LARGE-SCALE ELASTIC ARCHITECTURE FOR DATA AS A SERVICE

Project Number:	FP7-ICT-318809
Project Title:	Large-Scale Elastic Architecture for Data as a Service
Deliverable Number:	D5.6
Title of Deliverable:	Web graph service for multiple micro-clouds
Contractual Date of Delivery:	2015.04.30
Actual Date of Delivery:	2015.04.29

Abstract

This deliverable presents the Web Graph Service (WGS) for multiple micro-clouds prototype developed by the LEADS platform. This evolution of the WGS prototype exemplifies the usage and tests the capabilities of the LEADS infrastructure running across multiple micro-clouds.

List of Contributors

Name	Organization	E-mail
Jacques Ohannessian	adidas	Jacques.Ohannessian@adidas-Group.com
Pawel Skorupinski	adidas	Pawel.Skorupinski@adidas-group.com
Wojciech Barczynski	Cloud & Heat	wojciech.barczynski@cloudandheat.com
Marcel Gädig	Cloud & Heat	marcel.gaedigk@cloudandheat.com
Jens Struckmeier	Cloud & Heat	jens.struckmeier@cloudandheat.com
Anja Strunk	Cloud & Heat	anja.strunks@cloudandheat.com
David Garcia Soriano	BM-Y!	davidgs@yahoo-inc.com
Kourtellis Nicolas	BM-Y!	kourtell@yahoo-inc.com
Timothy Potter	BM-Y!	tep@yahoo-inc.com
Hossein Vahabi	BM-Y!	puya@yahoo-inc.com
Emmanuel Bernard	Red Hat	ebernard@redhat.com
Jonathan Halliday	Red Hat	jonathan.halliday@redhat.com
Mark Little	Red Hat	mlittle@redhat.com
Mircea Markus	Red Hat	mmarkus@redhat.com
Pedro Ruivo	Red Hat	pedro@infinispan.org
Aggelos Aggelidakis	TSI	aaggelidakis@softnet.tuc.gr
Eleftherios Chatzilaris	TSI	echatzilaris@softnet.tuc.gr
Antonios Deligiannakis	TSI	adeli@softnet.tuc.gr
Ioannis Demertzis	TSI	idemertzis@softnet.tuc.gr
Minos Garofalakis	TSI	minos@softnet.tuc.gr
Odysseas Papapetrou	TSI	papapetrou@softnet.tuc.gr
Evagelos Vazeos	TSI	vagvaz@softnet.tuc.gr
Christof Fetzer	TUD	christof.fetzer@tu-dresden.de
André Martin	TUD	andre.martin@tu-dresden.de
Do Le Quoc	TUD	do@se.inf.tu-dresden.de
Jons-Tobias Wamhoff	TUD	jons@inf.tu-dresden.de
Pascal Felber	UniNE	Pascal.Felber@unine.ch
Raluca Halalai	UniNE	Raluca.Halalai@unine.ch
Marcelo Pasin	UniNE	Marcelo.Pasin@unine.ch
Etienne Rivière	UniNE	Etienne.Riviere@unine.ch
Valerio Schiavoni	UniNE	Valerio.Schiavoni@unine.ch
Anita Sobe	UniNE	Anita.Sobe@unine.ch
Pierre Sutra	UniNE	Pierre.Sutra@unine.ch

Document Approval

	Name	Email	Date
Approved by WP Leader	Jens Struckmeier	jens.struckmeier@cloudandheat.com	2014-04-29
Approved by GA Member 1	Etienne Riviere	Etienne.Riviere@unine.ch	2014-04-29
Approved by GA Member 2	Minos Garofalakis	minos@softnet.tuc.gr	2014-04-28

Contents

List of Figures

1. Introduction

The Web Graph Service (WGS) allows users to access and analyze the Web graph data collected by LEADS and the associated content by providing an open querying interface to users and responding with graphical results to these queries. This deliverable summarizes the efforts in LEADS for the implementation of a WGS prototype that uses the LEADS platform running on multiple micro-clouds. Note that the previous delivery's [10] goal was the implementation of a WGS prototype that used the platform running on a single micro-cloud. The prototype is envisaged to serve two main purposes: i) enabling an early end-to-end integration testing of the components that form the LEADS infrastructure albeit on multiple micro-clouds setting, ii) showcasing possible use-cases of the LEADS infrastructure by providing access to public data crawled, stored, and processed by the infrastructure. In other words we aim to test if all the WGS functions are working within multiple micro-clouds context, while providing a "debug" WGS version that can show details such as micro-clouds that stores a particular URL. These details are useful for experts that want to know more about the platform behavior.

The WGS prototype showcases all main components of the LEADS infrastructure on multiple micro-clouds. Specifically, within the prototype, publicly available Web content and the associated user-generated content and interconnected Web graph (e.g., Web content, inter-connected Web graph, and Web content enriched by user-generated content) is collected using the distributed data collection mechanisms developed in the work package WP1. Collected data is stored using the distributed data storage systems developed in the work package WP2. Data placement decisions for efficient and cost-aware placement and processing of the data are made by the technologies developed in work package WP4. LEADS platform provides WGS access to the publicly available data it collected via open APIs provided by WP3.

This document presents the high-level design and the components of the WGS prototype in LEADS. Section 2 presents the WGS, the supported queries and their business cases. Section 3 briefly describes the components that compose LEADS, explaining each part of the LEADS platform, running on multiple micro-clouds to support the WGS prototype. It provides an overview of how the Web graph is collected, stored, maintained in micro-clouds, how it is queried via the APIs, and how the system determines which micro-clouds to access in the system for a given data processing task. Section 4 summarizes this document.

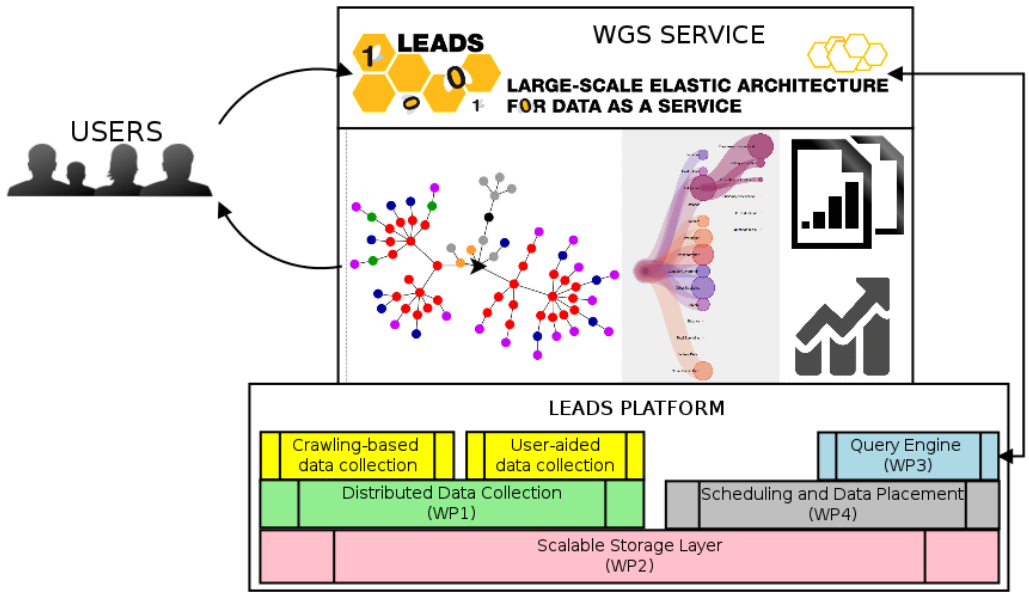


Figure 1: WGS prototype

2. WGS high-level design

As shown in Figure 1, the WGS prototype sits on top of the LEADS platform. Note that the main difference between this delivery and the previous [10] is that now we have multiple micro-clouds, so WGS will need to show and to deal with new attributes such as micro-cloud that store a particular URL. WGS offers an interface for querying the LEADS data collection including the metadata constructed from this collection by sending queries to the LEADS Query Engine and presenting graphical representations of the publicly available data collected and served by the platform. A sample usage scenario and its business cases, which are provided by Yahoo, will be depicted in detail in this section. The aim is always to check if the new implementation of multiple micro-clouds is working correctly and is able to generate useful results also when need to deal with several micro-clouds instead of a single one.

2.1 WGS queries

WGS currently demonstrates two different data retrieval APIs:

- Basic data retrieval API: Return base data associated with the web pages stored in the Web graph, e.g., out-links, PageRank/importance score of a page, etc.
- Public data processing API: Process publicly available data in the Web graph and return the results to users. Such processing may include finding all the Web pages that satisfy a given set of predicates (e.g., raw mention of a term, positive or negative opinion on a term, etc.) by the user, finding the top-k frequent keys (terms, entities, etc.) appearing in a set of pages, finding top-k frequent keys over sliding windows (e.g. last week, last minute, etc.), and computing the PageRank of a user specified set of Web pages.

The functionalities available in WGS are detailed with output examples, and technical descriptions in the following sections. The functionalities presented in the following queries are implemented and being tested at the Cloud&Heat micro cloud at Dresden and are exposed to public use via open access URLs. The access URL is available in the reviewers' section of the project Website, and upon request by email for any reader who would wish to test the functionalities.

2.1.1 Basic data retrieval queries

The simplest query offered by the WGS system is primarily a key/value querying mechanism built on top of the Query Engine, which basically acts as a proxy to the KVS interface. Note that even that we are in a multiple micro-clouds context, the main protocol does not change for the end user. The purpose here is to adapt the previous code so it can work properly in the case of multiple micro-clouds and to check if the results generated are correct with respect to the single micro-clouds case.

Similarly to previous delivery [10], we show how users can query the preprocessed meta-data constructed by LEADS for the publicly available Web resources. An example query pattern, which requests the sub-tree reachable from a URL is defined below.

Given a {URL, depth } tuple,
display the Web graph presenting pages reachable from the given URL within the given depth.

Figure 2. depicts the result of a basic data retrieval (in the multiple micro-clouds case) query that visualizes all pages reachable from *www.yahoo.com* up-to a depth of three.

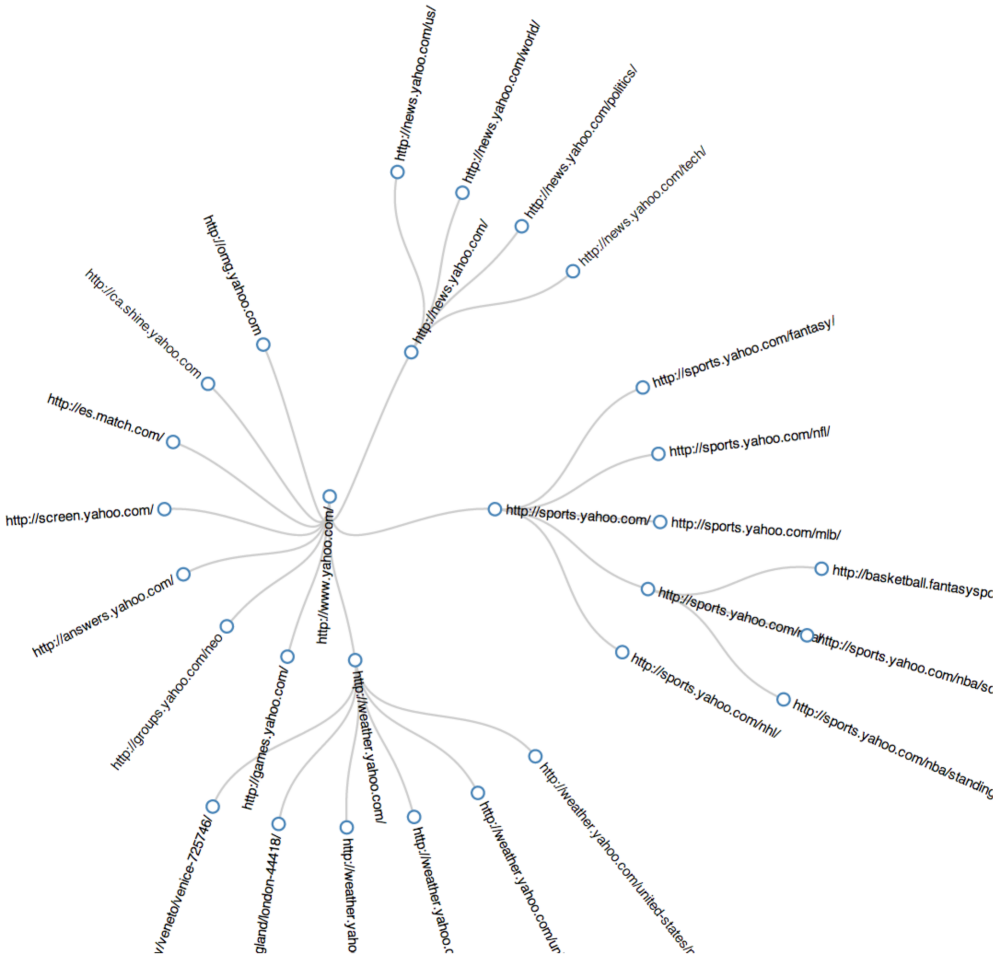


Figure 2: A sample basic data retrieval API output

Figure 3 depicts the results of the data retrieval (in the multiple micro-clouds case) of the query *www.yahoo.com* up-to a depth of three with a tree based visualization..

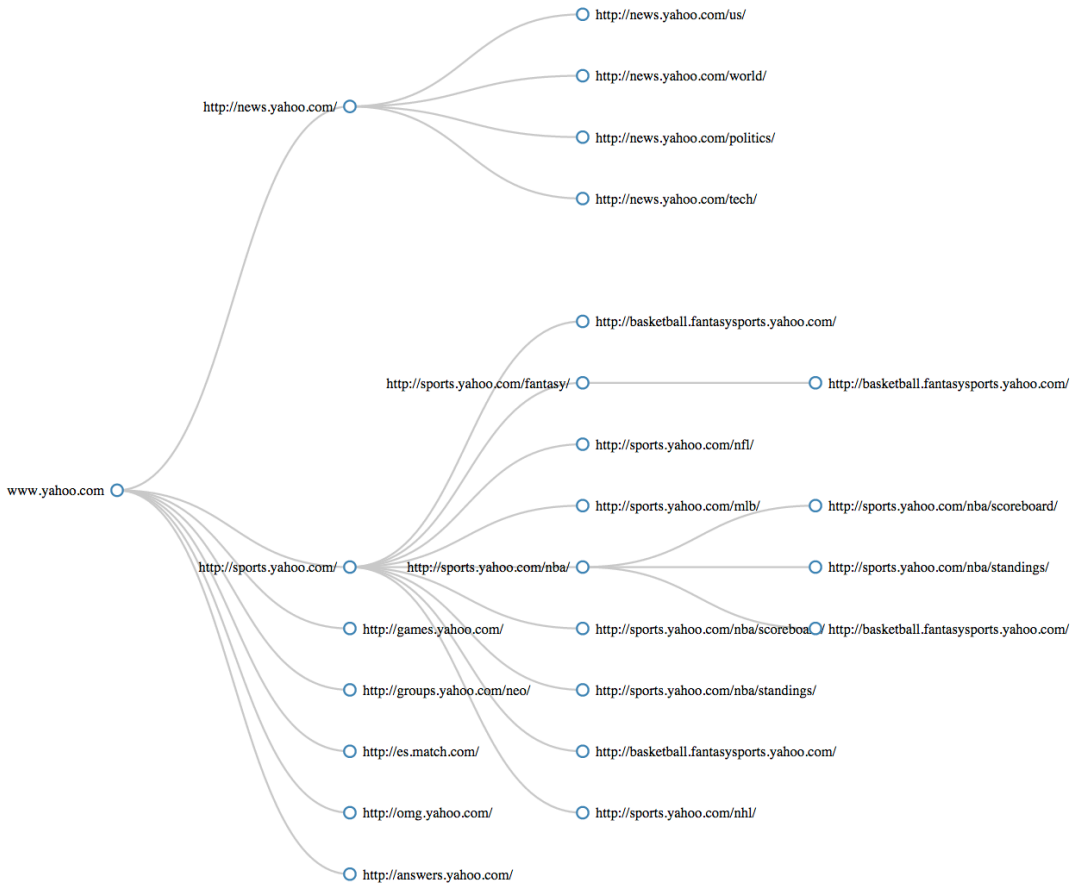


Figure 3: A Tree-Based sample basic data retrieval API output

2.1.2 Public data processing queries

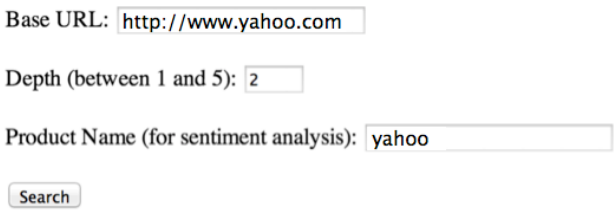
The aim of this deliverable is primary to validate the behavior and functionalities of the LEADS platform running on multiple micro-clouds. In this section we give a brief overview of the public data processing aim (as described in the previous delivery [10]) and we present a plot where we show how the data of a particular query is distributed among the micro-clouds (Figure 4 c).

The aim of the public data processing is to extract useful information from public data. We provide here a simple Web Graph Service that users or experts can use to issue queries about a particular URL of interest, or product of interest. These are structured queries (i.e., string plus other type of information). Similarly to the previous delivery a sample query pattern currently supported by WGS is given below.

Given a {URL, depth, product name} tuple,
display the Web graph presenting pages reachable from the given URL within the given depth,
where vertices of the graph are coloured according to the sentiment on these pages about the given product,
the sizes of the vertices vary in concordance with the PageRank values of the pages they represent,
and the vertices are labelled with the URL string and PageRank of the pages they represent.

This query pattern showcases many of the basic functionalities exposed by the LEADS platform while presenting a meaningful use case for customers. Using this query pattern, it is possible to visualize the flow/distribution of sentiment from a given URL.

Figure 4 depicts the query *http://www.yahoo.com* with depth 2 and we want to know the sentiment of people with respect to the Yahoo brand name. This is a very useful use case to know better what users think about particular pages of Yahoo. Figure 4 (b) just a part of the JSON results of the submitted query. It is possible to see a new field "micro-cluster" which represents the micro-cloud that contains the information related to that URL. The aim of this experiments was to assess that the WGS multiple micro-clouds is working properly and is able to collect data in a distributed environment, get the results and show them.



(a) A sample WGS query

```
json:  {
    "pageRank": "1.00000",
    "micro-cluster": "hamm5",
    "name": "https://www.yahoo.com",
    "sentiment": "1.4000000000000001",
    "children": [
            ...
        {
            "pageRank": "0.70124",
            "micro-cluster": "hamm5",
            "name": "https://www.yahoo.com/health",
            "sentiment": "-0.5",
            "children": [
                {
                    "pageRank": "0.40077",
                    "micro-cluster": "hamm5",
                    "name": "https://www.yahoo.com/health/the-green-drink
                    "sentiment": "2.6999999999999993"
                },
                {
                    "pageRank": "0.40077",
                    "micro-cluster": "hamm6",
                    "name": "https://www.yahoo.com/health/10-must-have-healthy
                    "sentiment": "3.599999999999999"
                }
            ]
        }
            ...
    ]
}
```


(b) JSON result (accessible via a RESTful interface) for the given query in the multiple micro-clouds context

(c) WGS output for the given query

Figure 4: A sample public data retrieval API output – in the case of multiple micro-clouds

To provide a better understanding of the global system behaviour and its components' interactions, we describe some of the steps for this use case:

Periodically, crawlers collect pages from the Web. Crawlers store the fetched pages and the Web graph in the Key-Value Store (KVS) by calling the API of the KVS. When a client registers a query similar to the query defined above, the query is forwarded to the Query Planner. The Query Planner, analyses the query, creates two sub-queries. The first part is persistent, and is implemented with a KVS Listener. The second part is instantaneous, and is implemented with the Query Executor.

The first part, based on a listener, will be running permanently in the system in order to pre-populate a list of pages that match the selection criteria. In our second example, the selection criteria are that the page must contain the word “Yahoo”. Upon receiving a request from the Query Planner, the KVS installs the Listener code, adding it to the list of existing Listeners. After the installation, any page written to the KVS will be forwarded to the new Listener.

The second part, based on query executor, reads the indexed data executes the necessary computations and puts the results back to the KVS to be returned to the query interface.

The WGS, upon receipt of the results, draws a graph based on the contents of the result using the d3.js JavaScript library[1].

2.2 Business cases for WGS queries in multiple micro-clouds

The WGS is currently implemented and designed to test the integration of the LEADS features. However these functionalities are evaluated using a business prospective. The main aim of this delivery was to understand and check if all the functions of LEADS are working in a distributed context. This requires a huge effort in order to understand and test all the functionalities that were previously available in a single micro-cloud context. The development of WGS multiple micro-clouds is of great importance during the development and in future for professionals that want to extend the framework.

2.2.1 Business case for basic data retrieval queries

Basic data retrieval queries, which shows the web graph without any additional information and visualizes the paths between the pages, can be used by the companies to evaluate a graph of their site and compare with the competitors in order to see how many clicks the user would need in order to get from the home page to some specific page. We assessed this feature during this delivery to check the behaviour in the multiple micro-clouds context.

2.2.2 Business case for public data processing queries

Public data that are constantly served and analysed by LEADS platform will increase constantly. In this context our solution to deal with the amount of data was to have multiple micro-clouds framework that is able to process this amount of data and to perform analysis over a distributed system, as it is a single micro-cloud. We have tested and shown how the multiple micro-clouds are working.

The visualization we have implemented is very interesting considering cases where a professional wants to understand better the underlying framework to extend the platform. This has been shown also in [10]. Here we assess that the visualization is perfectly coherent with the single micro-cloud case.

[1] http://d3js.org/ - Library released under BSD license.

2.3 WGS multiple micro-clouds and LEADS Query Engine interaction via RESTful API

The LEADS Query Engine exposes a number of RESTful API calls to LEADS users and WGS makes use of these APIs to implement its services. In this delivery we extended those API to be able to work in multiple micro-clouds context, and we assess that the results do not change with respect to a single micro-cloud system. Below we provide the interfaces of the API calls used by WGS for serving basic data retrieval and public data processing queries. We also briefly describe the processes behind these calls and how WGS makes use of these calls.

2.3.1 Query Engine REST interface for basic data retrieval queries

The publicly available Web resources are stored into LEADS Key Value Store using the following RESTful call to the Query Engine. The input format is similar in single and multiple micro-clouds context.

```
request url: @POST application_url/rest/object/put/ request
json:
{
      "table":"cacheName",
      "key":"key_of_object",
      "object":"string representation of the object"
}
```

The query engine expects as parameters the name of the table that the object would be stored, the key that should be used to store the object and the JSON representation of the object. Note that in general metadata associated with Web resources are stored in KVS using their URL address as keys. Stored metadata can be queried by the following call:

```
request url: @POST application_url/rest/object/get/
```

request json:

```
{
      "table":"cacheName",
      "key":"key_of_object",
      "attributes":["attribute_1","attribute_2"]
}
```

response json:

```
{
      "attribute_1":"value_1",
      "attribute_2":"value_2"
}
```

Using the above call, WGS can query about the basic properties and metadata collected by the LEADS platform. To serve graph-like results, WGS makes recursive calls to the Query Engine (this is again similarly to single micro-cloud case). The call takes three parameters i) the name of the table from which the object should be read, ii) the key of the object, and iii) a list with the names of the attributes that should be returned. The response contains a map with the name of the attributes and their values.

2.3.2 Query Engine REST interface for public data processing queries

Upon receipt of a public data processing query from a client, WGS makes the following call to the LEADS Query Engine:

```
@POST
appl_url/rest/query/wgs/rec_call
Request json: {“url”:”input_url”,”depth”:”3”, “user”:”username”}
Response json: {“queryId”:”id_of_the_query”,”output”:”output_table_name”}
```

The above call finds all the URLs that have depth equal or less than the input depth parameter from the given URL. Specifically, the query engine starts a breadth first search starting from the input URL for depth equal to the depth parameter. Additionally, for each output URL, the query processor computes the current PageRank value and the overall sentiment of that web page. Furthermore we have also the micro-cloud that gives us the response. This is clearly due to the fact that we are in a multiple micro-cloud scenario.

All URLs with equal depth are stored as a list to the output table under the same key. So, with key “0”, the input URL is stored, with key “1” all its outgoing links with their PageRank and overall sentiment are stored, and so forth, and fifth. WGS reads the respective keys from the output table using the RESTful API, which returns the attributes of an object stored in an Infinispan cache.

The example below depicts the set of requests that WGS will make to retrieve the results of a query with depth equal to 3:

depth 0:
application_url/rest/object/get/ json{“table”:””output_cache_name”,”key”:”0”,”attributes”:”result”}
depth 1:
application_url/rest/object/get/ json{“table”:””output_cache_name”,”key”:”1”,”attributes”:”result”}
depth 2:
application_url/rest/object/get/ json{“table”:””output_cache_name”,”key”:”2”,”attributes”:”result”}
depth 3:
application_url/rest/object/get/ json{“table”:””output_cache_name”,”key”:”3”,”attributes”:”result”}

The returned responses to these calls are lists of JSON objects with the following structure

```
{
      “url” : “the webpage url”,
      “pagerank”: “the pagerank of url”
      “sentiment”: “the sentiment of the webpage”
      “links”  : “outgoing links”
      “micro-cluster”: “micro-cluster url coming from”
}
```

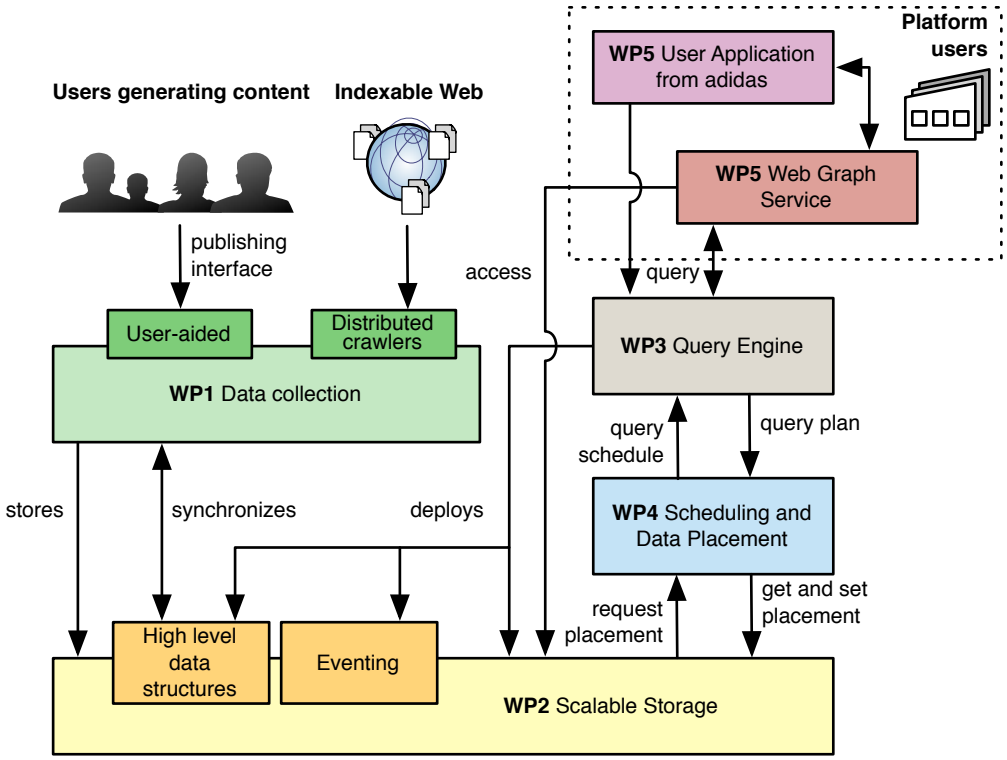


Figure 5: LEADS platform architecture

3. LEADS platform high-level design

Figure 5 displays the high level design of the LEADS platform. This is similar to WGS in single micro-cloud context. As seen in the figure, in LEADS platform, users and applications can submit queries to the system through the Query Engine (using the open APIs or the Web-based UI). The Query Engine parses the query and extracts a set of possible query plans. All plans are forwarded to the Scheduling component, where the execution cost is estimated for each plan, e.g., based on the cost of accessing and transferring the data required by the plan, and the cost of executing the plan. The plans annotated with the cost estimates and scheduling decisions (i.e., the execution location of each operator) are returned to the Query Engine, and the plan with the minimum cost is selected for deployment. Deployment of the plan may involve the extraction of a set of listeners, and the deployment of operators to multiple system nodes. Figure 5 also displays the means of data collection available to the system, which include crawler-based and user-aided mechanisms. The components composing the LEADS are further detailed below.

3.1 Distributed data collection

Distributed data collection for WGS is performed through two different mechanisms: crawler-based data collection and user-aided (browser-based) data collection.

The crawler-based data collection mechanism relies on Web crawlers to fetch content from the Web. Each crawler discovers and fetches Web pages from the Internet by following the link structure of the pages that have been fetched. In order to scale not only across but also within micro clouds, LEADS uses the Nutch[2] web crawler. Nutch is based on Hadoop[3].

[2]https://nutch.apache.org

The user-aided data collection mechanism allows users to publish, in near real-time, the Web pages that they are visiting or creating, to the storage system through the user-publishing interface. Users can also provide additional metadata such as tags, reviews, etc. to pages that can be leveraged for data analysis. The user-publishing interface is designed as a plug-in for popular Web browsers. The data collection in the WGS prototype utilizes both the crawler-based data collection mechanism hosted on the target micro-cloud and the user-aided data collection mechanism by allowing users publish Web pages to the storage system hosted on the target micro-cloud. Further details of the distributed data collection system can be found in [5, 6].

3.2 Distributed data storage

Distributed data storage of WGS is performed by a KVS with extended capabilities to be deployed in a federation of micro-clouds. At the scale of a micro-cloud, the storage layer consists of Infinispan[4], an open source KVS. The core of Infinispan is a specialised shared data structure, tuned to and geared for a great degree of concurrency. Through the data storage API, a client application can *create* and *remove* a *key space*, and can *put* and *get* a key with an associated value into a particular key space. In addition, the storage layer supports a *listener* pattern via a client registration mechanism that takes a key space as argument. Once registered, a client receives notifications for the events occurring on the registered keys. The storage layer implements failure detection mechanisms and supports the *addition* and *removal* of nodes (elasticity). Storage layer in a micro-cloud also uses an internal mechanism for balancing the storage load between its servers. The storage layer is able to return the location(s) of a data item, and to receive data placement requests. Further details of the distributed data storage system can be found in [1, 7].

3.3 Query processing

The query-processing layer of WGS focuses on efficiently executing user queries. The main subcomponents are: (a) the query description interface, which enables users to define their information needs in a flexible and straight-forward approach, (b) the query planner, which derives efficient query execution plans across multiple micro-clouds, (c) the query deployer, which deploys, coordinates, and monitors the query execution at the micro-clouds, and, finally, (d) the node query executor, which runs at nodes in a micro-cloud and handles the actual query execution.

The tasks of query processing component include indexing of the resources in order to enable efficient answering of user-defined queries (e.g., all webpages containing the term ‘sports’), computing statistics frequently required for query execution (e.g., PageRank), and other tasks specifically defined by the user to satisfy special requirements.

The query-processing component exposes its capabilities via a RESTful API that enables executing simple SQL-like queries. The platform also includes a novel streaming PageRank algorithm developed specifically for the LEADS infrastructure, which enables continuous maintenance of the PageRank scores of all crawled webpages. The algorithm is fully integrated with the KVS, receiving and processing all crawler updates as a distributed stream. Further details of the query processing system can be found in [3, 8].

---

[3] http://hadoop.apache.org
[4] http://infinispan.org

3.4 Scheduling & data placement

The scheduling layer of WGS is transparent to users. It schedules the virtual machines running crawler and data processing tasks with the objective of minimizing the costs of storing, transferring, as well as processing data items. It utilizes a RESTful API for the interaction of the individual LEADS components with the scheduler. In order to cope with workload fluctuation, it provides vertical scaling mechanisms that allow dynamic allocation of resources to virtual machines on demand without imposing reconfiguration overhead at the application level. Further details of the scheduling and data placement system can be found in [2, 9].

4. Summary

This document presents the Web graph service of LEADS, which provides graphical access to public data collected from the Web by the LEADS platform. It also demonstrates the high-level design of the key components that support WGS service. WGS tests the capabilities of the LEADS infrastructure on "multiple" micro-clouds.

5. References

[1] WD2.1 of LEADS: Data storage requirements specification. 2013.

[2] WD4.1 of LEADS: Design document for scheduling and data placement. 2013.

[3] WD3.1 of LEADS: Specification of the LEADS real-time processing platform and tools. 2013.

[4] WD5.1 of LEADS: Web Graph Service Specification. 2013.

[5] WD1.1 of LEADS: Specification of Distributed Data Collection. 2013.

[6] D1.2 of LEADS: Real-time content discovery through on-the-fly publishing. 2013.

[7] D2.2 of LEADS: Initial prototype of the key-value store, 2013.

[8] D3.2 of LEADS: LEADS real-time processing platform, 2013.

[9] D4.2 of LEADS: First prototype of scheduling and data placement with vertical agile elasticity, 2013.

[10] D5.3 Web Graph Service Prototype 2014.