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:	D5.8
Title of Deliverable:	Final evaluation results of LEADS across multiple micro-clouds
Contractual Date of Delivery:	2015.09.30
Actual Date of Delivery:	2015.09.30

Abstract

The document presents the final evaluation of the multi-cloud environment with a usage of datasets, components, and queries prepared by adidas, forming the representative application of the LEADS platform. We present the LEADS platform setup on Cloud&Heat micro-clouds and tools for the LEADS platform deployment automation. The tools --- based on SaltStack --- simplify and speed up horizontal scaling of the platform and let us to migrate the platform instances between micro-clouds if necessary.

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
Daniel Poelzleithner	Cloud & Heat	daniel.poelzleithner@cloudandheat.com
Steve Schmerler	Cloud & Heat	steve.schmerler@cloudandheat.com
Jens Struckmeier	Cloud & Heat	jens.struckmeier@cloudandheat.com
Gaurav Singh	BM-Y!	gauravonline20@gmail.com
Ioanna Tsalouchidou	BM-Y!	ioanna@yahoo-inc.com
Janette Lehmann	BM-Y!	janettel@yahoo-inc.com
Necati Bora Edizel	BM-Y!	edizel@yahoo-inc.com
David Garcia Soriano	BM-Y!	davidgs@yahoo-inc.com
Aslay Cidgem	BM-Y!	aslayci@yahoo-inc.com
B. Barla Cambazoglu	BM-Y!	barla@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 Struckmaier	jens.struckmeier@cloudandheat.com	2015-09-22
Approved by GA Member 1	Pawel Skorupinski Jacques Ohannes-sian	pawel.skorupinski @adidas-group.com jacques.ohannessian@adidas-group.com	2015-09-23
Approved by GA Member 2	Andre Martin	andre.martin@tu-dresden.de	2015-09-23

Contents

List of Figures

Executive Summary

The document presents the final evaluation results, including new test deployments and extensions to the adidas representative application. The LEADS platform is running in the multi-micro-cloud environment. It provides all necessary data and querying capabilities for the user application. The recent improvements in the query engine allowed us to reduce 10x the time required to retrieve the data. Additionally, we extended the evaluation scenarios with the AMPLab benchmark. AMPLab is a widely recognized benchmark for big data storages and databases.

Improving the meaningfulness of the dataset. Section 2 outlines the extensions that have been made to components forming the distributed application to enhance accuracy and increase business value. We have extended technological capabilities, information extraction precision, as well as proposed a new set of visualizations.

Multi-cloud setup. Section 3 gives details of the setup of the LEADS cluster in Cloud&Heat that we used to perform experiments. We used six micro-clouds for testing. The main evaluation ran on three of them.

Evaluation of the LEADS platform running the application. Section 4, 5, 6, and 7 present the evaluation of the LEADS platform from different perspectives. Section 4 discusses the performance data results for the adidas application running on the LEADS platform. In Section 5, we employee the AMPLab benchmark to evaluates the LEADS query engine as a Big Data application. In Section 6, we look into system performance metrics to analyse how the query engine and the underlying virtualized hardware handle AMPLab queries. Section 7 analyses the LEADS platform as a cloud application.

Automation of LEADS cluster deployment. Section 8 gives details on how we use state-of-the-art configuration management system to setup the LEADS cluster in Cloud&Heat micro-clouds.

1. Introduction

In Work Package 5, our task was to build a representative application on top of the capabilities and features that the LEADS platform provides. The application design follows the requirements from the use-case partner adidas.

The application enables adidas sales experts to monitor the mentions of products and assets of interest related to their products and competitors. It allows the extraction of business intelligence from the massive amounts of public Web data without having to set up a crawling, storage and processing solution in house. Yet, the application of mention monitoring requires specific computations to be performed on the data that is crawled, as soon as it is crawled (a near-real-time aspect). A distinctive feature of LEADS compared to other Big Data processing solutions is that it allows its clients to provide the system with such specific processing units in the form of plugins.

Three types of components (data storage, plugins, and user interface) form the structure of a typical LEADS application, and the representative adidas application follows this pattern as well. First, we offer the installation of custom plugins. A client can activate them on the platform to extract data of her interest from the pages contents. The extracted data may include information on keyword level (e.g. sentiment), page level (e.g. page language), and site level (e.g. site category). Second, we offer a query interface. A client can query **the e**xtracted data with her custom set of SQL-like queries. Finally, the client can define specific visualizations that form the presentation layer for the results of the queries obtained from the system (and in addition to the visualizations available from the LEADS Web Graph Service, described in D5.6).

2. Final workflow for business purposes

Throughout the last year of the project, we have been working on testing and fixing previous implementations of functionalities proposed by adidas. The goal was to have components working smoothly in the distributed environment of the LEADS platform. On top of that, we experimented with visualizations that create opportunities for valuable insights out of LEADS dataset.

In the previous M24 deliverable M5.4: Prototype of the distributed application for a single micro-cloud, we described our gradual approach for information extraction and our division of work in between batch and real-time jobs. We explained that our goal is to form multilevel information about the context of each mention out of the extracted data (each mention has local context features, page context features, its page has its own features as well as features in site context, and the site itself has features as well).

Throughout the last year, we have managed to make sure our approach works fine on a large-scale crawl in multi-cloud environment. In adidas, we have drawn our final business conclusions on how to apply the components that appeared to give valuable results in such environment.

2.1 Extracted data

As explained in the previous deliverables, we have been extracting data from E-commerce and news/blog pages. For E-commerce sites, the algorithm has been storing references to pages, which offer products of interest of our business partner adidas (our algorithm was automatically recognizing name and price fields on product offering pages of each E-commerce website). For news/blog sites, the algorithm has been storing references to pages that mention predefined keywords in the main content (as given by state-of-the-art Boilerpipe article extraction library).

Partner adidas has defined keywords of their interest that have been searched in product names and news/blog articles. Our special keyword matching component called *Document Concept Search* enables to define search keywords with a couple of extended parameters, such as: number of words that don't need to match, number of characters that don't need to match in each word, maximum distance in between keywords, and if terms need to be ordered. We have defined the following standard-keywords set for comparative analytics:.

nike mercurial	nike jordan	adidas supercolor	adidas ultra boost
nike air max	nike free 5.0	adidas superstar	adidas ace
nike air zoom	nike lunarglide	adidas supernova boost	messi
nike roshe	ronaldo	adidas terrex	kanye west
		yeezy boost	pharrell williams

2.2 Visualizations

As described in deliverable D5.7: Full implementation of the distributed application for multiple micro-clouds, we experimented with state-of-the-art d3.js Java Script library for data visualizations. For the LEADS review purposes, we have chosen two of the considered visualizations. One presents a view on E-commerce data, another one on news/blogs data.

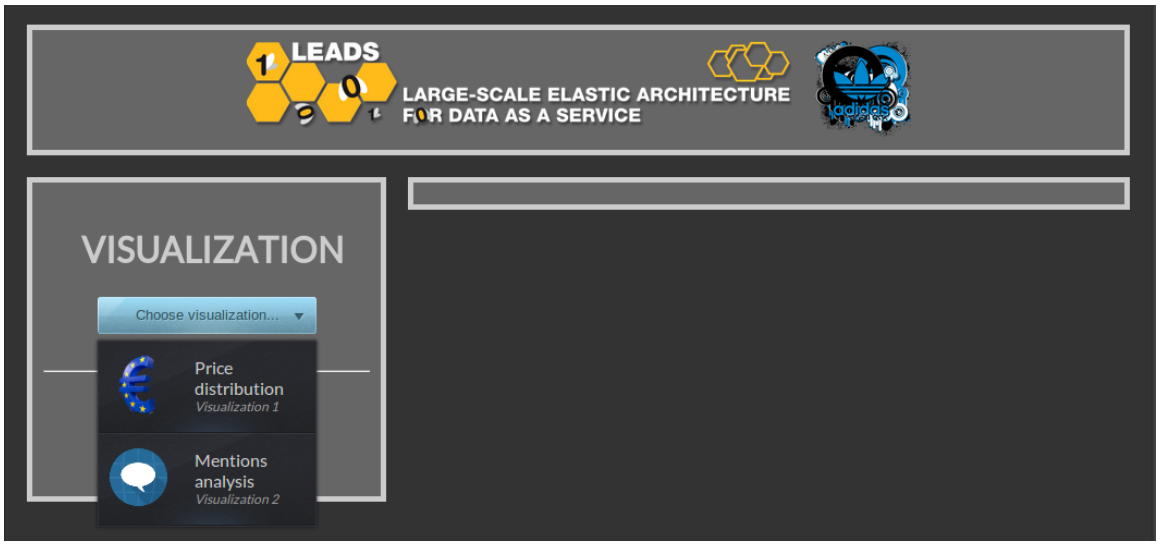

Figure 1: Expanded menu of LEADS visualizations UI

Figure 1 presents User Interface with the expanded list of the visualization that user can choose from. One is called Price Distribution, the second Mentions Analysis.

Figure 2 presents the options a user can choose from when defining data input for Visualization 1. This visualization enables the users to analyse how prices of specific products are distributed in between various E-commerce sites each week. She can choose keywords out of the list presented previously in the document - the products that will be returned need to contain these keywords in the name. The user can also choose specific E-commerce web sites and time period she is interested in. Later on, dynamically, she can choose to visualize a specific week.

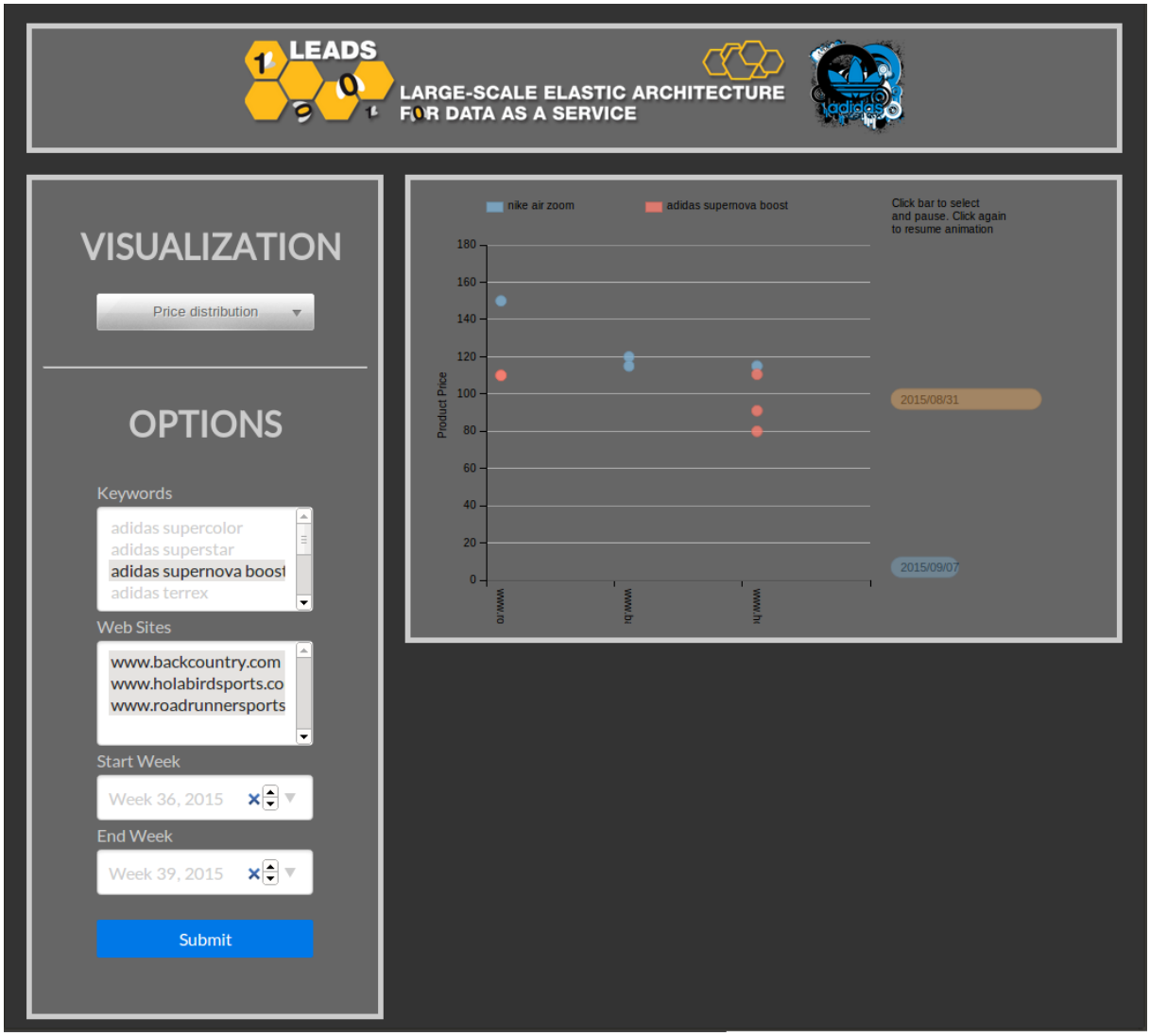

Figure 2: Menu and visualization of weekly price distribution of products in Ecommerce sites

Figure 3 presents the interface for analysing mentions in various news/blog site domains. User can name categories and choose which keywords each of them shall contain. She can choose web sites and time period as well. The output is presented in layers (the user can dynamically reorder these layers). The first layer presents always categories (in this example “adidas” and “Nike”). The following three ones (interchangeable) are: sentiment, web site, and keywords. On Figure 3, we see specifically that 9.80% of all of the analysed mentions are positive mentions of “adidas ultra boost” on Nike.com website.

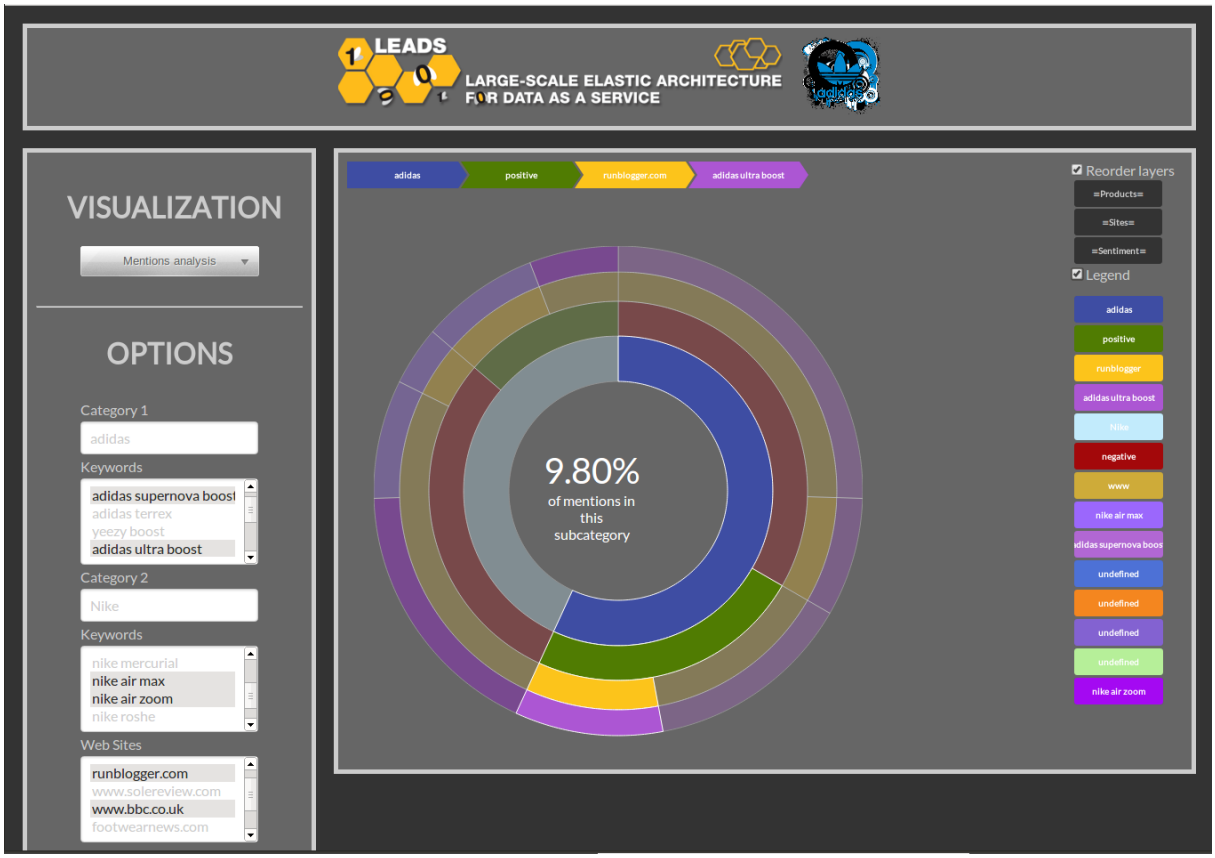

Figure 3: Menu and visualization for analysis of mentions of chosen keywords on various websites

3. Multi-micro-cloud setup

Figure 4 presents the LEADS platform cluster overview with a marked subset of nodes that we use for the detailed analysis of the query engine performance – 16 query-engine nodes. In the last period, we have extended the tests to six micro-clouds located in Germany. We run the final evaluation on mico-clouds: DD1A, DD2A, DD2C, and DRESDEN2. HAMM5 and HAMM6 provide additional resources to support the main application, i.e., object storage[1] for local AMPLab dataset, collected metrics, and keeping binaries of the LEADS application.

In hamm5 and dresden2 deployments, we have an on-demand Hadoop YARN cluster (in current testing phase, we run it permanently). It consists of three nodes. We always install the LEADS WP1 crawler Unicrawl on the primary YARN node. Unicrawl uses YARN to distribute crawling tasks per deployment.

Each micro-cloud runs at least one query engine instance (with Infinispan and Ensemble). DRESDEN2 runs 7 query engine instances; DD1A, DD2A and DD2C host 10 query engine instances each. Overall, we have 30 primary query-engine nodes to support AMPLab tests. In addition, 7 query-engine instances are available in DRESDEN2.

[1] Openstack Swift, http://docs.openstack.org/developer/swift

The query engine nodes can be dynamically added with our central configuration based on Saltstack [2](configuration management framework). It lets us extend our infrastructure with nodes from other micro-clouds and perform experiments with different configuration. For example, during the past months, we have moved the main nodes from hamm5 and hamm6 (and scale out) to DD1A, DD2A, and DD1C.

We run the query engine nodes in DD2A and DD2C to perform the experiments presented in Section 5. Section 4 discusses the evaluation of the adidas queries (queries necessary to get data for adidas application) runs on dresden2. The detailed tests of the query engine in Section 0 are executed on DD1A, DD2A, and DD2C.

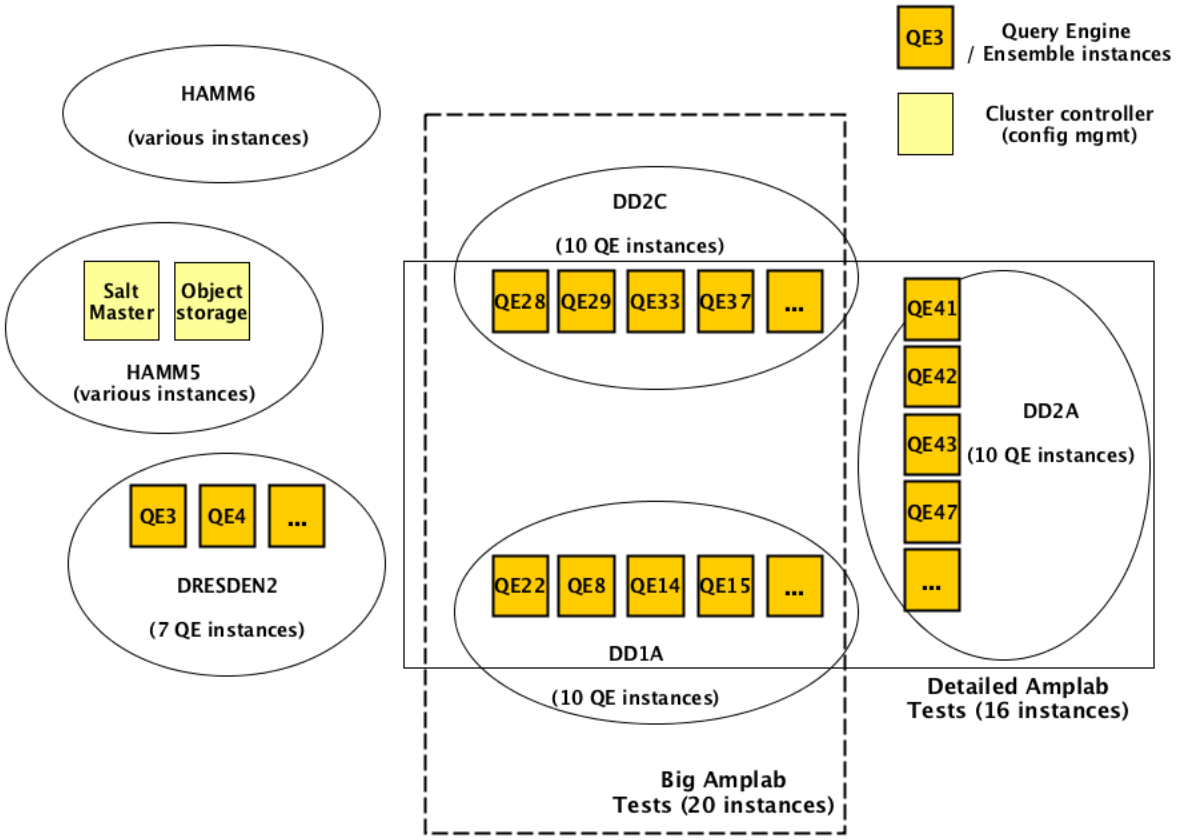

Figure 4: Multi-cloud LEADS setup

An essential requirement for the LEADS cluster setup was to deliver a robust way for collecting metrics. For this purpose, we deployed PCP.io on all nodes[3]. PCP.io is a service for collecting all system performance metrics, such as CPU usage. The most important performance metrics for our experiments are: CPU utilization, memory consumption, and network usage. PCP.io stores metric values in a very concise format (so we can collect them for long time) and --- in case of any issues --- we have almost thousand metrics collected additionally per default. The additional metrics help us to investi-

[2] See: http://saltstack.com/
[3] See: https://github.com/skarab7/leads_query-engine/blob/develop/salt/salt_master/srv_salt/top.sls#L22

gate deeper any issue or interesting observation. Figure 5 depicts one of PCP frontends – vector[4]. We store all PCP measurements in object store to share them among project partners.

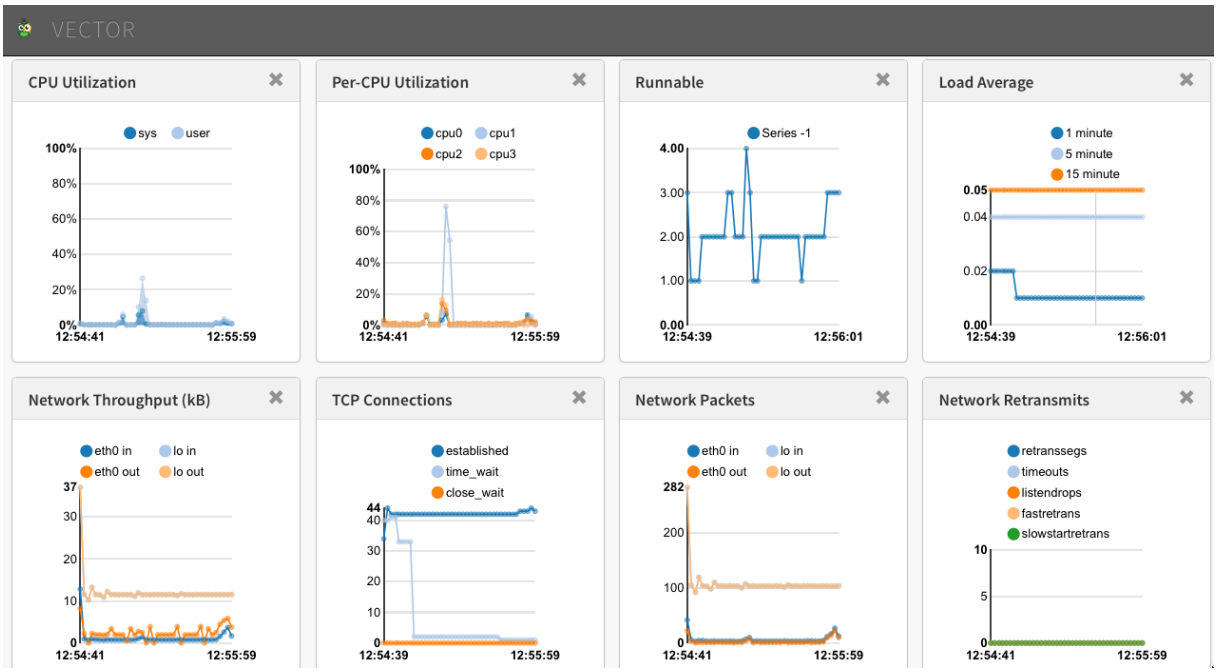

Figure 5: PCP frontend Netflix vector

To analyse the optimization techniques that minimize the cross-deployment traffic, we capture network traffic with Tcpflow[5] (the corresponding script is available on github[6]). **Figure 6** shows the standard Tcpflow report on traffic recorded.

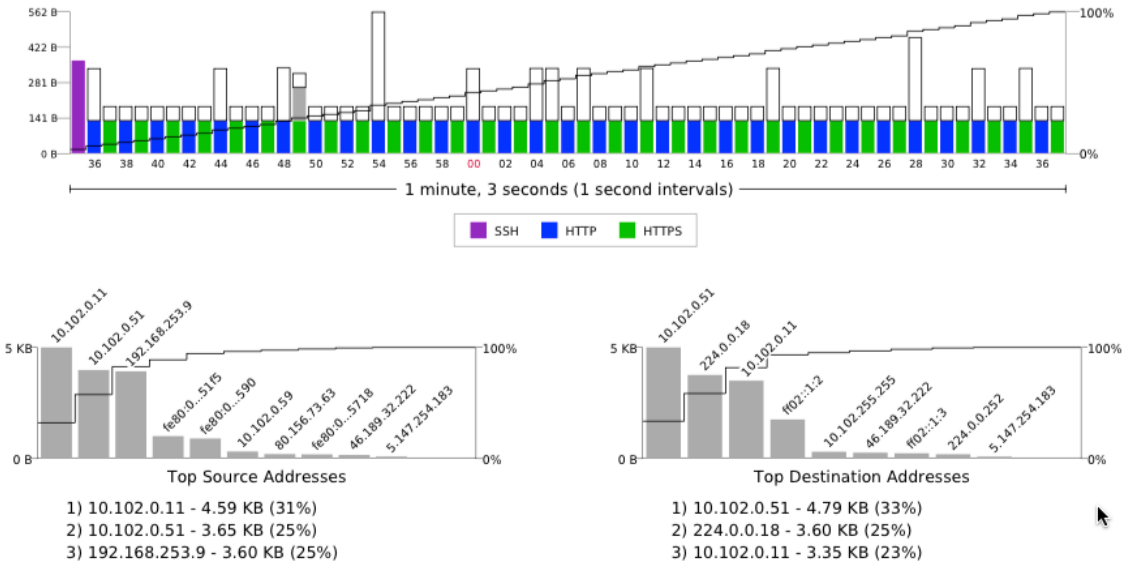

Figure 6: Screenshot of tcpflow report from Query Engine 3 in dresden2.

[4] See: https://github.com/Netflix/vector
[5] https://github.com/simsong/tcpflow
[6] https://github.com/skarab7/leads_query-engine/blob/develop/fabric_metrics.py#L66

4. Evaluation of the LEADS platform running the application

In this section, we present the performance results of the LEADS platform running the adidas application. We structured the results into two sections:

a) The information extraction plugins, which are executed over the streams of crawled Web pages;
b) The query engine, used through its interface for SQL queries.

Both plugins and SQL queries are crucial for adidas and for other potential users of the LEADS platform. The plugins support performing arbitrary, user-specific processing, and information extraction from the crawled web pages over the multi-user LEADS platform. This way we can share the cost of maintaining the crawl across all LEADS users.

adidas uses plugins to extract useful information from the crawled Web pages, such as sentiment of each web page, products, prices and keywords mentioned in the E-commerce web pages, etc. An adidas user can use formulate the analysis with SQL language. The SQL language is a standard that ensures that the user can easily use a tool of his choice. The LEADS API (Application Programming Interface) lets user to submit the queries in remote way.

The adidas application uses SQL queries both through API (in the representative application) and through the command-line interface, to quickly collect the required data.

4.1 Information extraction on streams of crawled web pages

To test the correctness and performance of the information extraction functionality in the representative application, we have separately considered two deployed information extraction plugins. The first plugin is a system plugin that enables batch messaging between micro-clouds (the batch messaging service is described in D3.4 in more detail). The plugin receives messages in batch from remote nodes/micro-clouds, and processes them. It is used to reduce the number of messages that need to be sent across micro-clouds, and thus also the network congestion.
The second plugin implements the following workflow:

1. Extraction of web page's domain;
2. Extraction of web page's text content;
3. Extraction of content's language;
4. Determination of an algorithm to extract valuable part of the content;
5. Extraction of valuable content;
6. Search for keywords inside of the valuable content and determination of relevance and sentiment of mentions.

The results produced by both plugins are stored using (distributed) ensemble caches in a relational format. Notice that both plugins perform computationally expensive NLP tasks, as well as SQL queries that need to process high volumes of data. As such, the ability of the LEADS platform to scale across micro-clouds is pivotal for the scalability of the plugins.

Both plugins were tested by initiating a live web crawl starting from two blog/news sites that have business interest for adidas as the crawling seeds. These sites are:

- *runblogger.com* - featuring running reviews and news,
- *footwearnews.com* featuring news on shoes.

The resulting data set after one full crawl over a period contained a total of 17848 web pages of interest among all crawled pages. The extracted information was saved in 32 thousand records, in a relational table.

The two plugins were tested on both a single micro-cloud and a multiple micro-clouds configuration. In the single-cloud configuration the plugins were deployed on two virtual machines (VMs) in micro-cloud hamm6. The multi-cloud configuration included two micro-clouds (hamm6 and dresden2), each hosting two VMs. All VMs were configured with four virtual processors and eight GBs memory. In the following, we will be focusing on the multi-cloud deployment, which is more meaningful in terms of scalability.

Table 1 shows the processing rates for the two plugins, under the two different system configurations. The *Processing rate* column corresponds to a single-node deployment of each plugin. We see that Plugin 1 is very efficient, whereas Plugin 2 is fairly time-consuming. This is due to the expensive text extraction tasks, string manipulations, and large number of SQL queries that are executed within Plugin 2. Therefore, it is important that better rates are achievable by increasing the computational capacity available to the plugins and the query engine, i.e., the number of nodes in the micro-cloud, or the number of micro-clouds. In fact, processing rate increases almost nearly with the number of nodes, e.g., when deployed over four nodes, Plugin 2 was able to process 11.43 pages per minute.

Table 1. Indicative plugins performance – rate is presented per node

Plugin	Processing time per record (average)	Processing rate (average)	Total extracted information
1	0.26 sec	3.8 records/second	1000 records per execution
2	19 sec	3.15 pages/min	32k records total

4.2 Performance and validity of the LEADS query engine running the application SQL queries

In the second experiment, we evaluate the query engine performance. For this, we used a larger data set, which was crawled for the M24 prototype. The data set contained 28 thousand web pages. Our experiment involved executing the following representative SQL queries - Q1 and Q2 - presented in Table 1, which are essential for the functionality described in deliverable D5.4.

Table 2 Evaluation scenario for detailed evaluation

Query id	Query
Q1	`SELECT K.uri, K.ts, K.keywords, K.sentiment AS sentiment FROM keywords K JOIN page_core C ON C.uri = K.uri WHERE C.ts = K.ts AND K.partid like 'article_content:000' AND C.lang like 'en' AND K.ts>=0 AND K.ts<=1416504380252 AND keywords IN ('adidas')`
Q2	`SELECT uri, ts FROM keywords WHERE ts>=0 AND ts<=1416501600106 AND partid = 'ecom_prod_name:000'`

Intuitively, query Q1 requests all articles that are written in English, that contain the keyword ‘adidas’ and were crawled before time stamp ‘1416504380252’ (expressed in Unix timestamp, i.e., as milli-

seconds since January 1st, 1970). The query consists of a join between two tables, i.e., table keywords that contains all keywords detected in each page, and table page_score, which stores page-related information, e.g., the language for each Web page.

Query Q2 requests all pages of E-commerce sites that contain a product offer, as adidas plugin extracts a product name and price from this type of pages. We are interested only in versions of the pages site that were crawled with timestamp before '1416501600106'.

Table 3. Query answering

Query	#Records	Execution time (Execution time in D5.7)	Transfer volume
Q1	21	6 sec. (50 sec)	141 MB
Q2	751	1.1 sec. (11 sec.)	7.4 MB

Table 3 summarises the number of returned records, execution time and transfer volume for the two queries when these are executed over the multi-cloud configuration described in the previous section.
In comparison to the numbers reported in D5.7, execution times improved by an order of magnitude. We achieve the time reduction with an improved (cost-aware) query planner, an optimized batch messaging service, integration of indexes and, finally, utilization of local auxiliary storage. All these techniques are described in more detail in D3.4.

5. AMPLab-based query processing engine evaluation

We evaluated the performance of the query execution engine on different configurations using the AMPLab benchmark.[7] It contains both a configurable data generation tool and a query generator, and is frequently used for Big Data systems evaluation.

5.1 Query generation and system configuration

To evaluate our engine we used two types of queries: (a) the AMPLab queries that are generated by the benchmark scripts, and (b) a set of additional – synthetic - queries on the same data set.

The AMPLab scripts create four different query types: Queries 1-3 are standard SQL queries, whereas Query 4 involves user-defined functions implemented as external scripts, which is not supported by our query engine. We therefore focus on the first three queries. All AMPLab queries have a very high selectivity, i.e., they return several hundreds of thousands of results, sometimes even several millions of results, or they limit the size of the results at the last step. These types of queries are useful for offline data mining and data analytics, e.g., run a clustering algorithm on a subset of the crawled Web pages.

In addition, we have devised a set of additional queries that are more selective than the previous (i.e., return a few hundreds of results). These selective queries are the ones that a human would be interested to execute, e.g., via an API, the command-line interface, or via our Apatar-based GUI. No-

[7] Available at https://AMPLab.cs.berkeley.edu/benchmark/

tice that these queries are the ones that will typically utilize indexing functionalities and clever distributed processing algorithms, to avoid loading everything from disk or shipping too much data over the network. Table 4 presents both sets of queries.

Table 4 Evaluation Queries

Query id	Query
	AMPLab queries
Q1	`SELECT pageURL, pageRank FROM rankings WHERE pageRank > X` for X=1000 (**Q1a**), 100 (**Q1b**)
Q2	`SELECT SUBSTR(sourceIP, 1, X), SUM(adRevenue) FROM uservisits GROUP BY SUBSTR(sourceIP, 1, X)` for X=8 (**Q2a**), 10 (**Q2b**)
Q3	`SELECT sourceIP, totalRevenue, avgPageRank FROM (SELECT sourceIP, AVG(pageRank) as avgPageRank, SUM(adRevenue) as totalRevenue FROM Rankings AS R, UserVisits AS UV WHERE R.pageURL = UV.destURL AND UV.visitDate BETWEEN Date(`1980-01-01') AND Date(`X') GROUP BY UV.sourceIP) ORDER BY totalRevenue DESC LIMIT 1` for X='1980-04-01' (**Q3a**), '1983-01-01' (**Q3b**)
	Additional queries
Q4	`SELECT pageURL, pageRank FROM rankings WHERE pageRank = X` (executed with random X values which are contained in the data set)
Q5	`SELECT sourceIP FROM uservisits WHERE visitDate = Date(X)` (executed with random dates X which are contained in the data set)
Q6	`SELECT pageURL, pageRank FROM rankings WHERE pageRank >= X AND pageRank <= Y` (executed with random X and Y values which are contained in the data set, with Y-X=5)
Q7	`SELECT sourceIP FROM uservisits WHERE visitDate BETWEEN Date(X) AND Date(Y)` (executed with random dates X and Y which are contained in the data set, and are set such that the date predicate covers two days)
Q8	`SELECT sourceIP, totalRevenue, avgPageRank FROM (SELECT sourceIP, AVG(pageRank) as avgPageRank, SUM(adRevenue) as totalRevenue FROM Rankings AS R, UserVisits AS UV WHERE R.pageURL = UV.destURL AND UV.visitDate BETWEEN Date(X) AND Date(Y) GROUP BY UV.sourceIP) ORDER BY totalRevenue DESC LIMIT 1` (executed with random dates X and Y which are contained in the data set, and are set such that the date predicate covers two days)
Q9	`SELECT SUBSTR(sourceIP, 1, 10), SUM(adRevenue) FROM uservisits WHERE visitDate BETWEEN Date(X) AND Date(Y) GROUP BY SUBSTR(sourceIP, 1, 10)` (executed with random dates X and Y which are contained in the data set, and are set such that the date predicate covers two days)

Considered configurations. The considered experimental setups are presented in Table 5. Our strategy on choosing the configurations was to evaluate how the query engine scales with the data volume, and with the size of the network (number of nodes per micro-cloud, for two micro-clouds). All reported experiments were executed on resources provided by our project partner Cloud&Heat (micro-clouds DD1A and DD2C). The used computing resources were shared, i.e., the real machines were hosting additional virtual machines (not related to LEADS). The virtual machines were configured with 4 cores and 8 GB RAM. We used the default configuration provided by the Bootstrapper, which deployed two system components at each virtual machine. Infinispan Ensemble was configured to keep 1024 entries in main memory per node, and with 128 MB in-memory cache. Also, LevelDB auxiliary storage was configured to use a RAM buffer index of maximum size 32 MB.

In the following, to avoid repetition, we will be reporting only the configuration values that deviate from the default values of Table 5.

Table 5 Considered configurations

Setting	Possible values (default values are emphasized)
#tuples	4 Million per table, **16 Million per table**, All data (18 Million in Rankings, 155 Million in userVisits)
#nodes per micro-cloud	**4**, 6, 9

5.2 Varying the data set size

Table 6 presents the time taken for answering the AMPLab queries in a deployment of 2 micro-clouds with 4 machines each. Our first observation is that execution time for the AMPLab queries increases almost linearly with the data set size. The small deviation on this linear relation is attributed to CPU spikes due to variances in the performance of the virtual machines, e.g., due to network load or workload caused by other virtual machines that were hosted at the same physical machine. This linear scaling is normal for these types of queries, since traditional centralized database optimizations cannot be utilized for these queries, due to the large volume of data that satisfies the predicates. For example, indexes cannot help; even though indexes can be used for quickly retrieving all tuple ids that satisfy the predicate, the actual tuples still need to be retrieved from the distributed storage infrastructure (the Ensemble cache), which takes a lot of time when the tuples in the intermediary result set are many. Therefore, the planner does not utilize indexes on these queries. In fact, since sequential distributed GETs are substantially slower than batch loading and iteration over all tuples (due to the distributed deployment of the Ensemble), the execution planner utilizes indexes only for fairly selective queries. Also note that the size of the result also affects execution time, since more results need to be shipped over the participating network. For example, Q1a requires less time than Q1b, since the former is more selective than the latter (the results of Q1a are a subset of the results of Q1b).

In Table 7, we present the required execution time for the 9 additional queries. For comparison purposes, we present time with and without indexes. Interestingly, execution time with indexes grows sub-linearly with the number of tuples. This happens because the planner incorporates the indexes for answering these queries, which enable efficient (logarithmic-cost) retrieval of the tuple ids that belong in the results without parsing the whole data set. Since the number of results that need to be retrieved for these queries is relatively small (in the order of hundreds or a few thousands), distributed GETs over the Ensemble cache do not pose performance bottleneck.

Table 6 AMPLab queries

Total tuples	Execution time (sec.)					
	Q1a	Q2a	Q3a	Q1b	Q2b	Q3b
4 M	8	177	28	11	199	61
16 M	22	687	95	30	738	257

Table 7 Additional queries

Total tuples	Execution time (sec.)					
	Q4	Q5	Q6	Q7	Q8	Q9
4 M	WO/Index:8	11	8	12	25	13
	W/Index:1	3	1	3	7	5
16 M	WO/Index:21	41	21	40	67	45
	W/Index:1	3	1	3	7	5

Table 8 presents the required time for executing the same queries over the full AMPLab data set (table uservisits with 155 Million tuples, and table rankings with 18 Million tuples). To achieve better performance, in this experiment we have used all the available free virtual machines in the two micro-clouds (9 VMs per micro-cloud). As expected, the queries require more time due to the increased data volume (overall an order of magnitude more tuples in table uservisits compared to the 16M data set). Nevertheless, utilization of indexes drastically cuts down on the IO costs, as evident, e.g., in queries 4 and 6, which are plain selections with filters. Queries Q5, Q7, Q8, Q9 also benefit from the index in terms of scanning the data, but the required time in these queries is dominated by the network time.

Table 8 Experiments with the full AMPLab data set (18 Million in rankings, 155 Million in uservisits, 2 micro-clouds, 9 nodes each)

Presence of indexes	Execution time (sec.)					
	Q1a	**Q2a**	**Q3a**	**Q1b**	**Q2b**	**Q3b**
No (index not utilized)	34	4130	511	30	4280	2006
	Q4	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**
No	34	365	35	366	409	384
Yes	2	37	2	61	51	47

5.3 Varying the number of nodes per micro-cloud

The set of experiments was designed to test the availability of the system to scale out, i.e., incorporate the capacity of more nodes to increase system performance. Therefore, all queries described at Section 5.1 were executed at two different settings: (a) with 4 nodes per micro-cloud, and (b) with 8 nodes per micro-cloud.

Table 9 and Table 10 present execution time for all queries. We see that querying performance improves by adding more nodes, which is a desired property for Big Data systems. The scale-up ratio is almost linear for the AMPLab queries, i.e., querying time is proportional to the number of machines.

This happens because, as stressed earlier, these queries disable traditional relational databases optimization strategies, such as indexes.

Interestingly, the number of nodes per micro-cloud does not have a substantial performance impact for queries Q4 - Q9, when index is used. For example, Q4 requires 2 seconds in both deployments. This happens because the generated query plans of Q4 – Q9 utilize indexes, which already improves performance by reducing the tuples that need to be loaded from disk (recall that indexes offer logarithmic-cost lookups). As such, adding more nodes has a negligible improvement on the already-fast loading time (logarithmic cost), but cannot help with the time spent on network operations, e.g., for shipping intermediary results across nodes and micro-clouds, which becomes the dominant cost. Nevertheless, the ability of the platform to run on larger infrastructure is pivotal for the scalability of the system with more data.

Summarizing, our experiments have shown that the query engine scales with the data set size. Scalability is achieved either by utilizing indexes, or by adding more computational capacity, e.g., more nodes, or more micro-clouds.

Table 9 AMPLab queries

Nodes per micro-cloud	Execution time (sec.)					
	Q1a	Q2a	Q3a	Q1b	Q2b	Q3b
4	22	687	95	30	738	257
6	15	465	63	21	510	185

Table 10 Additional queries

Nodes per micro-cloud	Execution time (sec.)					
	Q4	Q5	Q6	Q7	Q8	Q9
4	WO/Index:21	41	21	40	67	45
	W/Index:2	4	1	4	9	5
6	WO/Index:16	25	15	27	47	30
	W/Index:2	5	1	7	9	6

6. Detailed evaluation

Additionally to measuring the performance and the time for executing the queries, we investigate the performance metrics for the platform as a whole when running the application. We were interested in the node load generated by executing the queries, memory consumption, and sample packet exchange between tested nodes. The package exchange patterns also allowing us to assess whether the load is distributed properly. For this purpose, we will use a subset of AMPLab queries to run and analyse the performance of the LEADS query engine. The queries are shown in Table 10.

As depicted on the Figure 4, we run the experiments in three micro-clouds, four instances of query engine each. Every query-engine instance has two Infinispan servers installed. We loaded six million tuples in AMPLab Ranking and Uservisits tables. We split the data equally on all the instances.

Table 11 Evaluation scenario for detailed evaluation

Query id	Query
	Queries on AMPLab dataset
Q1	`SELECT pageURL, pageRank FROM rankings WHERE pageRank > 1000;`
Q2	`SELECT SUBSTR(sourceIP, 1, 8), SUM(adRevenue) FROM uservisits GROUP BY SUBSTR(sourceIP, 1, 8);`
Q3	`SELECT sourceIP, AVG(pageRank) as avgPageRank, SUM(adRevenue) as totalRevenue FROM Rankings AS R JOIN uservisits UV ON R.pageURL = UV.desturl WHERE UV.visitDate >= '1980-01-01' AND UV.visitDate <='1980-04-01' GROUP BY UV.sourceIP ORDER BY totalRevenue DESC LIMIT 1;`
Q4	`CREATE INDEX myindex1 ON Rankings (pageRank);`
Q5 (Q1 with index)	`SELECT pageURL, pageRank FROM rankings WHERE pageRank > 1000;`
Q6 (Q1 with index and warm cache)	`SELECT pageURL, pageRank FROM rankings WHERE pageRank > 1000;`

Figure 7 depicts the CPU load (reported by the Linux kernel) distribution at each of the four nodes in the two micro-clouds, while running the SQL queries from Table 10 (the timeline in Table 11). As in the previous evaluation presented in D5.7, we recognize that there is an almost even workload distribution across the nodes. Notice, that this is a required property for scalability and high parallelization to avoid bottlenecks, and it is achieved due to a combination of the load balancing scheme (uniformly distributing of the records across all nodes/micro-clouds) and the distributed implementation of the SQL operators (each node processes the data contained in the local portion of the Ensemble).

A similar pattern can be identified in Figure 8, Figure 9, and Figure 10 that presents the disk reads, writes, and network traffic (received data) during queries executions.

We can see that Q2 and Q3 require the biggest number of resources between the 8'th and 24'th minute (see Table 10) of the experiment. Q1 requires very little CPU resources and more network re-

sources. After creating an index in Q4, Q1 with index and warm cache minimized the required communication between nodes.

Table 12 Query timeline

Query	Time line (min)
	0
Q1	5 – 6
Q2	8 – 13
Q3	14 – 24
Q4	25 – 29
Q5	30 – 31
Q6	31 - 32

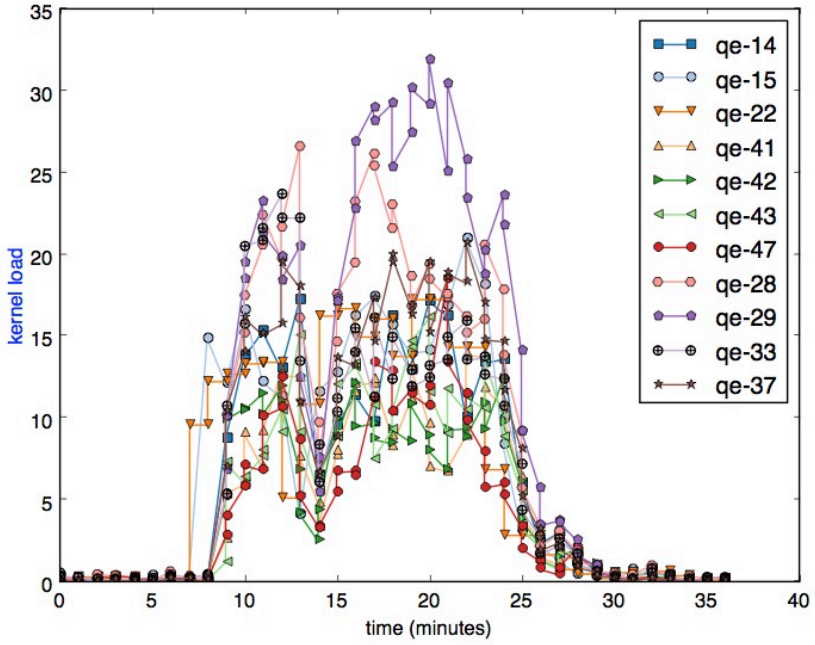

Figure 7: Node load running Query Engine Instances

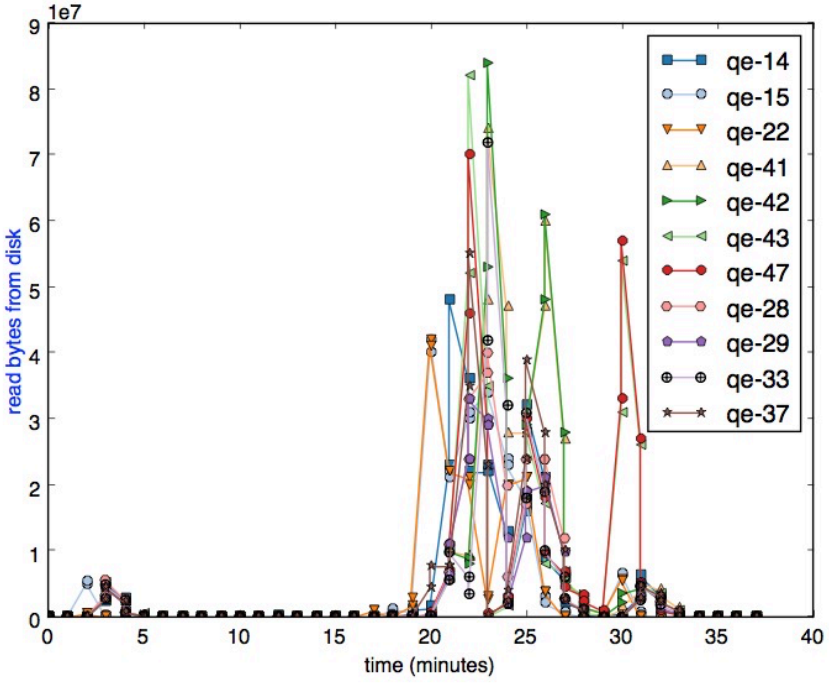

Figure 8. Read operations from disk

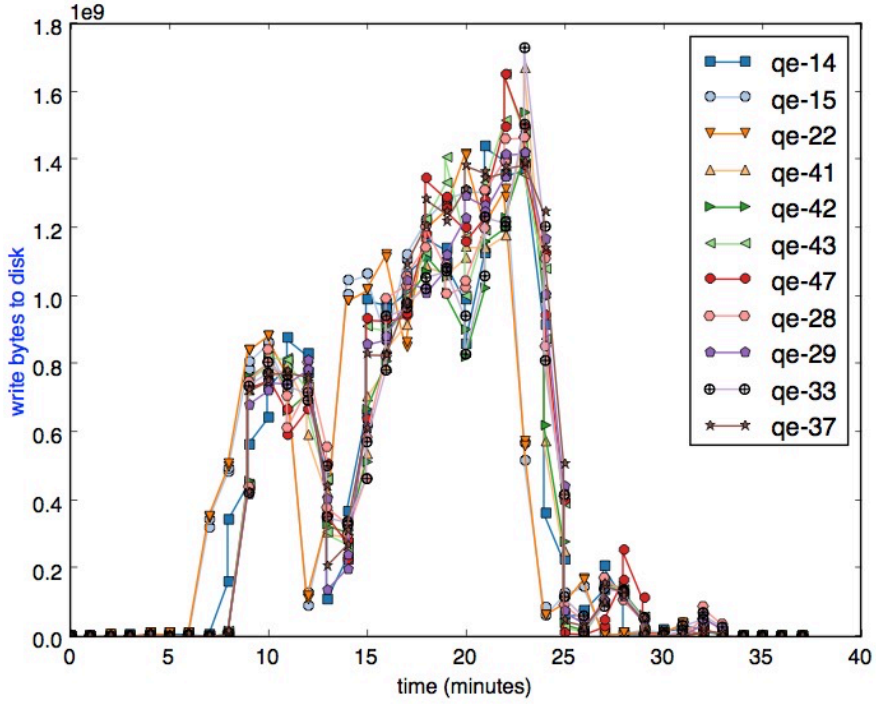

Figure 9: Write operations to disk

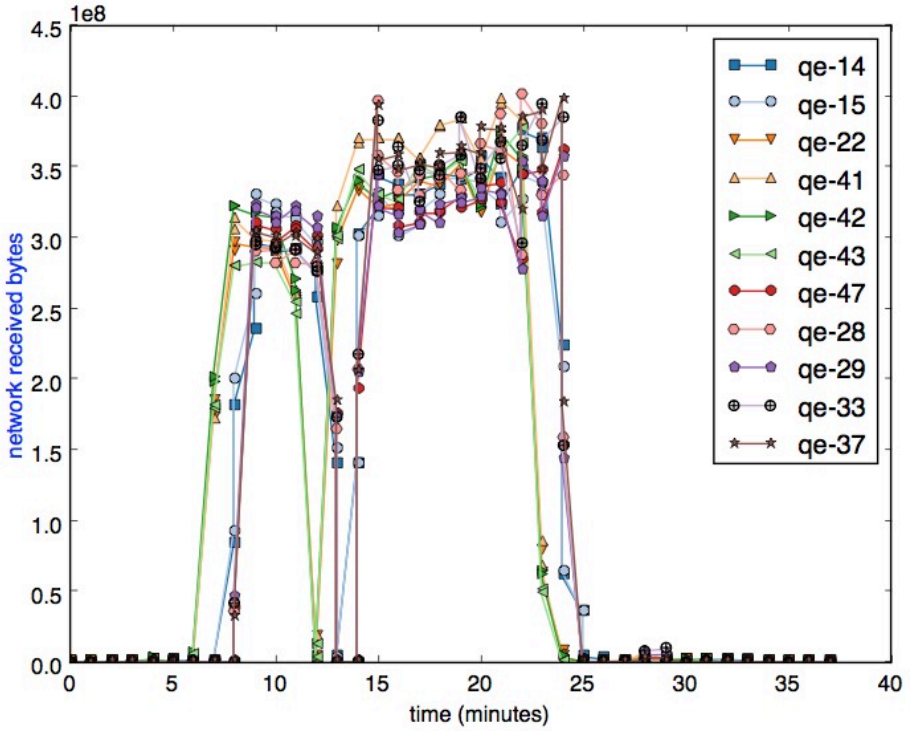

Figure 10: Received bytes from network

7. Cost evaluation

Table 13 shows some selected configurations of installing the LEADS platform, in particular the adidas application and the query-engine backend. For comparison, we have put an estimated cost of all VMs in DD1A, DD2A, DD2C, DRESDEN2, HAMM5, and HAMM6 that the LEADS project uses on Cloud&Heat infrastructure for testing. We took the current prices on Cloud&Heat IaaS.
The cost of VMs that support the adidas application is 408,80 Euro per month in case we would like to keep it running all the time. Additionally, we need to run 4 virtual machines with Hadoop for crawler. That would cost another 408.8. The 817.6 could be shared amount many companies. Moreover, the LEADS platform could be partially hosted on premise.

Table 13 Estimated cost for running LEADS platform

Conf	Description	VM # (size)	Cost per VM / month	Estimated Cost per month
1	Adidas application dataset	4 (L - QE) + 4 (L - crawler)	8 * 102.20	817.6
2	AMPLab cluster with subset of data	12 (L)	12 * 102.20	1226.4
3	AMPLab cluster with full dataset	20 (L)	20 * 102.20	2044.0
4	Large scale cluster	28 (L)	28 * 102.20	2861.6
5	All LEADS project VMs	3 (XS), 5 (S), 10 (M), 43 (L)	3 * 14.60 + 5 * 29.20 + 10 * 58.40 + 43 * 102.20	5168.4

The setup for AMPLab dataset is larger. We need at least 12 machines for its smaller set (see Section 6) and 20 (see Section 4). We are now talking about 1226.4 to 2861.6 Euros per month. Assuming that we have more than four companies, the cost, in compare to benefits, would be not so high per partner.

In Section 6, we have seen that the LEADS application using cache and indices may minimize the required traffic exchange between the nodes. The traffic between the nodes in different micro-cloud usually is charged. Therefore, it will help us to estimate the cost for the cross micro-cloud traffic (currently in Cloud&Heat: €0.02 pro Gb). In Table 14, you can find a cost estimation for traffic.

Table 14 Traffic cost estimation

	Data set	Size Gb	Cost Euro
1	AMPLab *1node dataset*	54.4	1.09 Euro
2	AMPLab cluster *5 node dataset*	263.7	5.274 Euro
3	Adidas crawl	5.5	0.11 Euro

8. Automation of LEADS cluster deployment

In comparison to deliverable D5.7, we extended the configuration management code based on SaltStack[8]. During the past months, our configuration management implementation helped us to manage instances (currently over 30 for tests) and migrate the LEADS application between micro-clouds.

We manage all the LEADS instances from one central SaltStack master (installed in hamm5 as shown in Figure 4).

Our configuration management code lets us to provision:

- Leads-query instances,
- YARN and Unicrawl instances,
- Infinispan/Ensemble clusters per micro-cloud,
- PCP.io packages on instances to collect system performance metrics.

Additionally, we have a set of Fabric[9] scripts for:

- Starting and stopping YARN, Unicrawl, and Infinispan/Ensemble clusters,
- Starting and stopping tcpflow[10] to record tcp connections for debugging and evaluation purposes,
- Starting YARN tests applications to verify the correctness of YARN installation,
- Collecting PCP and tcpflow metrics from the LEADS-platform instances to object storage.

We have chosen Saltstack, because we can define the LEADS platform (instances, networking, and required packages with configuration) using a declarative configuration files (YAML-based). Saltstack also includes tools to automatize the virtual machines creation and virtual machine provisioning. The YAML-based configuration files play a role of living documents and help us discuss its architecture internally. It is especially important for a distributed and diverse team like the consortium of LEADS. As an example, Listing 1 presents a map file that defines which query-engine instances running on which micro-cloud. In a different file (top.sls), we can specify what leads-qe instances are. Listing 2 presents this file. The top.sls specifies what should be installed on the query-engine instances. For example, the file leads/packages.sls (depicted in Listing 2 as leads.package) contains a list of common packages for LEADS components that should be installed.

```
ubuntu_large_dd2c:
    - leads-qe28
    - leads-qe29
    - …
ubuntu_large_dd2a:
    - leads-qe38
    - leads-qe39
    - …
```

Listing 1 A fragment of Saltstack map that maps query engine to instances on micro-clouds

[8] http://saltstack.com/
[9] http://www.fabfile.org/
[10] https://github.com/simsong/tcpflow

```
base:
  'leads-qe*':
    - leads.packages
    - leads.adidas_plugin_deps
    - leads.java
    - leads.setup_script
```

Listing 2 A fragment of top.sls that specifies what needs to be installed on QE instances

```
mypkgs:
    pkg.installed:
        - pkgs:
            - unzip
            - python-pip
            - python-dev
            - libffi-dev
            - libssl-dev
            - python-virtualenv
```

Listing 3 A fragment of leads/packages.sls file

The configuration platform helps us to have an agile (add and remove notes) and deterministic infrastructure (its definition is versioned in git). The Salt stack-based provision is a part of a larger effort to integrate all the LEADS components. The configuration of the LEADS test cluster and individual machines can be found in a GitHub repository[11].

9. Conclusion

This document presents the efforts taken to evaluate the correctness and performance of the integrated LEADS platform using the representative application in a multi-cloud environment and the widely used AMPLab benchmark. The tests were conducted on Cloud&Heat mico-clouds.

The results of the evaluation are good and show that the LEADS platform provides good performance to adidas application. With AMPLab tests, we validate that the LEADS platform can cope with big-data workloads as well.

[11] See: https://github.com/skarab7/leads_query-engine