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:	D5.3
Title of Deliverable:	Web Graph Service Prototype
Contractual Date of Delivery:	2014.03.31
Actual Date of Delivery:	2014.03.30

Abstract

This deliverable presents the Web Graph Service (WGS) prototype developed by the LEADS platform. WGS prototype exemplifies the usage and tests the capabilities of the LEADS infrastructure on a single micro-cloud.

List of Contributors

Name	Organization	E-mail
Jacques Ohannessian	adidas	Jacques.Ohannessian@adidas-Group.com
Pawel Skorupinski	adidas	Pawel.Skorupinski@adidas-group.com
Xiao Bai	BM-Y!	xbai@yahoo-inc.com
Diego Marron	BM-Y!	diegom@yahoo-inc.com
Tim Potter	BM-Y!	tep@yahoo-inc.com
Ata Turk	BM-Y!	ata@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
Antonios Deligiannakis	TSI	adeli@softnet.tuc.gr
Ioannis Demertzis	TSI	idemertzis@softnet.tuc.gr
Minos Garofalakis	TSI	minos@softnet.tuc.gr
Nikolaos Giatrakos	TSI	ngiatrakos@softnet.tuc.gr
Ekaterini Ioannou	TSI	ioannou@softnet.tuc.gr
Odysseas Papapetrou	TSI	papapetrou@softnet.tuc.gr
Nikolaos Pavlakis	TSI	npavlakis@softnet.tuc.gr imper-ros@softnet.tuc.gr
Ioakim Perros	TSI	
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
Frezewd Lemma Tena	TUD	Frezewd_Lemma.Tena@mailbox.tu-dresden.de
Pascal Felber	UniNE	Pascal.Felber@unine.ch
Marcelo Pasin	UniNE	Marcelo.Pasin@unine.ch
Etienne Rivière	UniNE	Etienne.Riviere@unine.ch
Valerio Schiavoni	UniNE	Valerio.Schiavoni@unine.ch
Pierre Sutra	UniNE	Pierre.Sutra@unine.ch

Document Approval

	Name	Email	Date
Approved by WP Leader	AoTerra	jens.struckmeier@aoterra.de	2014-03-30
Approved by GA Member 1	Christof Fetzer	Christof.Fetzer@tu-dresden.de	2014-03-25
Approved by GA Member 2	Minos Garofalakis	minos@softnet.tuc.gr	2014-03-21

Contents

List of Figures

GLOSSARY

EU	European Union
FP7	Seventh Framework Programme
RSS	Really Simple Syndication
URL	Uniform Resource Locator
DNS	Domain Name Service
WGS	Web Graph Service

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 runs 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 a simpler, single micro-cloud setting, ii) showcasing possible use-cases of the LEADS infrastructure by providing access to public data crawled, stored, and processed by the infrastructure.

The WGS prototype showcases all main components of the LEADS infrastructure on a single micro-cloud. 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 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 in WGS, and their business cases. Section 3 briefly describes the components that compose LEADS, explaining each part of the LEADS platform, running on a single micro-cloud to support the WGS prototype. It provides an overview of how the Web graph is collected, stored, maintained in the micro-cloud, 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.

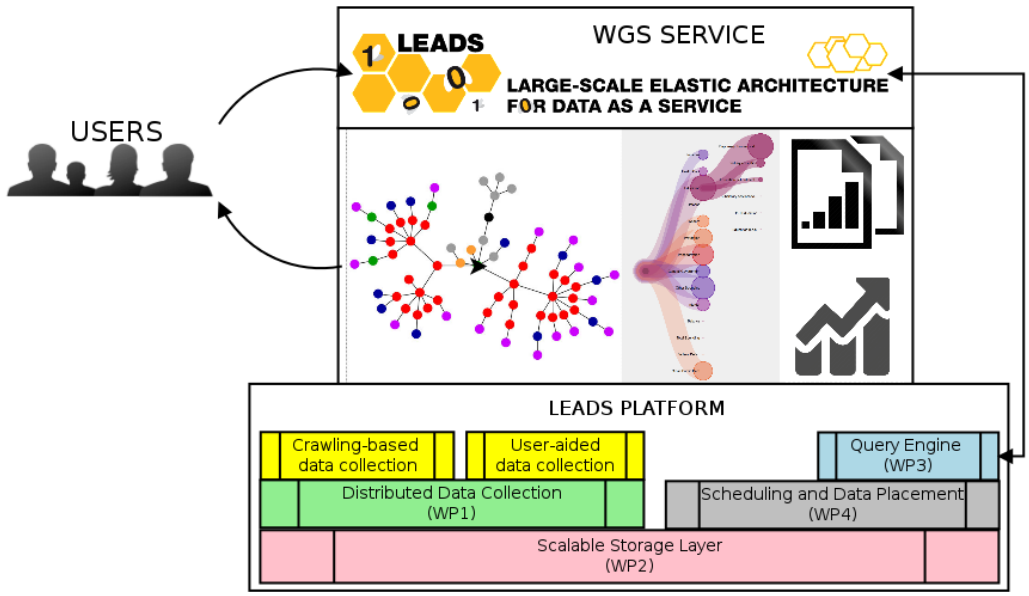


Figure 1: WGS prototype

2. WGS high-level design

As shown in Figure 1, the WGS prototype sits on top of the LEADS platform running on a single micro-cloud. It 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. Currently, the prototype exposes a number of predefined query patterns but additional query patterns are planned. Two sample usage scenarios and their business cases, which are provided by adidas will be depicted in detail in this section. Other potential query patterns together with their business and use cases are defined in a separate working document (WD5.2). Note that this document, for internal use in the project, has been updated for M18 to better concord with the possibilities of the WGS as described in the present deliverable.

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 AoTerra micro cloud at Dresden and are exposed to public use via open access URLs**. e.g., check **http://80.156.223.199:8000/restful-query-form/** for a POC of proposed 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. Applications and 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 urlwithin the given depth.

Figure 2. depicts the result of a basic data retrieval query that visualizes all pages reachable from *www.yahoo.com* up-to a depth of three.

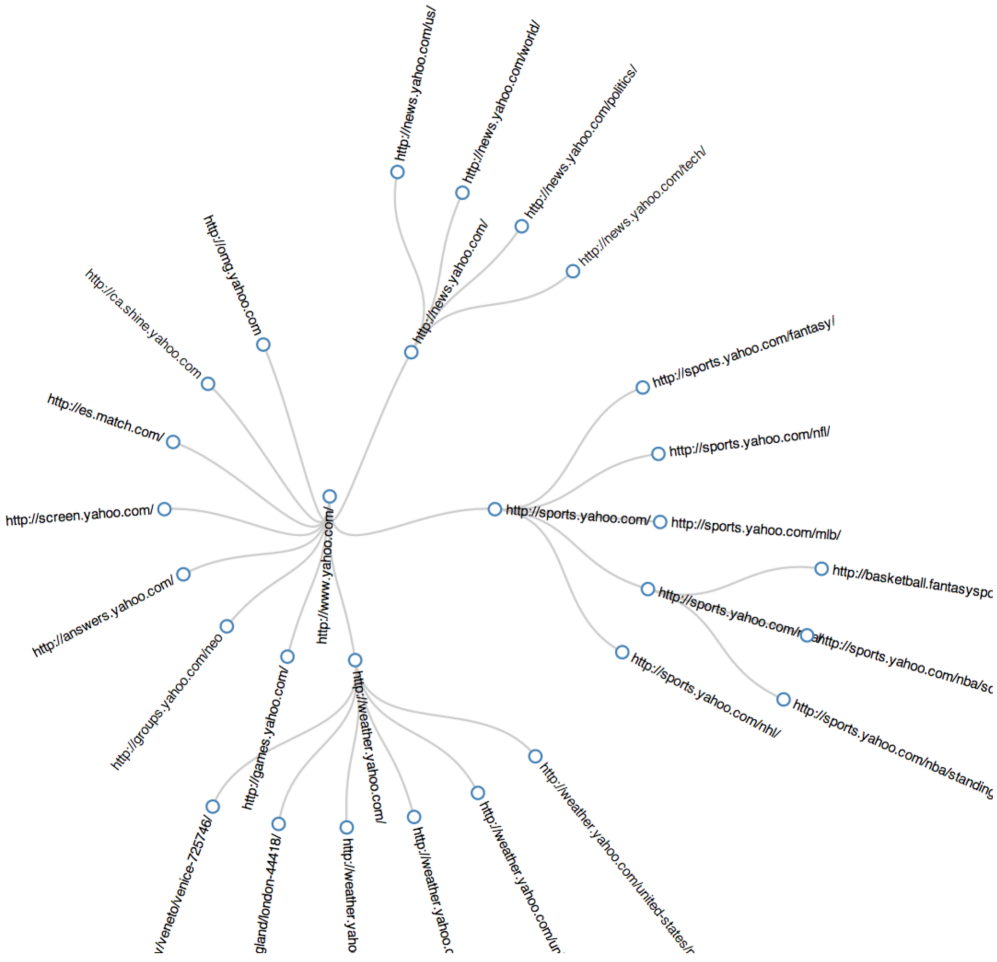


Figure 2: A sample basic data retrieval API output

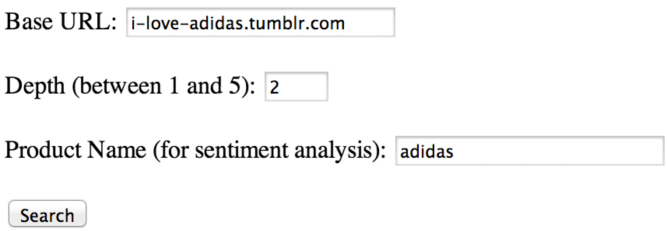
2.1.2 Public data processing queries

Public data processing queries provide access to pre-structured user queries. These queries are defined in coordination with typical LEADS customers and provided to users. 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.

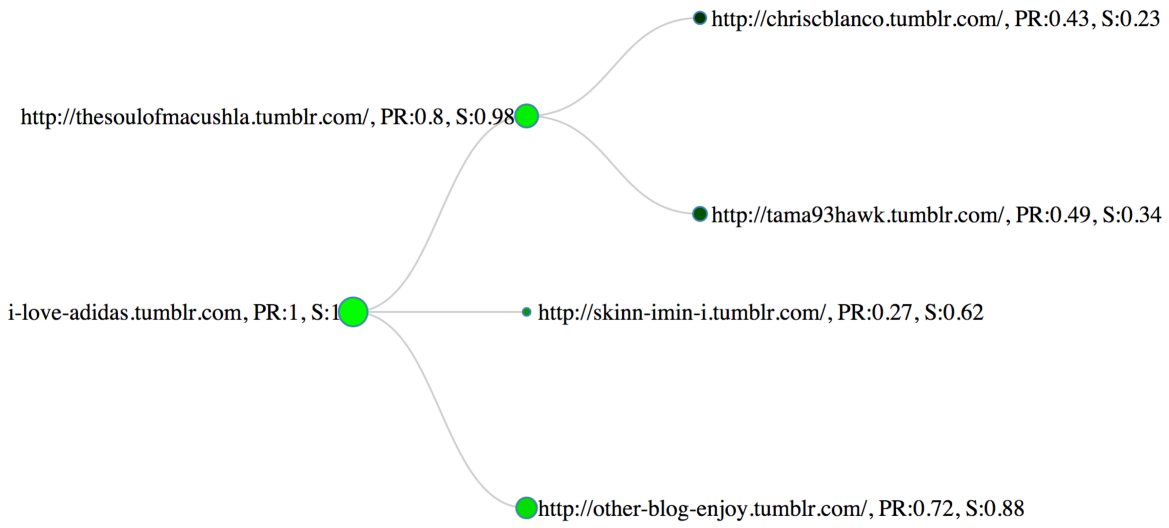
In Figure 3(a), the given query tuple is: <i-love-adidas.tumblr.com, 2, adidas> and the sentiment in sites such as thesoulofmacushla.tumblr.com, other-blog-enjoy.tumblr.com, that are linked from i-love-adidas.tumblr.com are visually displayed in Figure 3(c). WGS also exposes the collected data as a JSON object over a RESTful API as displayed in Figure 3(b). Using this API, it is possible for customers to use alternative visualization libraries of their own or streamline their graph-based analysis tasks. Note that in the figure different colours are used for positive (green) and negative (red) sentiments and the colour intensity is used to reflect the sentiment strength.



(a) A sample WGS query


```
{
    "pageRank": 1.0,
    "name": "i-love-adidas.tumblr.com",
    "sentiment": 1.0,
    "children": [
        {
            "pageRank": 0.8,
            "name": "http://thesoulofmacushla.tumblr.com/",
            "sentiment": 0.98,
            "children": [
                {
                    "pageRank": 0.43,
                    "name": "http://chriscblanco.tumblr.com/",
                    "sentiment": 0.23
                },
                {
                    "pageRank": 0.49,
                    "name": "http://tama93hawk.tumblr.com/",
                    "sentiment": 0.34
                }
            ]
        },
        {
            "pageRank": 0.27,
            "name": "http://skinn-imin-i.tumblr.com/",
            "sentiment": 0.62
        },
        {
            "pageRank": 0.72,
            "name": "http://other-blog-enjoy.tumblr.com/",
            "sentiment": 0.88
        }
    ]
}
```


(b) JSON result (accessible via a RESTful interface) for the given query



(c) WGS output for the given query

Figure 3: A sample public data retrieval API output

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 example, the selection critera is that the page must contain the word “adidas”. 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

Even though currently WGS implemented functionalities are designed to test the integration of proposed LEADS features, these functionalities are still evaluated from a business perspective in order to give some first simple examples of how links between pages, PageRank, keyword search or sentiment analysis can be used in order to provide an added value for the potential clients.

[1] http://d3js.org/

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.

2.2.2 Business case for public data processing queries

Public data available and served by the LEADS platform will constantly grow and enable various analytics. For the time being, the WGS provides visualization of a graph of linked pages extended by the information on the PageRank and the sentiment of chosen mentions in their text.

Such a visualization can be very interesting considering pages containing subjective statements, like the ones in blog pages. It would enable the efficient finding of influential bloggers. Moreover, it can also help in detecting anomalies like groups of linked blogs writing negative remarks about company products.

2.2.3 Requirements of WGS for the Distributed Application

In current WGS implementation, WGS provides tree-like Web graph outputs, which are good for visualizing the state of the web graph at a given point of time. However, there are visualizations other than tree-like graphs and depending on the use case, alternative visualisations may be required. By M24, WGS plans to serve a number of diagrams according to the needs of the distributed application proposed by partner adidas as part of T5.1. The types of useful diagrams and their general descriptions are presented below.

An important group of diagrams are line charts. They can present how values change over time such as the page rank of a page or the average sentiment of a product or the number of mentions of some keywords. Offering such visualization enables the execution of trend analysis of various aspects. In WGS, line charts can be supported with column diagrams.

Other important types of diagrams are pie charts and heat maps. To give a useful example, they can be used for more fine-grained visualizations of the sentiment. The sentiment of some product at a given time can be split into various dimensions (language, domain, topic of the site like fashion or sports). Colours of the parts of the heat map can express the sentiment, while the sizes can express the number of mentions.

Also graphs can be used to represent more general cases than a set of pages linked with each other. Single pages can be exchanged by domains (to visualize numbers of links between them) or by other objects (like keywords or languages).

Having all of these types of diagrams in the front-end coupled with a huge set of semi-structured information in the back-end, will introduce powerful opportunities for Business Intelligence.

2.3 WGS 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. 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.

```
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. 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 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.

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. 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"
}
```

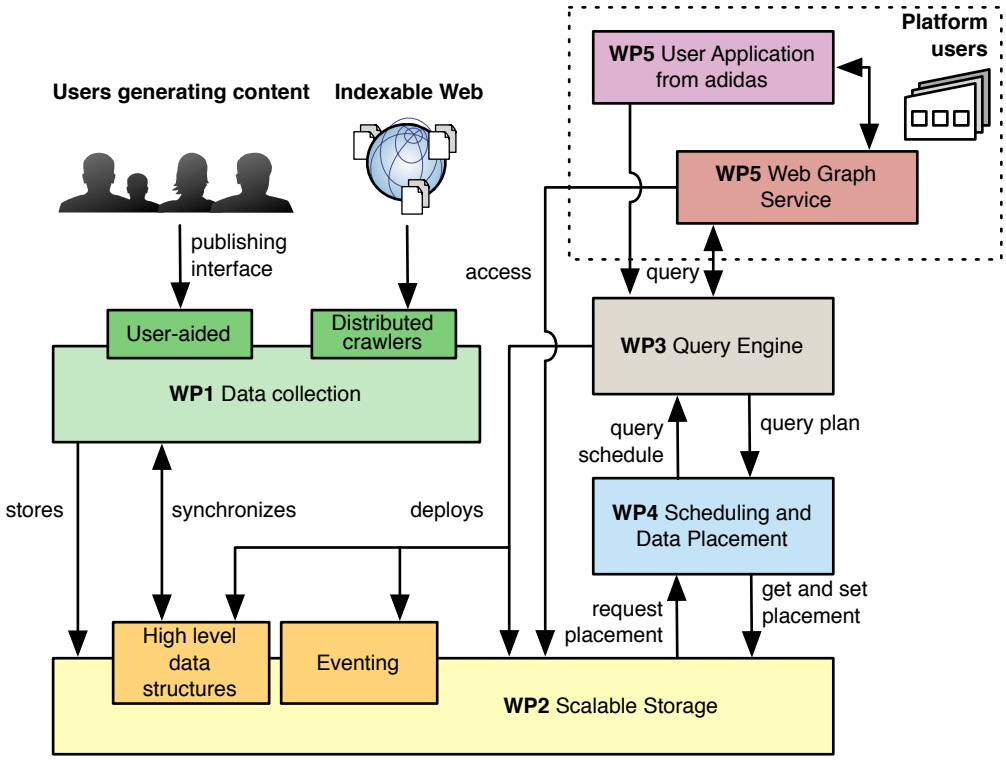


Figure 4: LEADS platform architecture

3. LEADS platform high-level design

Figure 4 displays the high level design of the LEADS platform. Single-site WGS prototype tests the integration of all components of the LEADS platform in a single micro-cloud. 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 4 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 can be 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 a single micro-cloud.

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.