data in MongoDB for urlk
Else then
Parse the content of aj for the HTML tag based on parentTag and aj filterTag (position, tagName, className, idName) using BeautifulSoup and store the data in MongoDB for urlk
End if
End for
End for
Pause the process based on waitingTimeInterval
Increment session to 1
End while

End Algorithm
services information. For this study, we consider harvesting cloud services advertisements details from two publicly available web portals, namely getapp.com and serchen.com. To demonstrate the feasibility of our proposed ‘HaaS framework’, we compare it with the other three widely used open-source harvesters – Parsers, Crawly and Scrapy as presented in Table 1. We compare them based on six criteria that we believe are an essential factor for user efficiency and satisfaction. From the comparative analysis, we see that there are many similarities between HaaS and the other harvesters. However, our approach does not need the end-users to code for harvesting, and it provides a user-friendly interface that makes it usable for the end-users to easily employ the harvesting process.

To harvest serchen.com, the HaaS system takes the end-user through some steps discussed as follows:

### A. CONFIGURATION

The purpose of this step is to help users to configure what they want to harvest on their target Web portal. First, the user needs to investigate serchen.com thoroughly to understand the serchen.com sitemap and the layout of serchen.com web pages with repetitive HTML structure that they wish to focus. To set up the configuration, users need to provide the following details on the configuration form. PAGE has Website URL: Original URL of the harvested website and Page URL: Sample page URL to obtain sample data. A page can have one or many objects. OBJECT has Object Name: Custom object name. An object can have one or many attributes, and Object Multiple: This option indicates whether to harvest the same object multiple times. ATTRIBUTE has Attribute Name: Custom attribute name, Attribute Sample: Attribute sample copied from the sample page. If the text is too long, partial text used, Multiple: Indicate whether to harvest the same attribute multiple times (e.g. harvest multiple li in an ul tag) and Full text: Indicate whether the attribute sample is in full text or partial text (check HTML structure. The Add Object button (+ object) inside the page panel is used to add a new empty object to the Configuration page. The Add Attribute button (+ attribute) inside the Object panel for the collection of attributes is used to add a new empty attribute to the Object on the Configuration page. Users can remove Objects/Attributes using the delete buttons. Clicking the Next button takes users to Step 2: View Sample. The screenshot in Figure 7 shows the configuration setup screen for serchen.com based on sample pre-defined data from the system.

### B. VIEW SAMPLE

The purpose of this step is to learn the HTML structure of all configured objects and their attributes in the configured page. It also assists users to validate the results to ensure
that the required data is sampled, per the user’s request at step 1: Configuration. If the data is not correct, users can move back to Step 1 using the Previous button to adjust their configuration of the error object/attribute. If this is the case, users click the button to trigger the system to re-learn the structure and re-get the sample data. Removing redundant attributes to customize the harvested result is carried out in Step 4: Start Harvesting. Users click the Next button to move to Step 3: RELATED LINKS. This step assists users to copy the URLs of all web pages that have the same HTML structure as the configured sample web page from Step 1: Configuration. The cloud services offerings located in a unique web page URL, but all pages have the same structure. The polite harvesting feature implemented in the HaaS system with constraints that include records per session and waiting for a time interval (refer to Section 3). Users click the Next button to move to the next step, Start Harvesting.

Start Harvesting: the users only need to press the Start Harvesting button to start the harvesting process. The system stores harvested information in MongoDB with JSON syntax. When the harvesting process finished, the system generates data in CSV file format and makes it available to the end-user. The web browser pops up another Tab that enables users to download the data in CSV file format (save the file name as `<filename>.csv`). If automatic popups blocked on the user’s browser, the user needs to allow popups for the web tool and re-start harvesting to download the file. The structure of the exported file is as follows: All Attributes which belong to an Objects of multiple values and assigned to No, they are combined and considered as columns in the top table (Main Table of output Datasets). To harvest the getapp.com Web portal, we followed the same steps. Endusers follow the HaaS user interface instructions to harvest the data from the target Web portals.

C. CONSTRUCTING ONTOLOGY AND REPOSITORY
In this step, the Web ontology language (OWL) used to represent the conceptual model and the knowledge of the collected cloud services advertisements. We extracted meaningful information into the SQL file that represents the main attributes of each cloud service advertisement. We consider that cloud service advertisement ‘A’ is represented by service metadata M, which describes the general knowledge of the cloud service commercial offer such as Service ID, Service Name, Service Category, Service Description, Provider Link, Free Trial (yes, no), Mobile App (yes, no), Rating, Starting Price, Year Founded.

<table>
<thead>
<tr>
<th>TABLE 2. Harvesting time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Targeted Web portal</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>serchen.com</td>
</tr>
<tr>
<td>getapp.com</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

D. RESULT
The HaaS was able to generate of 17657 cloud services items and 17337 consumers reviews experience with cloud services as presented in Table 2.

In addition we examine the efficiency of the proposed HaaS system compared to the Parsers, Scrapy and Crawly heterogeneous structured websites. We first harvested the serchen.com Web portal using all of them without applying the polite harvesting feature and compared both tools about crawl time. Tables 3 shows this comparison of harvesting time across three rounds of harvesting. To validate the harvesting results, we measured the percentage of change in the
TABLE 4. Efficiency comparison (in seconds) - getApp website.

<table>
<thead>
<tr>
<th>Harvesting Tool</th>
<th>Number of services harvested</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Harvested Cloud services (1st round)</td>
<td></td>
</tr>
<tr>
<td>HaaS</td>
<td>8.99</td>
</tr>
<tr>
<td>Scrapy</td>
<td>7.48</td>
</tr>
<tr>
<td>Crawly</td>
<td>8.03</td>
</tr>
<tr>
<td>Parser</td>
<td>8.52</td>
</tr>
<tr>
<td>Harvested Cloud services (2nd round)</td>
<td></td>
</tr>
<tr>
<td>HaaS</td>
<td>8.2</td>
</tr>
<tr>
<td>Scrapy</td>
<td>7.64</td>
</tr>
<tr>
<td>Crawly</td>
<td>8.65</td>
</tr>
<tr>
<td>Parsers</td>
<td>10.83</td>
</tr>
<tr>
<td>Harvested Cloud services (3rd round)</td>
<td></td>
</tr>
<tr>
<td>HaaS</td>
<td>14.7</td>
</tr>
<tr>
<td>Scrapy</td>
<td>13.3</td>
</tr>
<tr>
<td>Crawly</td>
<td>18.6</td>
</tr>
<tr>
<td>Parsers</td>
<td>19.04</td>
</tr>
</tbody>
</table>

The average efficiency of three rounds:

<table>
<thead>
<tr>
<th>Harvesting Tool</th>
<th>Harvested Cloud services (1st round)</th>
<th>Harvested Cloud services (2nd round)</th>
<th>Harvested Cloud services (3rd round)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HaaS</td>
<td>10.43</td>
<td>18.83</td>
<td>24.62</td>
</tr>
<tr>
<td>Crawly</td>
<td>11.76</td>
<td>24.73</td>
<td>30.04</td>
</tr>
<tr>
<td>Parsers</td>
<td>12.92</td>
<td>25.6</td>
<td>30.12</td>
</tr>
</tbody>
</table>

TABLE 5. Polite harvesting details.

<table>
<thead>
<tr>
<th>Harvesting Tool</th>
<th>Record per session</th>
<th>Waiting time interval (in secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrapy</td>
<td>10 pages</td>
<td>3</td>
</tr>
<tr>
<td>Crawly</td>
<td>10 pages</td>
<td>4</td>
</tr>
<tr>
<td>HaaS</td>
<td>10 records</td>
<td>3</td>
</tr>
<tr>
<td>Parsers</td>
<td>10 pages</td>
<td>6</td>
</tr>
</tbody>
</table>

The harvesting time as a function of the number of harvested services (20, 40, 60, 80 and 100). The harvesting results of serchen.com show that the proposed HaaS tool performs better than Parsers, Scrapy and Crawly.

The harvesting time usually depends on network bandwidth, CPU capacity at the time of running, server response time at the time of running, and the polite harvesting configuration for all HaaS, Scrapy, Parsers and Crawly. Tables 4 presents the comparison of harvesting time for HaaS, Crawly and Scrapy across three rounds. We next harvested getapp.com using HaaS, Crawly, Parsers and Scrapy. We applied the polite harvesting feature for both tools, as presented in Table 5.

Also, we tested the quality of the harvested data using our approach compared with the tradition tool, such as scrapy. We have compared the results of harvesting 100 cloud services from serchen.com using Scrapy and HaaS. The Fig 4 and Fig 5 present the harvested data in two columns service URL, service name.

The results indicate that significant values were missing from the service description column in the case of the Scrapy results. By “missing value”, we mean that the corresponding value was not present in the harvested data as presented in Figure 6 and Figure 7. There are 14 missing service name values out of 100 harvested services, with a successful harvesting quality rate of 86%. Interestingly, there are no missing values in the HaaS file (Fig 5), indicating the successful harvesting quality rate of 98%.

We used precision, recall, and F-score methods to tag extractions about the instantiated concepts in the knowledge base as presented in the below equations:

\[
\text{Precision} = \frac{\text{number of instances correctly acquired}}{\text{number of instances acquired}}
\]

\[
\text{Recall} = \frac{\text{number of instances correctly acquired}}{\text{number of instances existing in the conceptual tree}}
\]

\[
F - \text{measure} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}
\]

The harvested 17806 service entities are collected from three different web portals in the repository using the equations mentioned above. Table 5 presents the results of the set of individuals and concepts extracted from the repository to map and populate the cloud service advertise (CSA) ontology. A set of 17806 individual mappings according
to the conceptual tree CSA produced. Of these concepts, 17,793 correctly instantiated by the rules of the classification (SaaS, PaaS and IaaS) and 13 not instantiated. We thus obtain a recall of 99.59 and precision of 98.98. These accuracy measures are shown in Table 8.

**V. CONCLUSION**

In this study, we have presented a service-based harvester called *Harvesting as a Service* (HaaS) for crawling cloud services information from various structured Web portals. The critical contribution of the proposed system is the HaaS with a friendly user interface, which allows end-users to harvest websites without the need for developers or coding, unlike other traditional harvesting tools. Experiments were carried out, and the results show that compared to the traditional tool, our proposed approach demonstrates a significant improvement in the harvesting time quality when the number of harvested pages increased. Also, we used an ontology for storing and representing the harvested data. Future work will consider continuing harvesting of the various web portals to enrich the cloud service advertisements ontology.

**REFERENCES**


[16] Y. Yin, L. Chen, Y. Xu, and J. Wan, “Location-aware service recommenda-


