LarKC

The Large Knowledge Collider
a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D11.5 Evaluation of instrumentation and monitoring tools and methods

Coordinator: Raluca Brehar (UTC)
With contributions from: Raluca Brehar (UTC), Ioan Toma (SG), Ionel Giosan(UTC), Mihai Negru (UTC), Andrei Vatavu (UTC), Silviu Bota (UTC), Mihai Chezan (SG)
Quality Assessor: Mark Greenwood - (University of Sheffield)
Quality Controller: Sergiu Nedevschi (UTC)

Document Identifier:	LarKC/2008/D11.5/V1.0
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	version 1.0.0
Date:	September 29, 2011
State:	final
Distribution:	public

Executive Summary

Evaluation is one of the major tasks in assessing the quality and the significance of any software product and of any research outcome. This deliverable has as it's main goal the evaluation of the instrumentation and monitoring components produced by WP11 for the instrumenting and monitoring of LarKC plugins, workflows and platform. These tools collect different measurements about the behavior of the above mentioned components and they also record information about the system on which the platform is running. The main tools are the instrumentation components (that is formed of the instrumentation code, agents, server) for setting up the measurement process and collecting the monitoring data, a visualization component that creates views and displays monitoring data collected from the system, platform, plug-ins and workflows and the relevance feedback component that makes use of data mining and performs machine learning on top of the data in order to make estimations of the values of the metrics recorded by instrumentation.

This deliverable describes a systematic determination of the quality and significance of the main instrumentation and monitoring components and for evaluation it uses different criteria like flexibility, robustness, data storage capacity, accuracy.

The evaluation is performed on top of a large knowledge base composed of measurements acquired while running queries and workflows from LarKC use-cases.

Document Information

IST Project Number	FP7 – 215535	**Acronym**	LarKC
Full Title	The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search		
Project URL	`http://www.larkc.eu/`		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	**Number**	11.5	**Title**	Evaluation of instrumentation and monitoring tools and methods
Work Package	**Number**	11	**Title**	Instrumentation and Monitoring

Date of Delivery	**Contractual**	M42	**Actual**	30-September-11
Status	version 1.0.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)				
Resp. Author	Raluca Brehar		**E-mail**	raluca.brehar@cs.utcluj.ro
	Partner	UTC, Softgress	**Phone**	+40 264 404 484

Abstract (for dissemination)	Evaluation is one of the major tasks in assessing the quality and the significance of any software product and of any research outcome. This deliverable has as it's main goal the evaluation of the instrumentation and monitoring components produced by WP11 for the instrumenting and monitoring of LarKC plugins, workflows and platform. These tools collect different measurements about the behavior of the above mentioned components and they also record information about the system on which the platform is running. The main tools are the instrumentation components (that is formed of the instrumentation code, agents, server) for setting up the measurement process and collecting the monitoring data, a visualization component that creates views and displays monitoring data collected from the system, platform, plug-ins and workflows and the relevance feedback component that makes use of data mining and performs machine learning on top of the data in order to make estimations of the values of the metrics recorded by instrumentation. This deliverable describes a systematic determination of the quality and significance of the main instrumentation and monitoring components and for evaluation it uses different criteria like flexibility, robustness, data storage capacity, accuracy. The evaluation is performed on top of a large knowledge base composed of measurements acquired while running queries and workflows from LarKC use-cases.
Keywords	Instrumentation, Monitoring, Evaluation

Project Consortium Information

Participant's name	Partner	Contact
Semantic Technology Institute Innsbruck, Universitaet Innsbruck		Prof. Dr. Dieter Fensel Semantic Technology Institute (STI), Universitaet Innsbruck, Innsbruck, Austria Email: dieter.fensel@sti-innsbruck.at
AstraZeneca AB		Bosse Andersson AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA		Emanuele Della Valle CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA Milano, Italy Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O.		Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universitaet Stuttgart		Georgina Gallizo Höchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany Email : gallizo@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Dr. Lael Schooler, Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ontotext AD		Atanas Kiryakov, Ontotext Lab, Sofia, Bulgaria Email: naso@ontotext.com
SALTLUX INC.		Kono Kim SALTLUX INC Seoul, Korea Email: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT		Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany Email: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Prof. Dr. Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK Email: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM		Prof. Dr. Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands Email: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY		Prof. Dr. Ning Zhong, THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan Email: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER		Dr. Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France Email: brennan@iarc.fr
INFORMATION RETRIEVAL FACILITY		Dr. John Tait, Dr. Paul Brennan, INFORMATION RETRIEVAL FACILITY Vienna, Austria Email: john.tait@ir-facility.org

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA `http://www.utcluj.ro/`		Prof. Dr. Eng. Sergiu Nedevschi TECHNICAL UNIVERSITY OF CLUJ-NAPOCA Cluj-Napoca, Romania E-mail: sergiu.nedevschi@cs.utcluj.ro
SOFTGRESS S.R.L. `http://www.softgress.com/`	Softgress	Dr. Ioan Toma SOFTGRESS S.R.L. Cluj-Napoca, Romania E-mail: ioan.toma@softgress.com

Table of Contents

List of Figures

List of Acronyms

LarKC	Large Knowledge Collider
RDF	Resource Description Framework
RRD	Round-Robin Database
RDBMS	Relational Database Management System
SIM	Semantic Instrumentation and Monitoring
GWAS	Genome Wide Association Study
SNP	Single Nucleotide Polymorphism
LLD	Linked Life Data

1. Introduction

Evaluation is one of the major tasks in assessing the quality and the significance of any software product and of any research outcome. This deliverable has as main goal exactly the evaluation of the instrumentation and monitoring components produced by WP11 for LarKC – a platform that enables the development of large scale reasoning applications using and combining techniques from various Semantic Web related research fields. The main entities of LarKC are plug-ins that may be interconnected in a workflow in order to perform a reasoning task. LarKC platform allows a flexible interconnection of the plug-ins and provides support for running and testing them.

A set of tools built by WP11 enables instrumentation and monitoring of LarKC plugins, workflows and platform. These tools collect different measurements about the behavior of the above mentioned components and they also record information about the system on which the platform is running. These tools comprise the following main components: instrumentation component (that is formed of the instrumentation code, agents, server) having the role of setting up the measurement process and collecting the monitoring data, visualization component that creates views and displays monitoring data collected from the system, platform, plug-ins and workflows and the relevance feedback component that makes use of data mining and performs machine learning on top of the data in order to make estimations of the values of the metrics recorded by instrumentation.

This deliverable describes a systematic determination of the quality and significance of the main instrumentation and monitoring components and for evaluation it uses different criteria against a set of standards.

The evaluation is performed on top of a large knowledge base composed of measurements acquired while running queries and workflows from LarKC use-cases. The knowledge base contains measurements related to plug-ins, workflows, metrics describing the behavior of the platform and of the system on which the platform was run.

This deliverable is organized as follows: in chapter 2 we shortly describe the components that are evaluated, the tools used for evaluation and the criteria against which the evaluation is done. This chapter revises shortly the functional and non-functional requirements identified in the previous deliverables and shows to what degree these requirements have been met by the existing components. Chapter 3 presents the test cases and the experimental setup for evaluation while chapter 4 concludes the deliverable.

2. Evaluation criteria and tools description

In this chapter we provide a brief description of the tools that have been used for evaluation and specific functions that are implied in the evaluation procedure.

The instrumentation and monitoring solution for LarKC is composed of:

- Instrumentation component – collects measurements describing the behavior of the system, of LarKC platform, plugins and workflows run on the platform.
- Visualization and monitoring component – provides a graphical description of the instrumentation data.
- Relevance feedback component – performs data mining on top of instrumented data.

Each of these tools can be evaluated against a set of criteria. We have identified the most relevant evaluation standards for our instrumentation and monitoring solution.

2.1 Evaluation criteria

We assess the quality and reliability of the developed components against a set of well defined criteria. We have identified the following major evaluation issues:

- System rigidity – analyzes the behavior of the components with respect to changes - for example, when new instrumentation measurements are introduced.
- Robustness – refers to the ability of a computer system to cope with errors during execution or the ability of a component to continue its operation despite abnormalities in input.
- System overhead – describes the additional usage of computer resources (memory, execution time) for performing the instrumentation and monitoring task.
- Data storage capacity – describes the behavior of the storage component against different volumes of data.
- Accuracy – analyzes the degree of closeness of the outcomes produced by the components with respect to the true measured values.

All these functions have been evaluated (where possible) for each of the three components. In our evaluation we have used queries generated from two use-cases (WP6 and WP7b). The data generation procedure is described in chapter 3.

2.1.1 System rigidity

To measure the rigidity of our system and its components, we look at the behavior of the system with respect to changes. Many dimensions can be varied and the changes in the system can be observed, including: new instrumentation measurements are produced, new metrics are introduced, the system is used to monitor another application, etc.

Instrumentation

Flexibility is one of the main feature of the instrumentation module. Conceptually the instrumentation module is split in two parts, one providing a generic instrumentation functionality, the other providing support for instrumenting LarKC specific metrics. The generic module can be used for instrumenting any application without changing any line of code in the target instrumented application. The measurements that can be collected refer to generic metrics (e.g. CPU and memory usage, etc.) For application specific metrics the instrumentation module needs to be extended to support these new metrics. Same for any new metrics that are introduced. The module is flexible however when the new metrics can be defined in terms of existing metrics already supported and the effort is lower than implementing and supporting them from scratch. To conclude the instrumentation module is generic and can be applied easily for other applications. Extensions are needed when new metrics are introduced.

Visualization

The Visualization module is very flexible when new instrumentation metrics and measurements are introduced. The portlet-based architecture we have developed allows us to easily display information about new metrics that were added to the LarKC platform. All we have to do is insert a new metric entry into the databases. Then the visualization will be able to read these new metrics and display their values.

Relevance Feedback

The Relevance Feedback module can be easily extended in order to predict any new output performance measure of a query. The only condition that must be fulfilled in order for a metric to be predicted is that the metric is measured and recorded for the dataset with which the prediction model is trained.

2.1.2 Robustness

We refer to robustness as that desirable property of a system that is able to cope with errors during execution or the ability of a component to continue its operation despite abnormalities in input.

Instrumentation

In a broader sense the instrumentation includes the actual instrumentation module, the agents and the server component that stores the monitoring data. Each of these modules or components can function even if the other components are not working properly or not working at all. For example an agent will continue to work even if it receives no measurements from the instrumentation module. The instrumentation module is also fault-tolerant being able to cope with erroneous input.

Visualization

The Visualization module is quite robust, it proves to have an increased fault tolerance. If the user introduces some wrong inputs the module is able to display some nice warnings or error messages. All Exceptions are caught and logged in the Apache Tomcat log files.

Relevance feedback

The Relevance Feedback module has a high fault tolerance. If the user input is not the expected one or if the files/database selected by the user are not correctly formatted, warning messages are displayed. Missing values are successfully handled in the training and test methods. Spurious and noisy data are properly removed so that they do not influence the prediction models which are computed. The mentioned capabilities all contribute to the robustness of the module.

2.1.3 Data storage capacity

Instrumentation

Instrumentation is the WP11 responsible for collecting measurements and sending it to the server component which stores it. As described in D11.4 there are various storage mechanisms that are supported in the server module, including RDF, RRD and RDBMS storage. RDF storage is the main storage. WP11 relies on state of the art RDF storage servers, namely OWLIM and Sesame. The capacity and performance of the data storage thus depends on the underlying RDF storage being used. In all our experiments on the queries generated for the use-cases we have generated and stored about 10^9 metric instances.

Visualization

The visualization module uses mainly a MySQL based database server where all the relevant metrics obtained by the instrumentation and monitoring tools are recorded. Our experiments proved that the MySQL database is able to deal with large amounts of data, approximately 10^7 metric instances.

Relevance feedback

The Relevance Feedback module can operate with up to 10^6 training dataset instances (and each instance is characterized by about 10^2 metrics). The running time for the training phase increases with the number of instances, but a parallel version of the module is also implemented and it offers reasonable running times.

2.1.4 Accuracy

Visualization

The accuracy of the visualization module is highly dependent on the accuracy of the instrumentation. Even if the amount of metric data is very large, the visualization module is able to represent these metrics with high accuracy. An increased accuracy is obtained if the user selects a shorter period of time for the visualization.

Relevance feedback

The accuracy of the Relevance Feedback module is highly dependent on the number and variation of queries in the training dataset. The module behaves optimally when there is a large number of training instances and there is an approximately equal proportion of them for each of the possible types of queries and these types can be clearly delimited. The input metrics values that characterize the attributes of the training data should be highly variated from one query type to another.

2.1.5 Overhead

To assess the overall performance of the instrumentation and monitoring solution, we also measure and analyze the overhead introduced by the solution. Overhead is defined as the additional usage of computer resources (memory, execution time) for performing the instrumentation and monitoring task. If instrumentation has too much overhead (like most profiling tools) then it can not be run all the time on production machines because the performance penalty is too high. We designed SIM to be an instrumentation/monitoring that can be deployed and activated to run all the time on production environment.

To find out how much overhead SIM introduces for LarKC we run a series of tests with and without SIM. The system configuration we use to perform the tests is:

- CPU: i5-2520M CPU @ 2.50GHz
- Memory: 8GB DDR3
- LarKC version: 2.5
- JVM options: -Xmx2048M
- LarKC plugin directory containing 64 plugins

For testing we used LarKC-auto-query, an application we developed to do automating testing of Larkc. The tool can be downloaded from `https://github.com/semantic-im/larkc-auto-query`

We simulated 10, 20 and 30 concurrent clients. Each client was started at 1 second interval one from each other. Each client sent queries to LarKC at a 0.1 seconds rate. For 10 clients test simulation we used SparqlQueryEvaluationReasoner_0 to SparqlQueryEvaluationReasoner_9 (copied from directory “examples” to “queries-to-execute”). Total queries executed for this test was 4307. For 20 clients test simulation we used the same test data but in duplicate. Total queries executed for this test was 8614. For 30 clients test simulation we used the same test data but in triplicate. Total queries executed for this test was 12921.

LarKC “load time” was calculated as the time difference between “Starting LarKC” and “Done initializing LarKC” log messages.

We obtained the following results:

- SIM disabled
  - 4767 total JVM loaded classes
  - 18s LarKC load time
  - 80s 10 clients 4307 queries
  - 100s 20 clients 8614 queries
  - 126s 30 clients 12921 queries
- SIM enabled

  - 5396 total JVM loaded classes
  - 18s LarKC load time with sim pre-compiled into LarKC
  - 23s LarKC load time with LTW
  - 83s 10 clients 4307 queries
  - 104s 20 clients 8614 queries
  - 136s 30 clients 12921 queries

It can be observed that when using LTW (load time weaving) SIM introduces an overhead during loading of LarKC. This is to be expected because when using LTW any class loaded by JVM is first checked if needs to be instrumented and if that is the case code is injected into the class byte-code. This of course takes time and is direct proportional to the number of classes loaded by the JVM. If this is an issue, the solution is to pre-compile SIM into LarKC. By doing this, instrumentation code is injected when LarKC is compiled. With this approach the time to load LarKC is not modified by SIM. Using LTW still brings benefits as there might be cases when it is desirable to enable/disable SIM in an easy way. With LTW this is possible by just changing a command line option when LarKC is started. The alternative with pre-compiling SIM into LarKC is to have two builds: one with SIM pre-compiled into LarKC and another with LarKC without SIM. Another case where LTW could be needed is when the classes needed to be instrumented are available only at runtime (so pre-compilation is not possible in this case).

When running LarKC it can be seen that the overhead increases, from 3% to 4% and then to 8%, as more clients are added. When we add more clients to the test, LarKC gets to used almost 100% CPU. In this situation any CPU cycle counts and work performed by SIM is work taken from LarKC. By running tests with parts of SIM disabled we identified how much time each of these parts take:

- metrics collectors and agent communicator 28%
- method metrics - thread metrics: 42%
- method metrics - gcc and memory metrics: 28%
- LarKC metrics: 1%
- Platform metrics: 1%

We can see that, by far, method metrics take the most time to compute, and the most expensive part is the thread metrics. Thread metrics extract thread count, wait, block and cpu time (user, system, total). In case these are not needed and wall clock time is enough, then they can be disabled. Another option is to not instrument methods that are called too many times per second. Another option is to implement sampling option to sim-instrumentation module for these methods that don't take much time to execute but are executed many times per second. In case of metrics collector, ConcurrentLinkedQueue is used as a buffer to temporary store metrics until they are sent to the Agent by the agent communicator. If we have a high number of threads that generate a lot of metrics, using another thread safe data structure, that are designed for this case, could improve performance. Regarding the agent communicator, we use Java serialization and HTTP protocol to send collected metrics to the Agent. To improve performance some performance could be gained by tuning the metrics objects that are serialized. Also, implementation of another serialization mechanism could be explored, like google protocol buffers - this would also open up SIM to be used by other languages, not only Java, as google protocol-buffers serialization is language neutral. Another possibility is to replace HTTP with a "lighter" communication protocol.

2.2 Tools used for evaluation

AutoQuery

LARKC Auto Query is intended to be used as a tool to automate testing of LarKC [1] by creating workflows and sending queries to them.

LARKC Auto Query uses bash scripting and curl [2] to connect to LarKC REST APIs, create workflows and send queries to them.

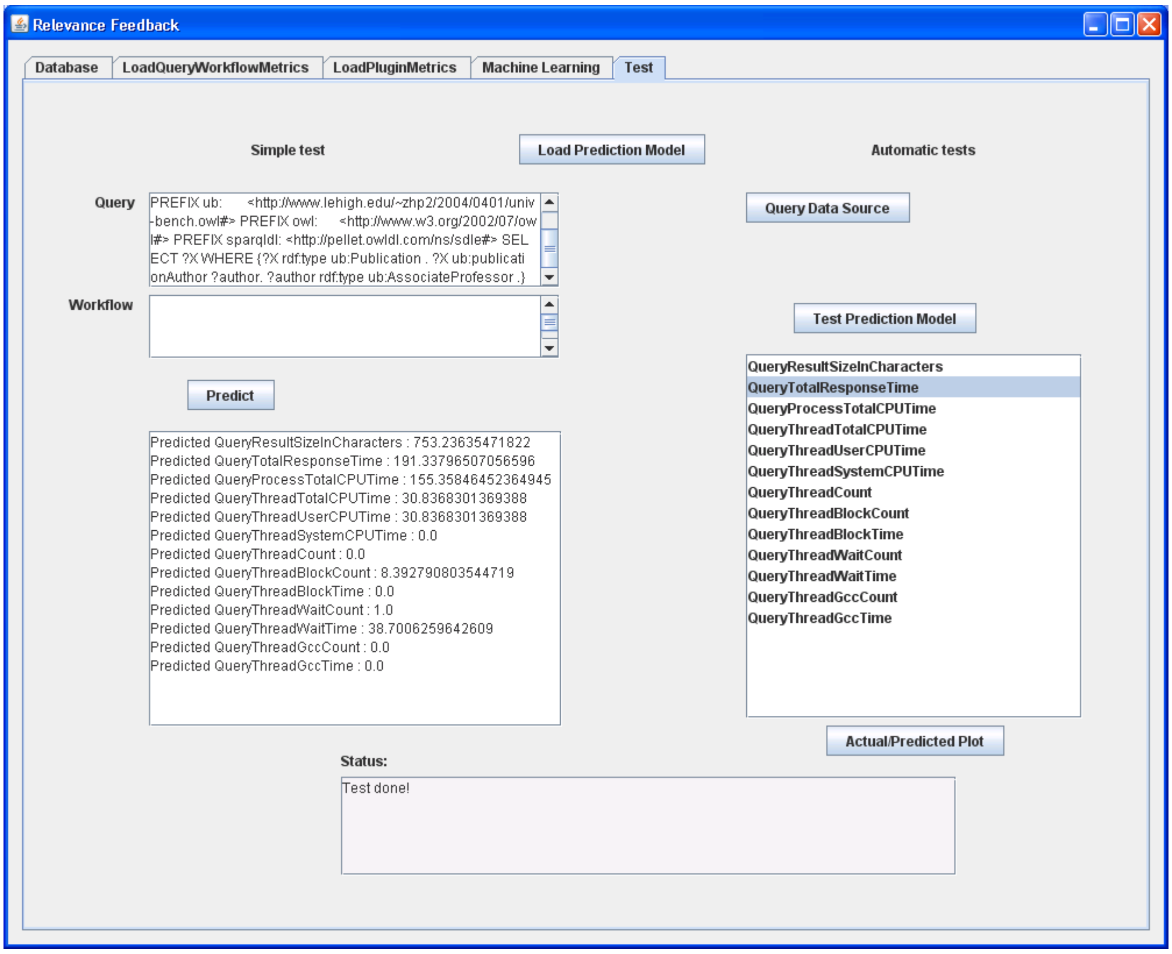
The code is split into three parts:

1. **script/larkc-access-functions.sh**: a library that contains functions to access Larkc REST APIs. It was built so it could be reused by other scripts. The implemented functions are:
   - larkc_is_running – check to see if LarKC is running.
   - larkc_submit_workflow – submits the workflow description to the platform and returns the workflow identifier.
   - larkc_get_workflow_endpoint – obtains the endpoint URL of the given workflow identifier.
   - larkc_delete_workflow – deletes the workflow identified by the workflow id.
   - larkc_query_endpoint – sends a query to the given endpoint and returns the result.
   - larkc_file_query_endpoint – sends a query, read from file, to the given endpoint and returns the result.
2. **script/autoquery.sh**: is a script that takes as arguments a “workflow directory”, creates the workflow from the file workflow.ttl and then sends the queries from “query*” or “queries*” files to the created workflow.
3. **run.sh**: main script for starting Larkc Auto Query Tool.

In order for the script to work, a certain directory structure is expected. Directory “queries-to-execute” should contain other directories, called workflow directories. Each of these workflow directories must contain one file named “workflow.ttl” which should contain the workflow intended to be created on LarKC. It should also contain one or more files named “queries*” or “query*” which should contain the queries to send to the created LarKC workflow. Files named “queries*” are assumed to contain more than one query separated by new line. Files named “query*” are assumed to contain just one query (that can spawn on multiple lines).

For each of the found “workflow directories”, a new process is executed. This process will read the workflow.ttl file and send it to LarKC to create the workflow.

[1] http://www.larkc.eu/
[2] http://curl.haxx.se/



```
PREFIX ub:      <http://www.lehigh.edu/~zhp2/2004/0401/univ
-bench.owl#> PREFIX owl:      <http://www.w3.org/2002/07/ow
l#> PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#> SEL
ECT ?X WHERE {?X rdf:type ub:Publication . ?X ub:publicati
onAuthor ?author. ?author rdf:type ub:AssociateProfessor .}
```

```
Predicted QueryResultSizeInCharacters : 753.23635471822
Predicted QueryTotalResponseTime : 191.33796507056596
Predicted QueryProcessTotalCPUTime : 155.35846452364945
Predicted QueryThreadTotalCPUTime : 30.8368301369388
Predicted QueryThreadUserCPUTime : 30.8368301369388
Predicted QueryThreadSystemCPUTime : 0.0
Predicted QueryThreadCount : 0.0
Predicted QueryThreadBlockCount : 8.392790803544719
Predicted QueryThreadBlockTime : 0.0
Predicted QueryThreadWaitCount : 1.0
Predicted QueryThreadWaitTime : 38.7006259642609
Predicted QueryThreadGccCount : 0.0
Predicted QueryThreadGccTime : 0.0
```

```
Test done!
```

Figure 2.1: Automatic evaluation environment

Then queries will be read from the "query*" and "queries*" files and send one after another to the created LarKC workflow.

Directory "examples" contain examples of such directory structure. Part of all of the examples directories can be copied into directory "queries-to-execute".

The file configuration.sh is the place where LarKC Auto Query can be configured. It allows to define LarKC platform URL location and the workflows endpoint name. Endpoint name must match the name from workflow description files (workflow.ttl).

LarKC Auto Query needs bash and curl in order to work. Also LarKC needs to be running. To start it simple execute the "run.sh" script.

Relevance feedback – automatic evaluation of accuracy

The automatic evaluation of prediction accuracy can be made either for a single query or for an entire test dataset (see Figure 2.1). In order to be able to perform the evaluation, a prediction model for a certain scenario needs to be loaded by pressing the "Load Prediction Model" button. Next, if we want to make a simple prediction, the user inputs the query's text into the text field and presses the "Predict" button and the query metrics prediction results will appear in another text field.

However, in order to have reliable accuracy evaluation, the accuracy needs to be computed for a larger dataset. An ARFF file containing more test instances can be loaded by pressing the "Query Data Source" button. By pressing the "Test Prediction

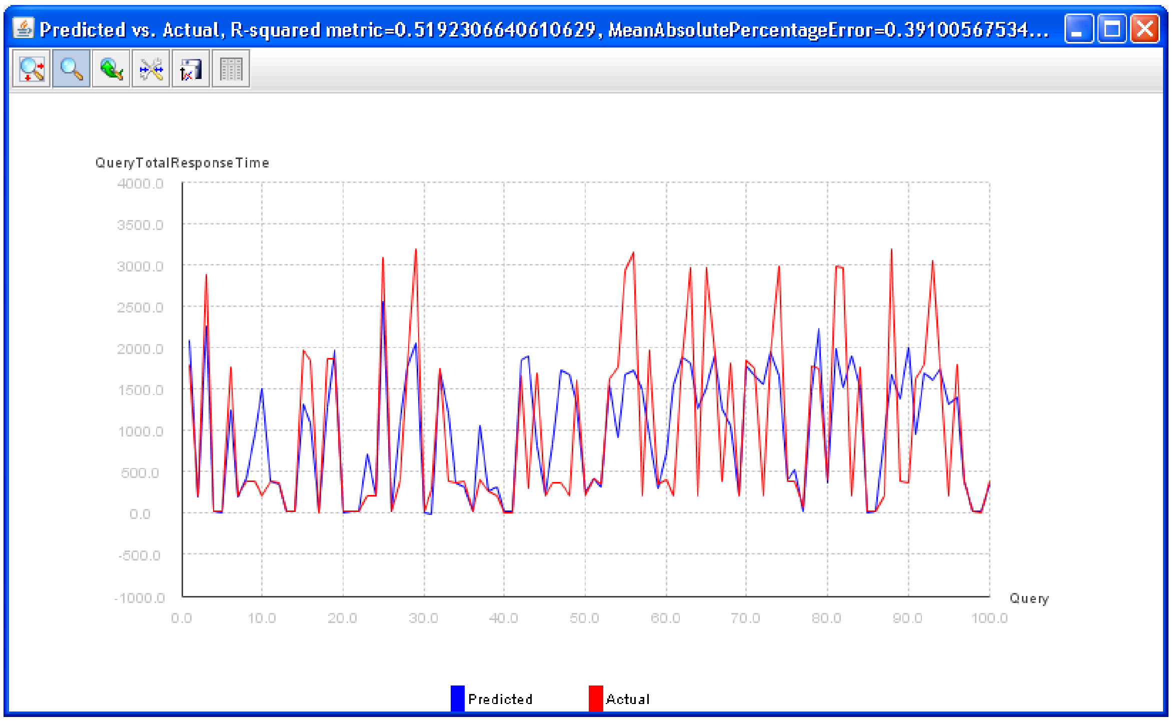


Figure 2.2: Automatic test chart with accuracy values

Model" button, an automatic testing process is started, each test instance being evaluated. A chart showing the actual and predicted values for each query performance metric can be visualized by pressing the "Actual vs. predicted plot" (see Figure 2.2). The values of two accuracy metrics can be seen in the title bar for each chart.

2.3 Compliance with respect to initial requirements

In our deliverable D11.1.1 [1] we have defined a set of specific requirements for each component. We will briefly revise these requirements and explain the degree to which they were attain.

2.3.1 Instrumentation requirements

A set of functional and non-functional requirements were defined in D11.1.1 [1] for the instrumentation component. In this section we revisit these requirements and describe how they are fulfilled. One of the functional requirements was "instrumentation should be done at the level of method, JVM, system and application". Instrumentation "must be at different levels i.e. plug-in, work-flow, platform". Both generic and LarKC specific instrumentation at the level specified in the requirement are supported by W11 instrumentation. Another functional requirement was to specify the metrics of interest using "various mechanisms e.g. annotations, API, library, etc." As described in D11.4 [2] various means to specify and collect data have been made available as part of WP11 instrumentation. Another functional requirement defined in D11.1.1 [1] was "instrumentation should make the measurements available to other components of the monitoring platform either through a pull or push approach.". WP11 instrumentation implements the push approach, namely data is sent to the monitoring agents.

In terms of non-functional requirements, one requirement was "to have a low overhead, such that it has no real performance impact on the instrumented application runtime". The overhead introduced by the instrumentation is relatively small. We performed a set of experiments with running the platform and executing workflows with the instrumentation turned off and on, and the difference in time is very small. WP11 instrumentation "should be characterized by low latency". This is actually the case as measurements are gathered and pass fast to the monitoring agents. "The instrumentation should be easy to use and should introduce a minimal overhead for the developer". WP11 instrumentation is easy to used. An application can be instrumented and generic metrics can be collected by simply running the application with specific parameters and including instrumentation jar in the path. Another requirement for instrumentation is portability. WP11 instrumentation can run on a wide spectrum of hardware and operating systems, is multi-platform, multi-operating system. Furthermore, instrumentation can turn off or on.

2.3.2 Visualization requirements

Most of the functional requirements described in [1] have been fulfilled.

The visualization module is an interactive component that provides a two-way communication channel. First it is able to display the stored metrics via tables, pie charts, line charts and bar charts. Second it allows the users to select what they intend to visualize. It allows the user to input the type of query they want to construct, from what category of metrics (system, platform, workflows, queries, plugins). Advanced users are even able to introduce their own MySQL queries in order to prepare a custom view for the desired portlet.

Functional Requirements

- The visualization component is able to correctly display context information about the instrumented LarKC platforms, workflows, queries, plugins and about the systems were these platforms were run on.
- The visualization component is able to interact with the user. A user can adjust the visualization parameters for each portlet (the time interval used for visualization, the level used for visualization (platforms, workflows, queries, plugins), the metrics used for visualization, the visualization type table, charts, etc.).
- Users with administration permissions may customize the visualization dashboard - location of the portlets, how many portlets to see on a given page, etc.
- The visualization component interacts with the data storage facilities.
- The visualization component is able to provide a more intuitive visual representation of the stored data.

Non-functional Requirements

- Integration:
  - The visualization functions and data are grouped in portlets. In order for a user to easily find the portlets, they are grouped into the LarKC category of the Liferay portal.

- Access Control:
  - The user must login in order to be able to customize the portlets and dashboard.
- Easily editable content:
  - The content provided by the visualization module is easily editable. Advanced users are able to construct more complicated visualization scenarios by introducing new mysql queries in each portlet's configuration.
- Extensibility:
  - The visualization component modules are easy to install. However in order to extend these modules one needs a good understanding about the liferay portal and portlet development.
- Back-up/Restore:
  - The system is able to back-up the configuration of each portlet instance and restore it when the visualization is restarted.
- Response Time:
  - The visualization component is able to provide a fast response time. The response time depends on the amount of data that needs to be visualized and the time interval used for visualization.
- Availability:
  - The visualization platform should be available for use at any time, since it will be deployed together with the LarKC platform.
- Portability:
  - The visualization component runs on main operating systems (Windows, Linux, etc.).
- Usability:
  - The visualization component is a simple, easy to use and understand web application.

2.3.3 Relevance feedback requirements

Most of the functional requirements described in [1] have been fulfilled. That is we have acquired a robust interaction with the data storage and aggregation component by constructing an access interface to all storage mechanisms (RDF, SQL, RRD).

The ability of the relevance feedback component to correctly predict the instrumentation measurements on new data is highly dependent on the knowledge base on which the training was carried out. We cannot make any estimations for workflows that have not been run on the platform, or for plug-ins that are not in the relevance feedback knowledge base. Feature selection (correlation based feature selection) and data clustering (expectation maximization) has been successfully tested and applied.

We have also meet the non-functional requirements mentioned in [1]. The relevance feedback has low computational costs and the measured time for the largest overhead we could generate is less than 3s. The code written for training the machine learning algorithms has been parallelized and hence a great improvement in speed has been gained.

The robustness of the component is ensured by the implemented algorithms that deal with missing values or with noisy data.

Portability is ensured by the fact that the relevance feedback component has been implemented in Java and has been tested on Linux and Windows operating systems.

3. Experimental set-up for evaluation

In order to evaluate the developed tools we need to gather as much instrumentation data as possible. To obtain this data we have run 10^5 queries for different workflows from existing LarKC use-cases.

3.1 Data generation

For the data generation we have collected queries for the use cases (WP6 and WP7b) and for other workflows that were running on version 2.5 of the platform[1]. We run the experiments on a 32-bit Windows machine, with 4GB of memory and on a 64-bit Windows machine with 4GB of memory.

3.1.1 Workflows from WP6

For this use-case we have considered three sub-workflows: path, event, and monument finding, i.e. the workflows that compose Urban LarKC [2]. For stress testing the platform we have envisioned the repeated execution of 10^2 real user queries and the execution of 10^4 synthetically generated queries (for the path finding workflow).

Urban Event

The workflow is composed of the following plug-ins: SourceSplitter, SparqlToCity-QueryTransformer, EventIdentifier, XML2RDFTransformer, SparqlQueryEvaluation-Reasoner.

The description of the workflow is provided in A.1.

The pattern for the queries of this workflow are provided in A.2. We have replaced Milan with other city names, and we have found about 50 different cities for which the workflow provided results. Some queries were fed several times to the platform and in total, for this workflow we ran about 10^2 queries. The workflow and the generated queries have the role of finding the events taking place during a given time interval in a given city.

Urban Monument Finder

This workflow is composed of the following plug-ins: SourceSplitter, UrbanSindiceI-dentifier, SparqlQueryEvaluationReasoner.

The workflow description is given in appendix B.1.

Unfortunately for the purpose of instrumentation based evaluation for this workflow only one query works: finding all the visitor attractions in Milan.

The query is listed in B.2.

Path finder

This workflow is composed of the plugins: SourceSplitter, RemoteGraphLoaderIden-tifier, OpResPathFinderReasoner.

The description of the workflow is provided in appendix C.1.

[1] All experiments were performed on v2.5 of the platform, commit revision 1797.

[2] http://wiki.larkc.eu/LarkcProject/WP6

A sample query for this workflow is listed in C.2. We have replaced the node identifiers (node8252 and node8253) with other nodes that have been provided to us by WP6. We have synthetically generated about 10^4 queries.

3.1.2 Workflows from WP7b use-case

For the WP7b use-case [3] we have used the workflow formed of the plug-ins: GWASIdentifier and SOStoVBtransformer.

The description of the workflow is given in D.1. Given a list of diseases and a list of keywords related to the diseases, it returns a score for each SNP ID given either within the SPARQL query or a parameter file. The target is to find the SNP IDs having the highest ranking values.

We have synthetically generated about 10^4 queries having the same pattern as in appendix D.2

In order to obtain different queries we have modified the SNPIds, the number of SNPIds, and the diseases in the query. The above example contains three SNPIds (rs1051730, rs401681, rs2736100) and it refers to lung cancer. We have used a list of about 10^7 SNPIds provided by WP7b and as diseases we referred to: 'lung cancer', 'kidney cancer', 'prostate cancer', 'breast cancer' and 'type 2 diabetes'.

3.1.3 Other workflows

We have also considered other workflows that were available at the moment of evaluation on version 2.5 of the platform. The workflows we have run are formed of the following plugins:

- LLDReasoner, SOStoVBtransformer
- SPARQLDLReasonerPlugin
- CRIONReasonerPlugin, SOStoVBtransformer
- OWLAPIReasonerPlugin
- PIONReasonerPlugin, SOStoVBtransformer

The first workflow (LLDReasoner, SOStoVBtransformer) in the list above has been also included in our evaluation experiments as we have generated a relatively large number of queries for it. For the other workflows the amount of generated queries is rather small and we have not considered them in our evaluation setup.

The description the workflow LLDReasoner, SOStoVBtransformer is provided in E.1 The workflow can be given queries having the form shown in E.2. We have synthetically generated about 10^3 queries by varying the disease in the query.

3.2 Storage of instrumentation data

The queries we have run for the use-cases have generated about 10^9 metric instances in the RDF storage and 10^7 metric instances in the SQL storage. The difference between RDF dimension and SQL dimension comes from the fact that in SQL we don't store all

[3]http://wiki.larkc.eu/LarkcProject/WP7b

the RDF metrics only the ones specific for plug-ins, workflows, platform and system. In RDF other method metrics are stored and hence the number of instances is greater.

For the RDF storage we have used OWLIM [4] because it is the best RDF storage available on the market capable to load tens of billions of RDF statements, using non-trivial inference and delivers outstanding multi-user query performance. This perfectly fits our storage requirements (10^9 metric instances). It is also suitable for the aggregation task as we periodically perform SPARQL queries against the repository. Furthermore the new introduced support for SPARQL 1.1 allows data aggregation directly in the storage layer. This is however a feature that we plan to support in future releases of SIM. In our experiments we have tried to use Sesame[5] but it does not scale to large datasets and that is why we moved to OWLIM.

3.3 Visualization of instrumentation data

The visualization tool enables easy interpretation of the data collected during instrumentation and monitoring thus it helps LarKC users and developers visualize real-time or historical data. Furthermore they can analyze reports based on aggregated metric values. Reports may include overviews of the monitored metrics for given time periods and aggregations of different metrics values both at the level of LarKC plug-ins, workflows and platform.

This section provides several examples of the visualization functionalities. A better understanding of visualization functions and facilities can be attained by accessing the on-line demo available at `www.larkc.utcluj.ro`. To log-in use as user name "demo" and as password "demo".

The visualization module is a portal containing several portlets. Each portlet is easily customizable and answers different user visualization demands. The portal can be extended by 3rd party widgets (using the standard JSR 168 and JSR 286 from the portlet specification)

The visualization interface contains:

- The dashboard that includes the widgets considered as being the most important for a given user and platform (example widgets can contain system metrics, platform measurements, workflow, plug-in or query related metrics).
- Visualization portlets that are included in the dashboard. The most important portlets are:
  - Metric Chart Portlet
  - Metric Table Portlet
  - Information portlet
  - Toolbar

3.3.1 Metric Chart Portlet

This portlet generates views of the type: Line Chart, Pie Chart, BarChart. It has two modes:

[4]http://www.ontotext.com/owlim
[5]http://www.openrdf.org/

- View Mode – Figure 3.1.
- Edit Mode – Figure 3.2.

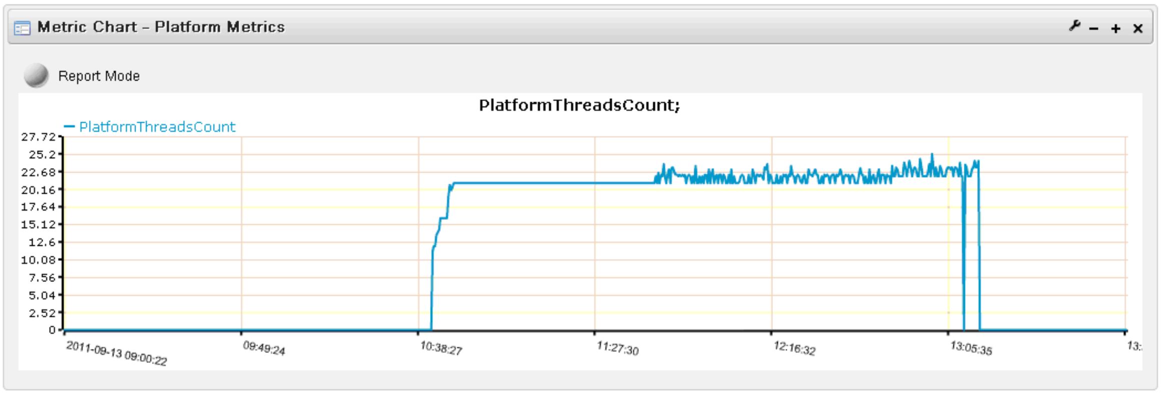


Figure 3.1: Frontend Interface (View Mode) for Metric Chart Portlet

The view mode displays the graphic for which the settings have been done in the edit mode.

The edit mode has several parameters. Figure 3.2 displays a screen shot of the edit mode. We shortly provide the description of each parameter, defining the values

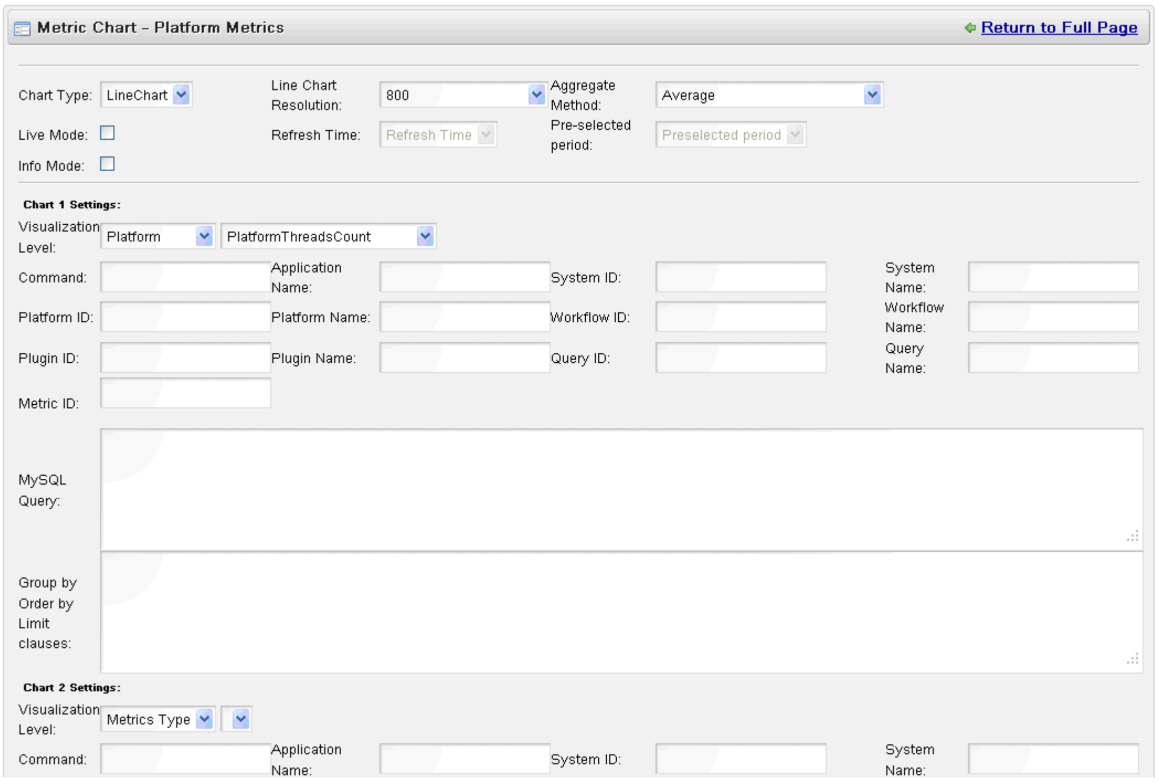

Metric Chart - Platform Metrics
Return to Full Page

Chart Type:	LineChart	Line Chart Resolution:	800	Aggregate Method:	Average		
Live Mode:	☐	Refresh Time:	Refresh Time	Pre-selected period:	Preselected period		
Info Mode:	☐						

Chart 1 Settings:

Visualization Level:	Platform	PlatformThreadsCount					
Command:		Application Name:		System ID:		System Name:	
Platform ID:		Platform Name:		Workflow ID:		Workflow Name:	
Plugin ID:		Plugin Name:		Query ID:		Query Name:	
Metric ID:							

MySQL Query:

Group by Order by Limit clauses:

Chart 2 Settings:

Visualization Level:	Metrics Type						
Command:		Application Name:		System ID:		System Name:	

Figure 3.2: Portlet Settings (Edit Mode) for Metric Chart Portlet

it may have.

- Chart Type: may have as values Line Chart, Pie Chart, Bar Chart.

- Line Chart Resolution – defines the number of bins on which the data set is displayed (large data sets are shrinked at a fixed resolution for visualization). The values it may have belong to the interval [100 . . . 1000]
- Aggregate Method – defines the aggregation method used for data compression. It may take the values Average, Average excluding zero values, Minimum Value, Maximum Value. Aggregation is done for each bin. For example if the resolution is 800 the aggregation will be done on 100 bins.
- Live Mode – if checked, the portlet will display real time data acquired during the last minute, last hour etc.
- Refresh Time – specifies the refresh rate – that is the time interval until a new query is sent to the server – for the Live Mode. The default value is 5000ms .
- Preselected Period – the time interval for which data is displayed for the live mode. The default value is the last minute.
- Info Mode – if checked it will display the generic settings and specific settings for each portlet. For example it will display information like: platform identifier and name, identifiers and names for plugins, queries, workflows, what metrics are displayed, visualization level.
- Chart 1 Settings, Chart 2 Settings: define the area in which we introduce specific values for the configuration of the visualization interface. For example: platform name, platform identifier, plugin, query, metric.
- Visualization Level – can be platform, system, workflow, plugin and query. Each level comprises metrics that are specific for the level.
- MySQL Query – is used for defining a custom query based on which visualization results are extracted.
- Group by, Order by, Limit clauses – can be added to the MySQL query.

3.3.2 Metric Table Portlet

This portlet generates context type visualization or it may display metrics in tables. It has two modes:

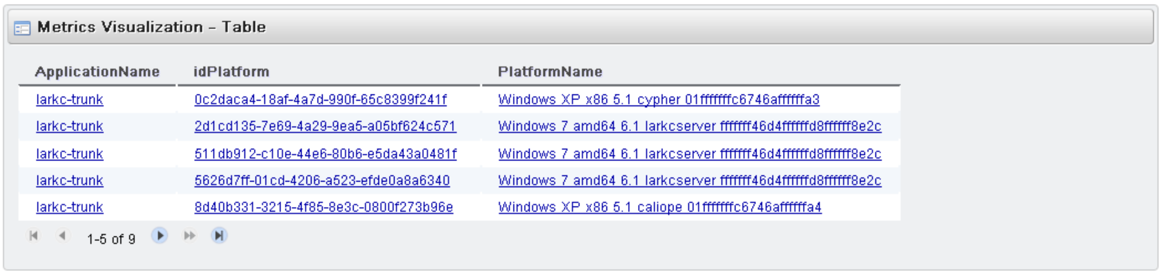
- View mode – Figure 3.3
- Edit mode – Figure 3.4

The view mode contains the front end interface.

The edit mode allows the users to set the parameters of the table to be displayed in the front end.

Table parameters may be:

- Level – used to display Context information or Metrics.
- Visualization - used to display context/metrics about: platforms, workflows, plugins, queries, systems

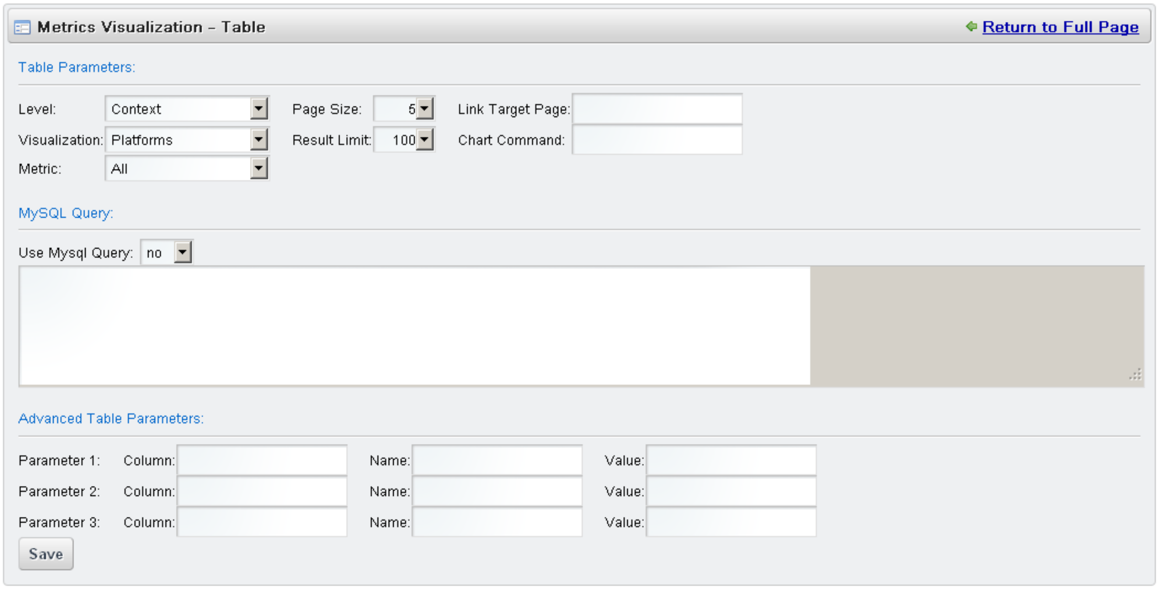


Metrics Visualization – Table

ApplicationName	idPlatform	PlatformName
larkc-trunk	0c2daca4-18af-4a7d-990f-65c8399f241f	Windows XP x86 5.1 cypher 01fffffffc6746afffffa3
larkc-trunk	2d1cd135-7e69-4a29-9ea5-a05bf624c571	Windows 7 amd64 6.1 larkcserver ffffff46d4fffffd8ffffff8e2c
larkc-trunk	511db912-c10e-44e6-80b6-e5da43a0481f	Windows 7 amd64 6.1 larkcserver ffffff46d4fffffd8ffffff8e2c
larkc-trunk	5626d7ff-01cd-4206-a523-efde0a8a6340	Windows 7 amd64 6.1 larkcserver ffffff46d4fffffd8ffffff8e2c
larkc-trunk	8d40b331-3215-4f85-8e3c-0800f273b96e	Windows XP x86 5.1 caliope 01fffffffc6746afffffa4

1-5 of 9

Figure 3.3: Frontend Interface (View Mode) for Metric Table Portlet



Metrics Visualization – Table

Table Parameters:

Level:	Context	Page Size:	5	Link Target Page:	
Visualization:	Platforms	Result Limit:	100	Chart Command:	
Metric:	All				

MySQL Query:

Use Mysql Query: no

Advanced Table Parameters:

Parameter 1:	Column:	Name:	Value:
Parameter 2:	Column:	Name:	Value:
Parameter 3:	Column:	Name:	Value:

Figure 3.4: Portlet Settings (Edit Mode) for Metric Table Portlet

- Metric – display all metrics or display a specific platform / workflow / plugin / query / system metric - this field is automatically changed according to the value selected in the visualization field
- Page Size - the number of rows to display in a page of the table. Drop down list with the following predefined values: 5, 10, 15, 20, 25, 30, 35, 40
- Result Limit – the number of results to obtain from the database
- Link Target Page – some values in the columns of the obtained table are used to set some GET parameters for the current page or for a given target page. For example (in the above table) if a user clicks on a platform id the value that was clicked will be passed as a GET parameter for the current page or for the specified target page, thus the next page will be dynamically reconfigured to take into account the submitted platform id.
- Chart Command – special parameter used to transmit to the metric chart portlet what kind of metrics it will display: platform / workflows / queries / plugins / etc. This parameter is similar to the Command parameter from the metric chart portlet and is also passed as a GET parameter. Thus the GET parameters have priority over the portlet's parameters.
- Mysql Query – this setting is for advance users, only. If someone desires to create a more specific visualization table (with multiple table joins, multiple data visualization, etc) he/she can introduce such a query here. The user must have a good knowledge of the database structure. An error is displayed in case the mysql query fails to execute.
- Use Mysql Query – drop down list with two values yes and no. If yes is chosen then the mysql query supplied in the text area below will be executed.
- Text Area – users must introduce a custom mysql query in this text area. The mysql query will have precedence over all other parameters (GET parameters and other portlet parameters)
- Advanced table parameters – these parameters are used in order to provide more GET parameters with a single click.
  - Column – the name of the column where to get the parameter
  - Name – the name of the parameter
  - Value – the value of the parameter

  For example:

  1. If we have: Column = idSystem; Name = idPlatform ; Value = blank the link will have the following form: url?idPlatform=ValueClickedInTableOnIdSystemColumn
  2. If we have: Column = blank ; Name = chartType ; Value = pieChart the url will have the following form url?chartType=pieChart

As the visualization component is highly customizable a large number of charts can be generated. We will display some example charts for platform, for workflows, for plug-ins and for the queries that were send to the platform.

The number of metrics that are currently instrumented are displayed in Table 3.1.

Platforms metrics	16
Workflow metrics	27
Plugin metrics	32
Query metrics	41
System metrics	29
Total	145

Table 3.1: Number of instrumented metrics used for visualization

3.3.3 Example chart for platform metrics

In order to better illustrate what happens inside the platform during the execution of some SPARQL Queries we present in Figure 3.5 a line chart with two important metrics: the allocated memory vs used memory. First one may notice that on the

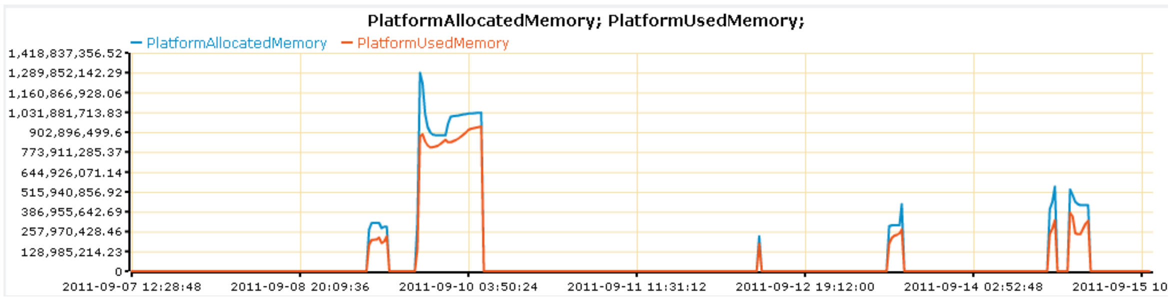


Figure 3.5: Platform Allocated Memory vs. Platform Used Memory

graphic we display a time interval of several days (2011-09-07 until 2011-09-15). The high peaks of the chart underline the time intervals in which the platform received queries and workflows to process. The two metrics, platform allocated memory and platform used memory are measured with respect to the virtual machine. We may notice that the allocated memory is always larger than the used memory.

3.3.4 Example charts for workflow metrics

The bar chart in figure 3.6 represents the total number of workflows that were run on every LarKC platform instrumented. Three different instances of the platform have been run on windows XP - 32 bit machine and on windows-7 64-bit machine. We count how many different workflow instances have been submitted to the platform not how many different workflows (as we mainly had five worklows that were run GWAS, Urban LarKC – with its three sub-workflows and LLD).

Figure 3.7 illustrates the total number of workflow specific metrics recorded for each of the five different workflows that were run on the platform. We may notice that most of the metrics have been collected for the plugin GWASIdentifier – as it was run most of the times (we had a large number of queries for it) and it is followed by SOStoVBTransformer – which was implied in most of the workflows.

The number of query metrics for each workflow are represented as a pie chart in figure 3.8. For each workflow in one of the platforms' run we may depict the number of query metrics that have been recorded for it. We may notice in figure 3.8 that most of the query metrics have been generated by the GWASIdentifier plugin, followed by SOStoVBTransformer.

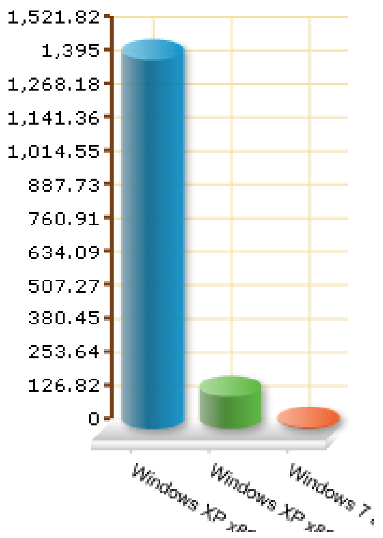


Figure 3.6: Number of workflows that were run on each platform

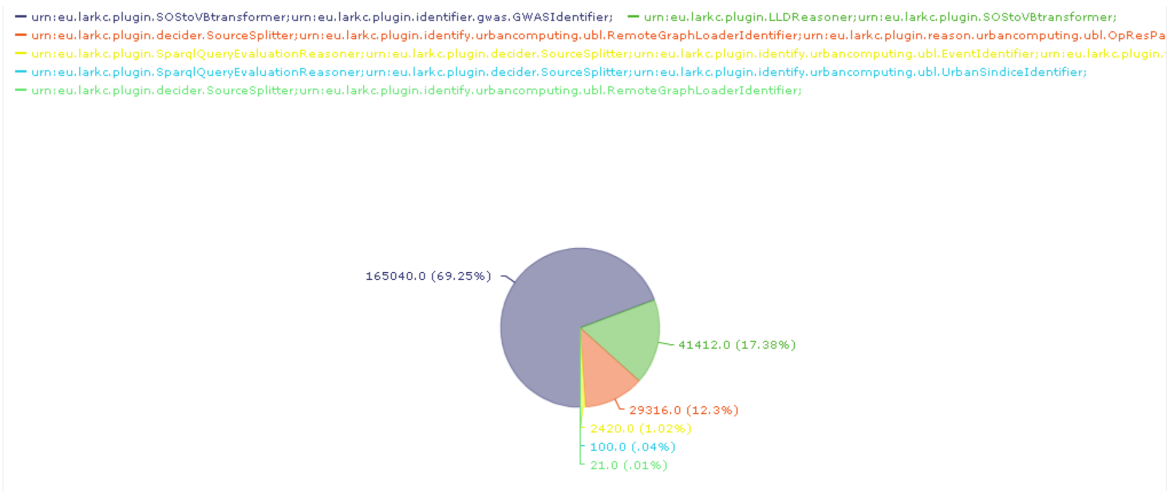


Figure 3.7: Number of workflow metrics recored for each workflow

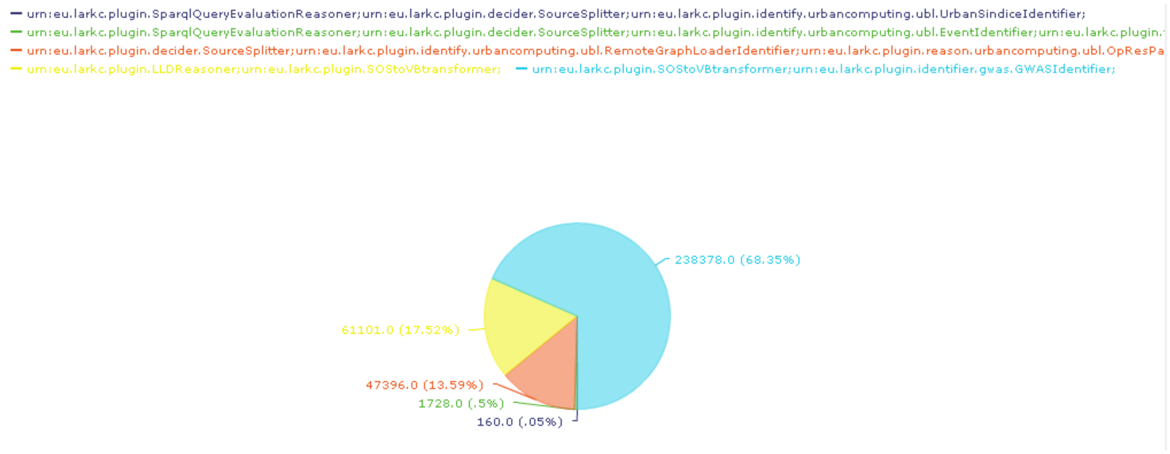


Figure 3.8: Number of query metrics for each workflow

3.3.5 Example charts for plug-in metrics

We have instrumented 15 LarKC plugins during our tests. The number of plugin metrics recorded for each plugin are presented in Figure 3.9. We may notice that

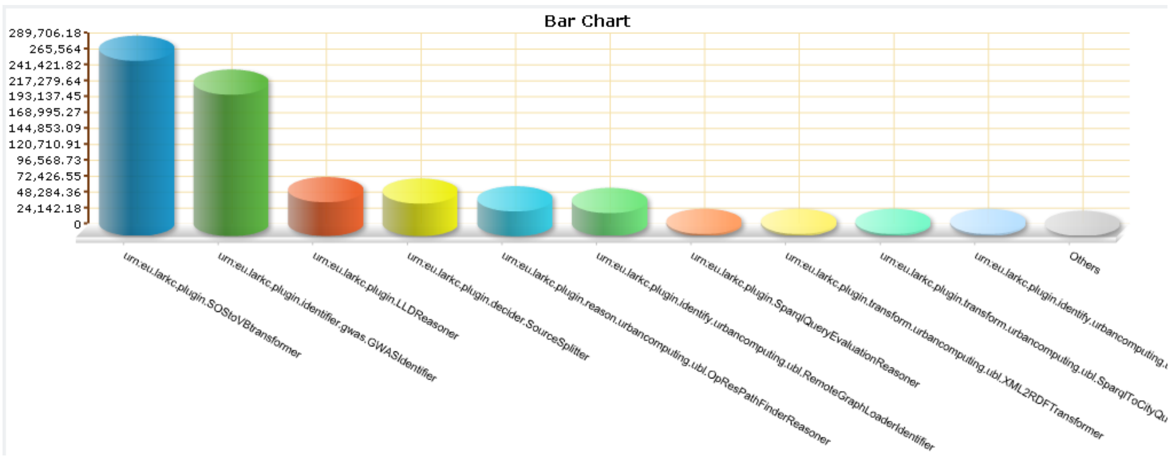


Figure 3.9: Number of plugin metrics for each plugin name

most of the plugin metrics are recorded for SOStoVBTransformer which is followed by GWASIdentifier and LLDReasoner.

The plugin average total response time is represented by the bar chart in figure 3.10 and the plugin average thread count is shown in figure 3.11 . The total response time

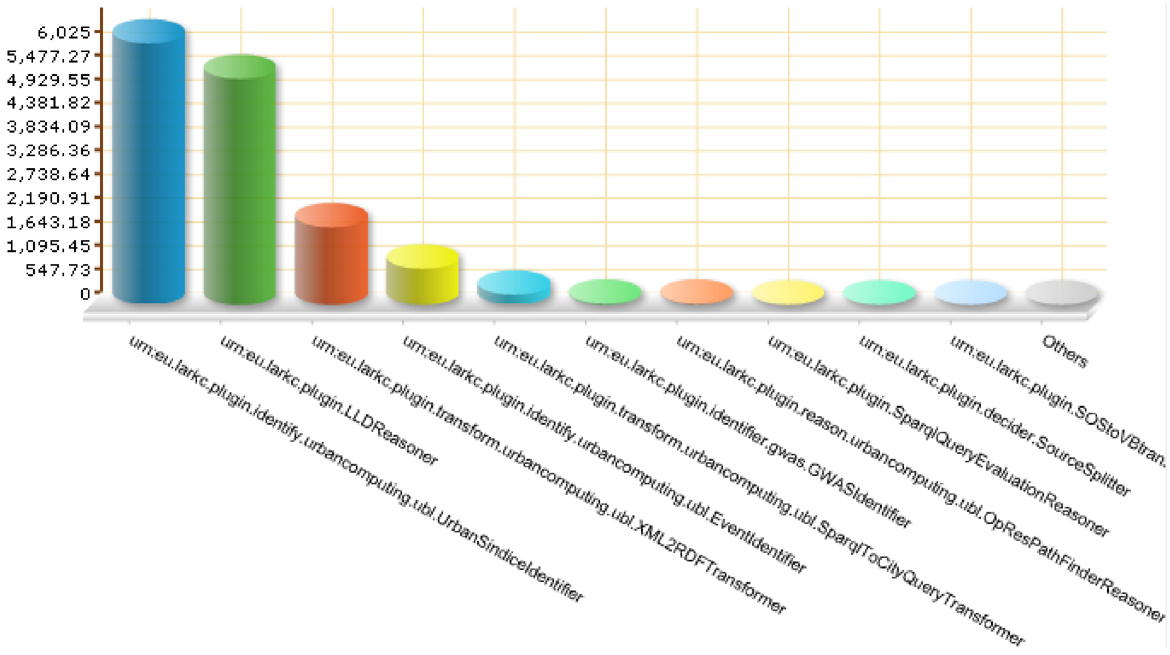


Figure 3.10: Plugin Average Total Response Time

of a plug-in models the time interval between the plug-in startup and its execution end for a given run of a workflow. It is interesting to note that the plug-in having the largest execution time is UrbanSindiceIdentifier followed by the LLDReasoner, while the smallest execution time is recorded for SOSToVBTransformer.

The Plugin Average Thread Count metric shows an average of the number of threads created by each plug-in. We count how many threads were created at each

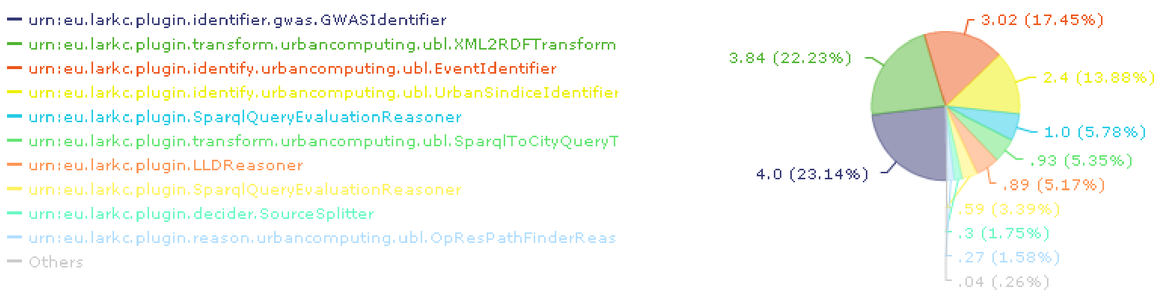


Figure 3.11: Plugin Average Thread Count

plug-in invocation and we divide the sum with the number of plug-in invocations. From figure 3.11 we may conclude that the plug-in GWASIdentifier used an average of about 4 threads followed by XML2RDFTransformer and by the EventIdentifier plugin.

3.3.6 Example charts for query metrics

Another example bar chart is presented in figure 3.12 that shows the average query thread total CPU time – a measurement that refers to total CPU time spent by current thread executing the query invocation method (given by the sum between the user time and the system time). The graphic in figure 3.12 displays the query identifiers on the

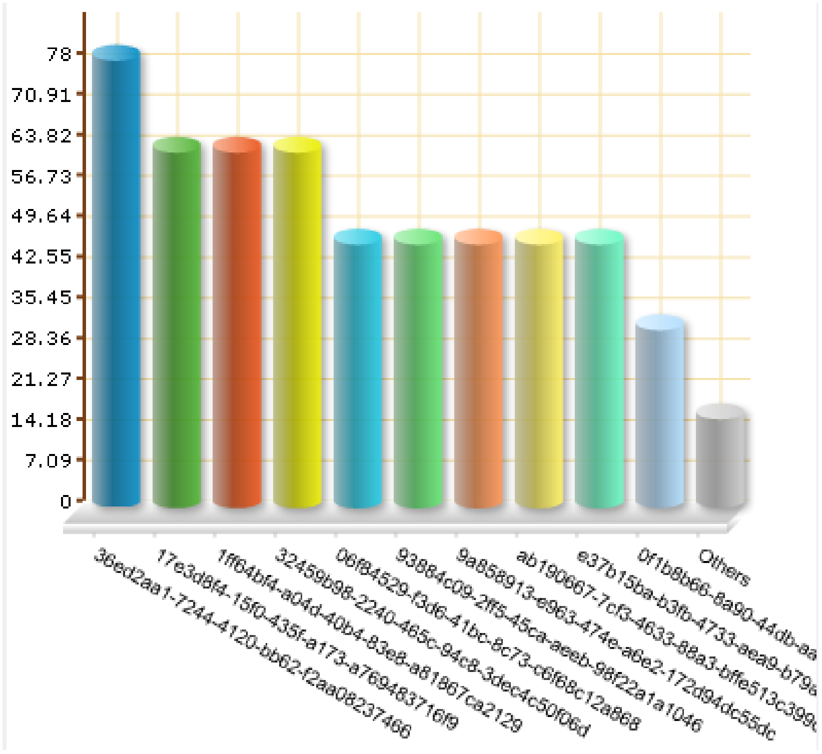
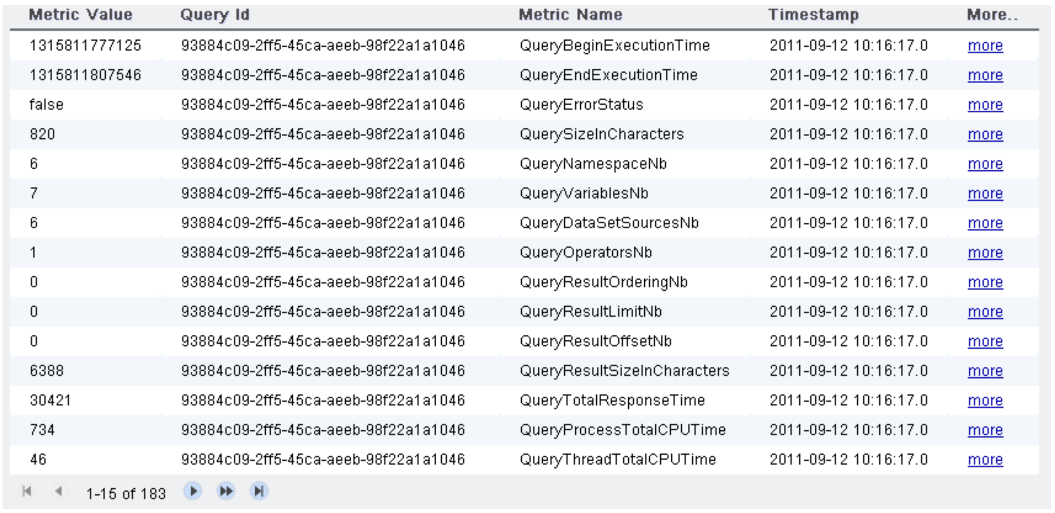


Figure 3.12: Average Query Thread Total CPU Time

x axis and on y the average query thread total CPU time. In this chart we have displayed the first 10 queries having the largest average query thread total CPU time. For example, the first query, having as instrumentation identifier 36ed2aa1-7244-4120-bb62-f2aa08237466 is part of the workflow comprising the GWASIdentifier plugin and it has the largest average query thread total CPU time followed by the query identified by 17e3d8f4-15f0-435f-a173-a769483716f9 that is part of the workflow containing the LLDReasoner plugin.

Figure 3.13 presents some query metrics from the Monument Finder workflow. It shows an example of the tabular view offered by the visualization component.



Metric Value	Query Id	Metric Name	Timestamp	More..
1315811777125	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryBeginExecutionTime	2011-09-12 10:16:17.0	more
1315811807546	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryEndExecutionTime	2011-09-12 10:16:17.0	more
false	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryErrorStatus	2011-09-12 10:16:17.0	more
820	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QuerySizeInCharacters	2011-09-12 10:16:17.0	more
6	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryNamespaceNb	2011-09-12 10:16:17.0	more
7	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryVariablesNb	2011-09-12 10:16:17.0	more
6	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryDataSetSourcesNb	2011-09-12 10:16:17.0	more
1	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryOperatorsNb	2011-09-12 10:16:17.0	more
0	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryResultOrderingNb	2011-09-12 10:16:17.0	more
0	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryResultLimitNb	2011-09-12 10:16:17.0	more
0	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryResultOffsetNb	2011-09-12 10:16:17.0	more
6388	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryResultSizeInCharacters	2011-09-12 10:16:17.0	more
30421	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryTotalResponseTime	2011-09-12 10:16:17.0	more
734	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryProcessTotalCPUTime	2011-09-12 10:16:17.0	more
46	93884c09-2ff5-45ca-aeeb-98f22a1a1046	QueryThreadTotalCPUTime	2011-09-12 10:16:17.0	more

1-15 of 183

Figure 3.13: Query metrics example for the Monument Finder Workflow

3.3.7 Other visualization facilities

The visualization module is highly customizable and it allows other views. For example, interested user may visualize metrics related to a single plug-in, to a single workflow, to a single platform instance or to a single query, or they may perform comparisons between several metrics. Due to lack of space we will exemplify shortly the charts that one may create for the worfklow that involves the GWASIdentifier plugin. Figure 3.14 displays example charts for some of the plug-in specific metrics.

Other charts can be displayed for workflow specific metrics. Figure 3.15 shows some of these charts for the workflow involving the GWASIdendifier plugin.

Figure 3.16 presents charts related to the behavior of the platform during the execution of the workflow and queries for the GWASIdetifier plugin. Just for exemplification we have chosen few charts to display.

The same types of visualization charts can be displayed for any plug-in and for any workflow and for any platform instance.

3.3.8 Graph based visualization

A complementary tool for visualization of monitoring data developed in WP11, is the graph based visualization tool. It uses the underlying representation of monitoring data in RDF and displays graphically the underlying ontological structure and the time series charts. Some of the charts are available in Figure 3.17 and Figure 3.18.

Figure 3.17 shows a preview of the CPU load and IO read, write. Moving the mouse over the graphic will result in the display of the actual values.

By zooming into a graphic, values are displayed using bar charts as shown in 3.18

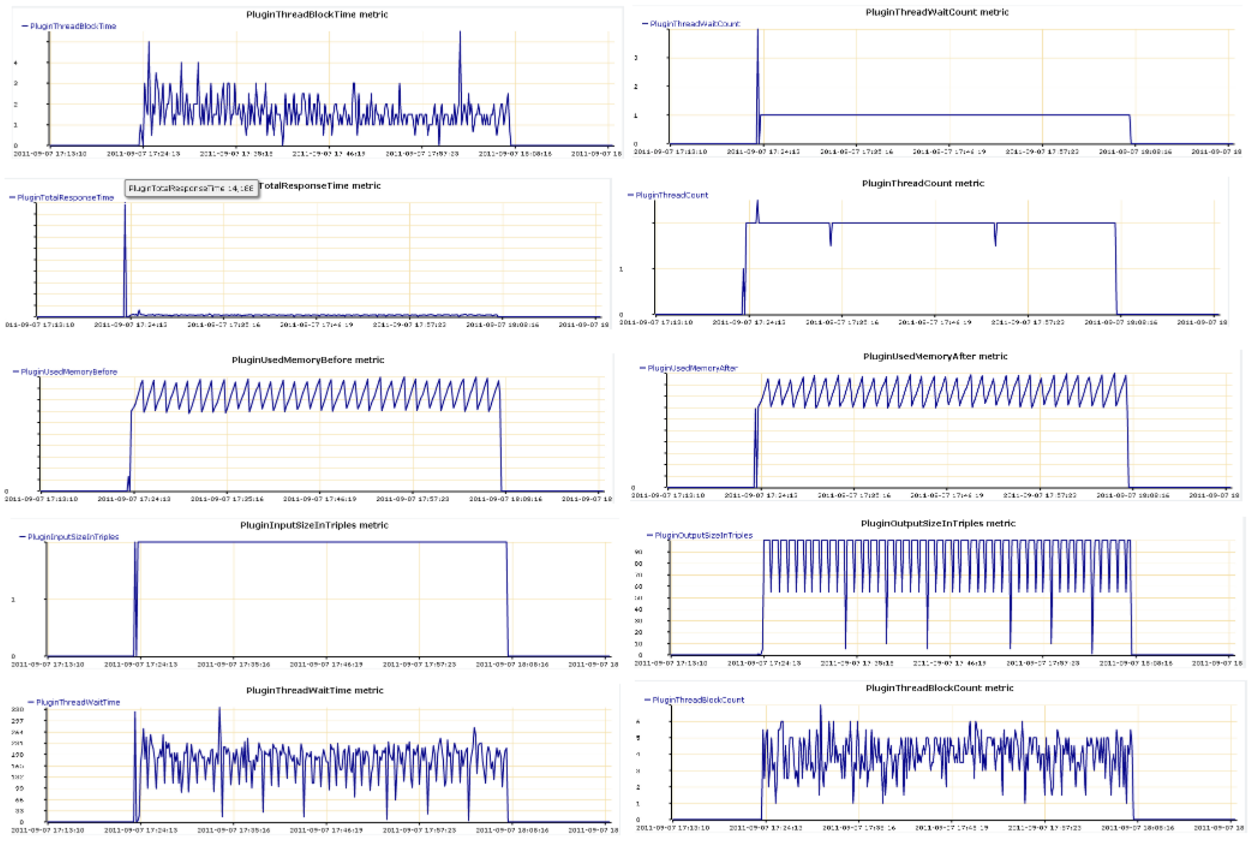


Figure 3.14: GWASIdentifier - plugin metrics

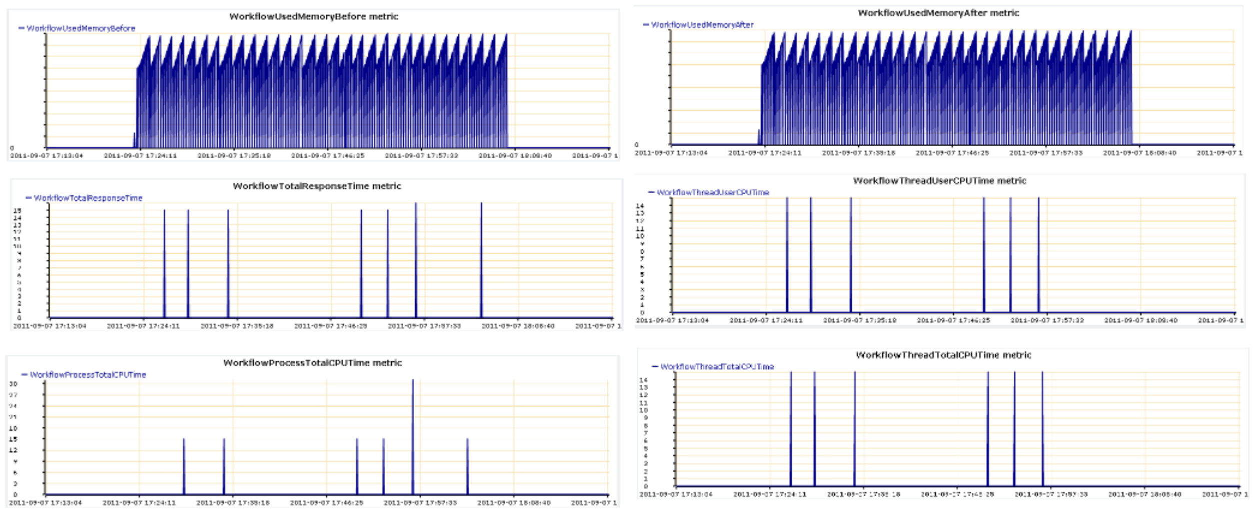


Figure 3.15: GWAS workflow metrics

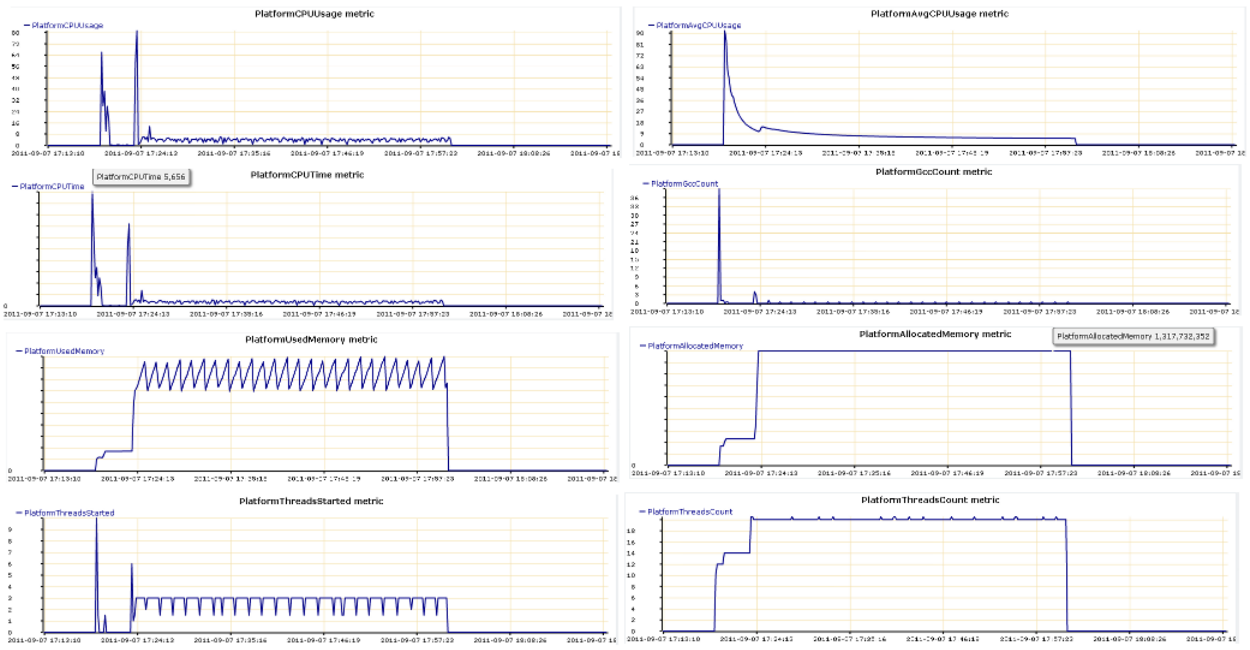


Figure 3.16: Plataform metrics collected only for GWAS workflow

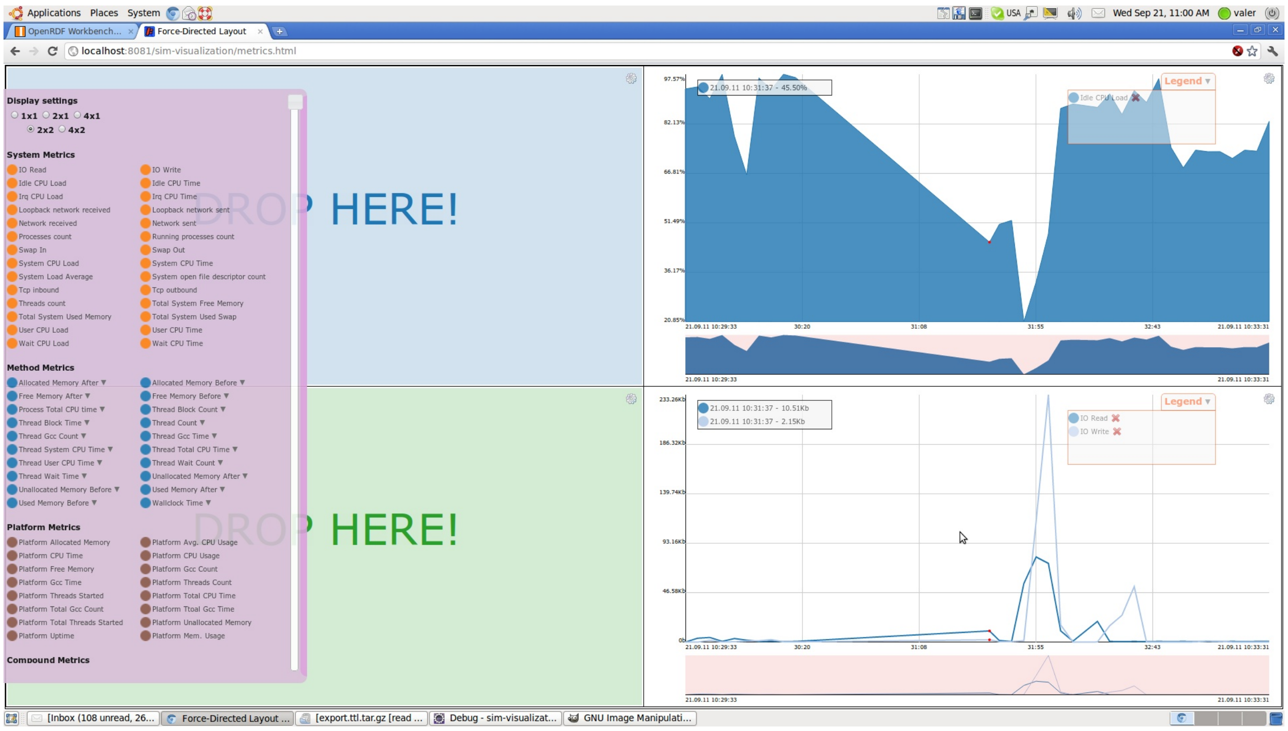


Figure 3.17: Graph base visualization - preview

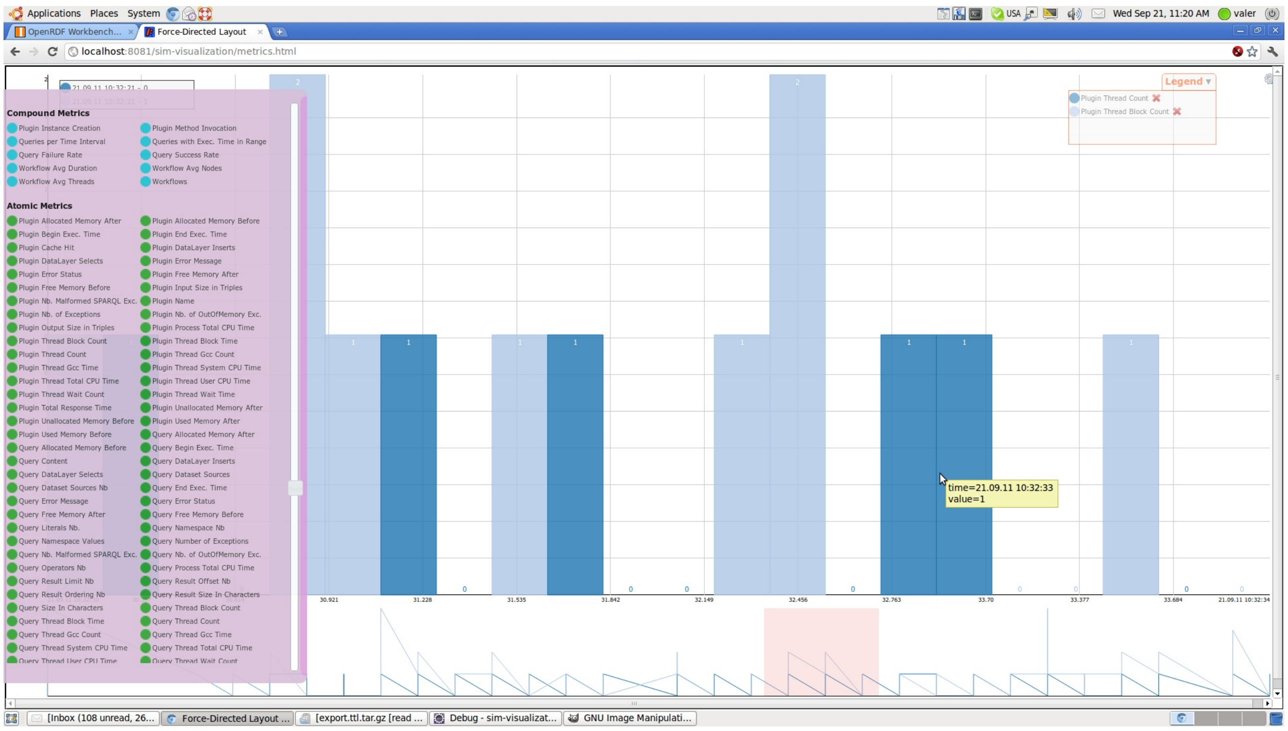


Figure 3.18: Graph base visualization - bar charts

3.4 Relevance feedback applied on instrumentation data

3.4.1 Goals

The goal of the experiments was to evaluate two numerical prediction methods used for predicting the performance parameters of SPARQL queries (eg. workflow/query execution time, number of threads, etc.). The indicators used in evaluating the prediction methods are the accuracy of the predicted results and the time needed to predict the output parameters of one query.

3.4.2 Test case and methodology

The numerical prediction methods are clustering followed by regression, and kernel canonical correlation analysis. Both were trained and tested on the same datasets. The training was performed on 3455 instances and a testing dataset was created for another 100 instances. Kernel canonical correlation analysis is a statistical machine learning method which projects the criterion and predictor set of variables onto new dimensions where the correlation between them is maximal (where the predictor set best explains the criterion set). A major advantage of this method is that it is able to predict multiple output values at a time. The first step in the clustering and regression approach is to perform clustering on the training data considering different input attributes that characterize the queries. Next, a simple linear regression model is computed for each of the clusters obtained in the previous step. In the prediction step, a new query is put in a best-fitting cluster and the corresponding regression model is applied to it in order to obtain the output performance measures. The workflows considered contained either the SOStoVBtransformer and GWASidentifier plugins, or the LLDReasoner and SOStoVBtransformer plugins. The accuracy of the methods is tested using the R-squared metric, computed with the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{\sum_{i=1}^{n}(actual_i - actual_{mean})^2}, \text{n} = \text{number of test instances}$$

A value close to 1 indicates almost perfect prediction. Negative values are also possible for the case in which the training and test datasets are disjoint. The metric is very sensitive to outliers.

Another accuracy evaluation metric which was considered in our experiments is Mean Absolute Percentage Error. Its formula is the following:

$$M = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{actual_i - predicted_i}{actual_i}\right|, \text{n} = \text{number of test instances}$$

It expresses accuracy as a percentage. The prediction model best fits the data when the values of this metric are close to 0.

3.4.3 Environment

The automatic test was performed using a tool written in Java. The tool reads an ARFF file which contains the test instances, predicts the output parameters for each instance and writes the results to a file and display graph charts (see Figure 2.1).

3.4.4 Results

The following charts show the actual and predicted values for some query output performance metrics. The x axis represents the test queries and the y axis represents either a time expressed in milliseconds or a discrete value. We also present the accuracy indicators values for each chart.

The prediction time elapsed for a single new query is on average 3 seconds using KCCA and 0.5 seconds using clustering and regression technique.

Query total response time prediction

The evaluation for the predicted *QueryTotalResponseTime* metric values was done by both two methods: clustering and regression (see Figure 3.19) and KCCA (see Figure 3.20). The accuracy values in terms of R-squared metric (RSM) and mean absolute percentage error (MAPE) are depicted in table 3.2.

*	RSM	MAPE
Clustering-Regression	0.30	0.60
KCCA	-0.15	0.44

Table 3.2: Accuracy for QueryTotalResponseTime prediction

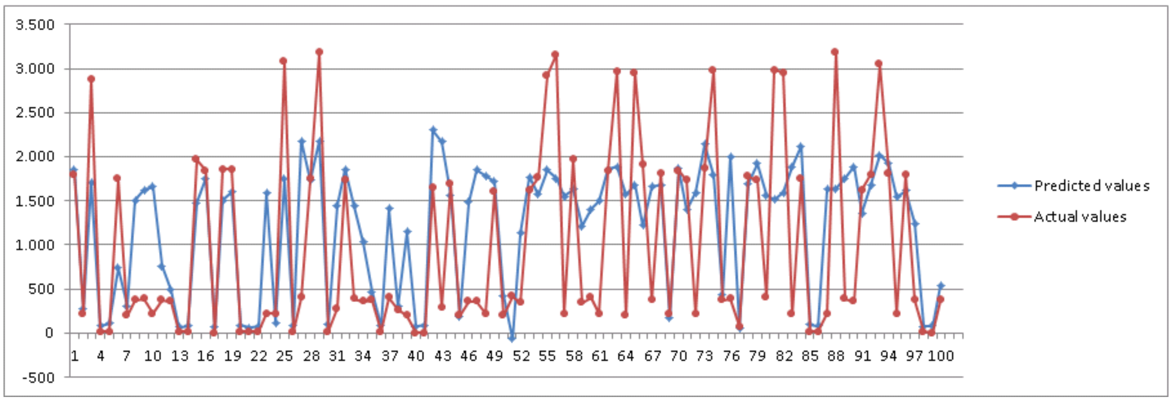


Figure 3.19: Query total response time prediction using clustering and regression method

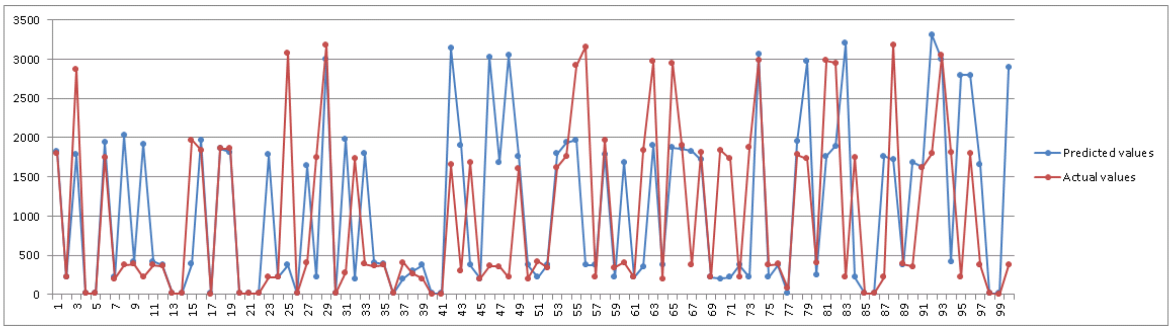


Figure 3.20: Query total response time prediction using KCCA method

Query process total CPU time prediction

The evaluation for the predicted *QProcessTotalCPUTime* metric values was done by both two methods: clustering and regression (see Figure 3.21) and KCCA (see Figure 3.22). The accuracy values in terms of R-squared metric (RSM) and mean absolute percentage error (MAPE) are depicted in table 3.3.

*	RSM	MAPE
Clustering-Regression	0.38	0.50
KCCA	-0.07	0.52

Table 3.3: Accuracy for QueryProcessTotalCPUTime prediction

Query thread total CPU time prediction

The evaluation for the predicted *QThreadTotalCPUTime* metric values was done by both two methods: clustering and regression (see Figure 3.23) and KCCA (see Figure 3.24). The accuracy values in terms of R-squared metric (RSM) and mean absolute percentage error (MAPE) are depicted in table 3.4.

Query thread count prediction

The evaluation for the predicted *QueryThreadCount* metric values was done by both two methods: clustering and regression (see Figure 3.25) and KCCA (see Figure **??**).

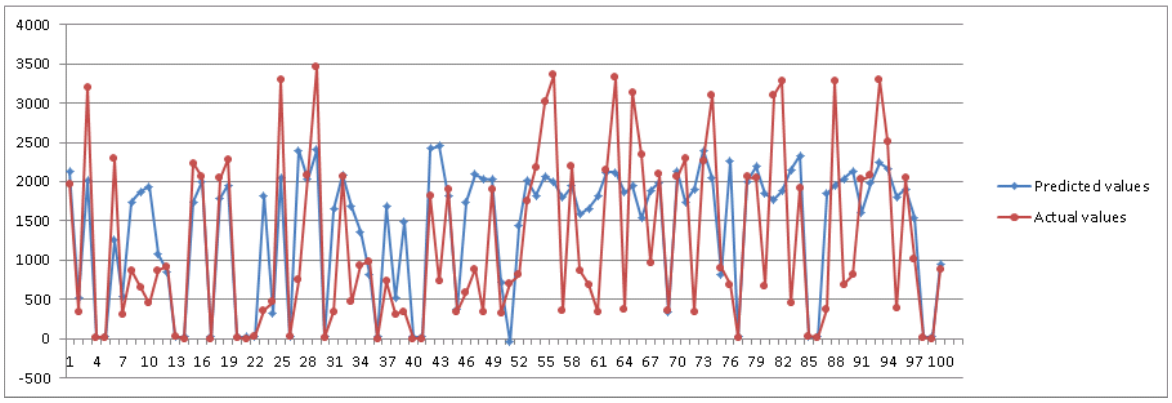


Figure 3.21: Query process total CPU time prediction using clustering and regression method

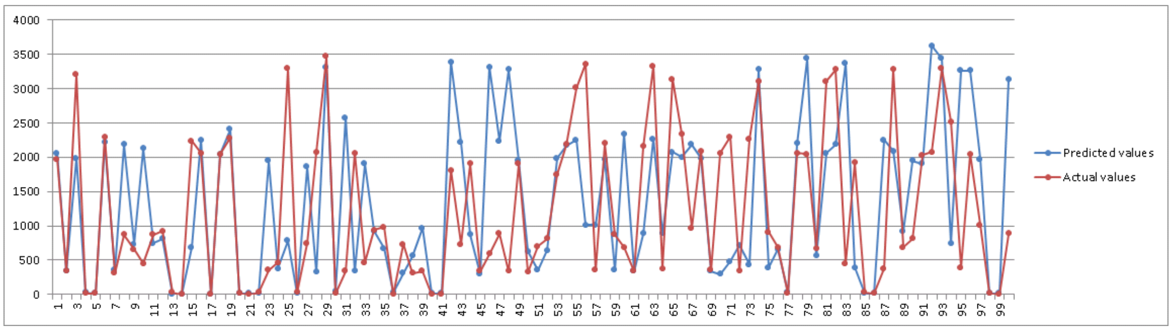


Figure 3.22: Query process total CPU time prediction using KCCA method

*	RSM	MAPE
Clustering-Regression	0.47	0.44
KCCA	-0.08	0.45

Table 3.4: Accuracy for QueryThreadTotalCPUTime prediction

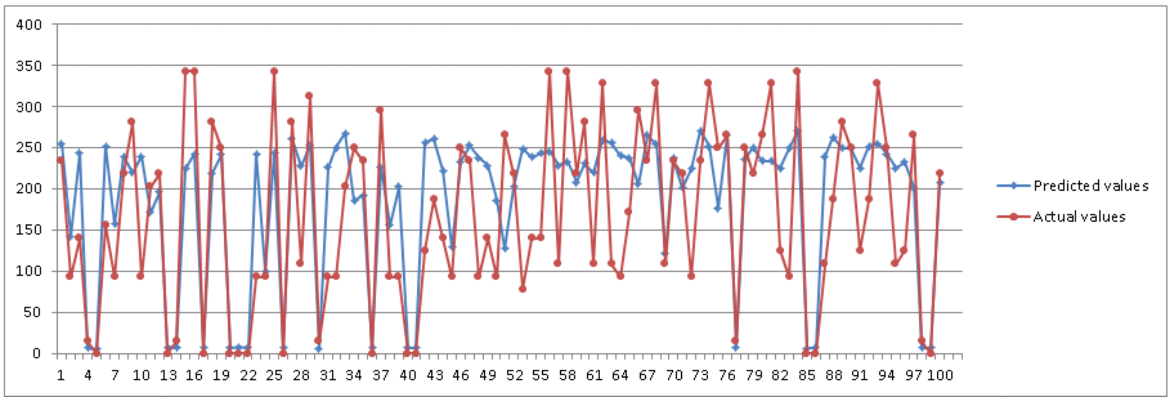


Figure 3.23: Query thread total CPU time prediction using clustering and regression method

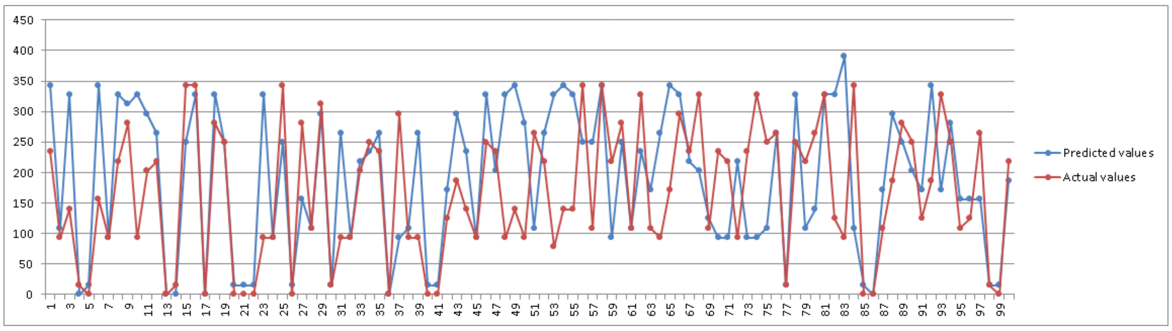


Figure 3.24: Query thread total CPU time prediction using KCCA method

The accuracy values in terms of R-squared metric (RSM) and mean absolute percentage error (MAPE) are depicted in table 3.5.

*	RSM	MAPE
Clustering-Regression	0.31	0.19
KCCA	-0.07	0.22

Table 3.5: Accuracy for QueryThreadCount prediction

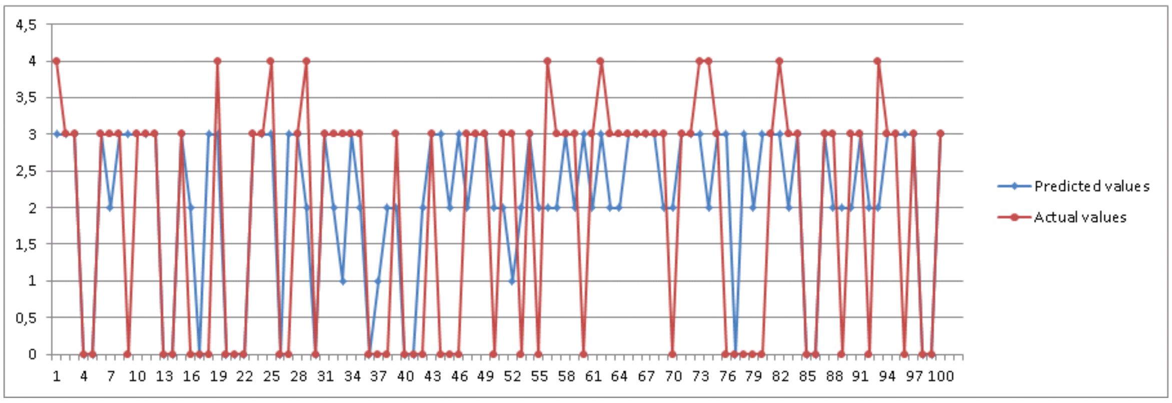


Figure 3.25: Query thread count prediction using clustering and regression method

Query result size in characters prediction

The evaluation for the predicted *QueryResultSizeInCharacters* metric values was done by both two methods: clustering and regression (see Figure 3.27) and KCCA (see Figure 3.28). The accuracy values in terms of R-squared metric (RSM) and mean absolute percentage error (MAPE) are depicted in table 3.6.

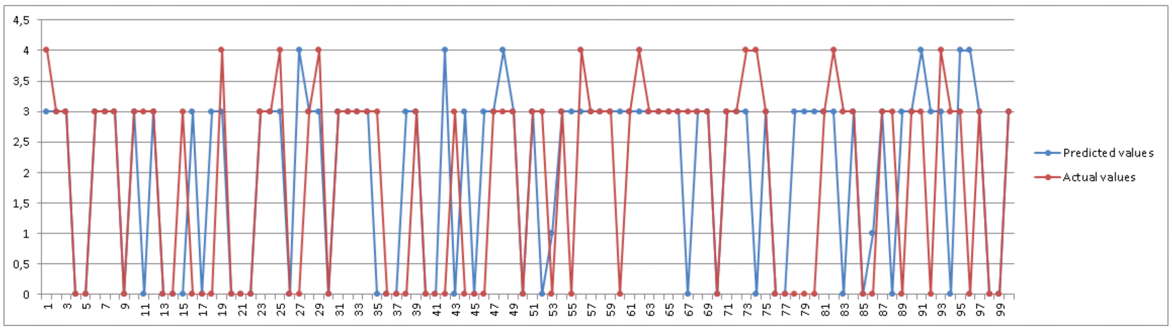


Figure 3.26: Query thread count prediction using KCCA method

*	RSM	MAPE
Clustering-Regression	0.99	0.12
KCCA	0.99	0.12

Table 3.6: Accuracy for QueryResultSizeInCharacters prediction

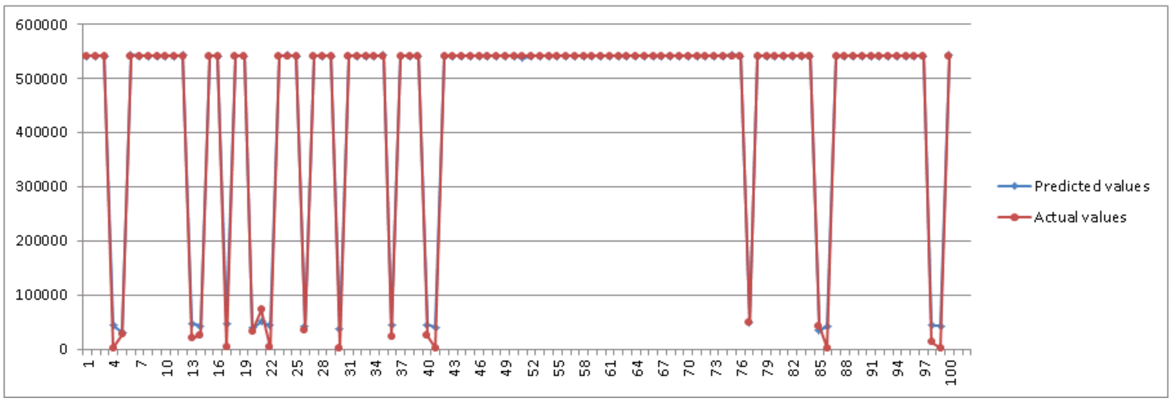


Figure 3.27: Query result size in characters prediction using clustering and regression method

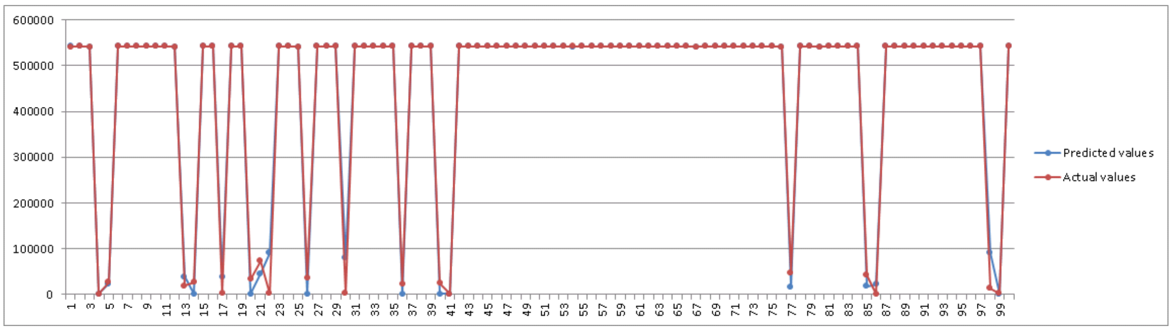


Figure 3.28: Query result size in characters prediction using KCCA method

4. Conclusion

This deliverable made a detailed evaluation of the instrumentation and monitoring tools introduced by WP11.

We have evaluated each of our major components, namely the instrumentation component, visualization and relevance feedback component using a set of criteria like: rigidity, robustness, data storage capacity, accuracy.

We revisit the initial requirements defined in deliverable D11.1.1 [1] and explained the degree to which these requirements have been satisfied.

An indirect evaluation of WP11 tools was performed by evaluating the LarKC platform, the workflows and plug-ins available in the use-cases. We performed a large scale evaluation in order to stress the platform as much as possible. This evaluation resulted in the generation of a large knowledge base of instrumentation metrics. For generating this data we have used both real and synthetic queries from LarKC use-cases.

The experiments we have made show that the tools are highly flexible, that is they can cope with new instrumentation metrics, are robust and scalable.

A. Urban Event

A.1 Workflow description

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
@prefix larkc:<http://larkc.eu/schema#>  .
@prefix uc:<http://larkc.cefriel.it/ontologies/urbancomputing#>  .
_:sourceSplitter a<urn:eu.larkc.plugin.decider.SourceSplitter>  .
_:sourceSplitter larkc:connectsTo _:cityExtractor .
_:sourceSplitter larkc:connectsTo _:sparqlEvaluator .
_:cityExtractor a
<urn:eu.larkc.plugin.transform.urbancomputing.ubl.SparqlToCityQueryTransformer>
.
_:cityExtractor larkc:connectsTo _:eventIdentifier .
_:eventIdentifier a
<urn:eu.larkc.plugin.identify.urbancomputing.ubl.EventIdentifier>  .
_:eventIdentifier larkc:connectsTo _:xml2rdf .
_:eventIdentifier larkc:hasParameter _:eventIdentifierParams.
_:eventIdentifierParams uc:hasApiKey "Md2mzZP7Tk4GkPhw";
                          uc:hasXsltTransformation
"http://seip.cefriel.it/urbanlarkc-public/evdb-event2rdf.xsl".

_:xml2rdf a
<urn:eu.larkc.plugin.transform.urbancomputing.ubl.XML2RDFTransformer>  .
_:xml2rdf larkc:connectsTo _:sparqlEvaluator .

_:sparqlEvaluator a<urn:eu.larkc.plugin.SparqlQueryEvaluationReasoner>.
<urn:eu.larkc.endpoint.sparql.ep1>  a<urn:eu.larkc.endpoint.sparql>  .
<urn:eu.larkc.endpoint.sparql.ep1>  larkc:links _:path .
_:path a larkc:Path .
_:path larkc:hasInput _:sourceSplitter .
_:path larkc:hasOutput _:sparqlEvaluator .
```

A.2 Sample queries

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfcal:<http://www.w3.org/2002/12/cal/icaltzd#>
PREFIX addr:<http://schemas.talis.com/2005/address/schema#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
SELECT ?e ?s ?summary ?cat ?desc ?l ?lab ?lat ?lng
WHERE{
?e rdf:type rdfcal:Vevent.
?e rdfcal:summary ?summary.
?e skos:subject ?cat.
?e rdfcal:description ?desc.
?e geo:location ?l.
?l rdfs:label ?lab.
?l geo:lat ?lat.
?l geo:long ?lng.
?e rdfcal:dtstart ?s .
?l addr:localityName "Milan".
FILTER(?s>  xsd:dateTime("2011-03-30T00:00:00Z")&&  ?s<
xsd:dateTime("2011-04-04T23:59:59Z")).
}
```

B. Monument Finder

B.1 Workflow description

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
@prefix larkc:<http://larkc.eu/schema#>  .
@prefix uc:<http://larkc.cefriel.it/ontologies/urbancomputing#>  .
_:sourceSplitter a<urn:eu.larkc.plugin.decider.SourceSplitter>  .
_:sourceSplitter larkc:connectsTo _:monumentLocator .
_:sourceSplitter larkc:connectsTo _:sparqlEvaluator .
_:monumentLocator a
<urn:eu.larkc.plugin.identify.urbancomputing.ubl.UrbanSindiceIdentifier>  .
_:monumentLocator larkc:connectsTo _:sparqlEvaluator .
_:sparqlEvaluator a<urn:eu.larkc.plugin.SparqlQueryEvaluationReasoner>  .
<urn:eu.larkc.endpoint.sparql.ep1>  a<urn:eu.larkc.endpoint.sparql>  .
<urn:eu.larkc.endpoint.sparql.ep1>  larkc:links _:path .
_:path a larkc:Path .
_:path larkc:hasInput _:sourceSplitter .
_:path larkc:hasOutput _:sparqlEvaluator .
```

B.2 Sample queries

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX georss:<http://www.georss.org/georss/>
PREFIX dcterms:<http://purl.org/dc/terms/>
SELECT DISTINCT ?monument ?geopoint ?img ?wiki ?name ?desc
WHERE{
{
{?monument dcterms:subject ?subject.
?subject skos:broader
<http://dbpedia.org/resource/Category:Visitor_attractions_in_Milan>.}
UNION
{?monument dcterms:subject
<http://dbpedia.org/resource/Category:Visitor_attractions_in_Milan>.}}
?monument georss:point ?geopoint.
?monument foaf:depiction ?img.
?monument foaf:page ?wiki.
?monument rdfs:label ?name.
?monument rdfs:comment ?desc.
FILTER ( lang(?name) = "en"&&  lang(?desc) = "en" )
}
```

C. Path Finder

C.1 Workflow description

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix larkc: <http://larkc.eu/schema#> .
@prefix uc: <http://larkc.cefriel.it/ontologies/urbancomputing#> .

_:sourceSplitter a <urn:eu.larkc.plugin.decider.SourceSplitter> .
_:sourceSplitter larkc:connectsTo _:remoteLoader .
_:sourceSplitter larkc:connectsTo _:pathFinder .

_:remoteLoader a <urn:eu.larkc.plugin.identify.urbancomputing.ubl.RemoteGraphLoaderIdentifier> .
_:remoteLoader larkc:connectsTo _:pathFinder .
_:remoteLoader  larkc:hasParameter _:remoteLoaderParams.

_:remoteLoaderParams uc:location
"file:///d:\\UrbanPathFinding\\RemoteGraphLoaderIdentifier\\lib\\ama-xml-milano_navigli_graph.rdf";

uc:graphName "http://seip.cefriel.it/ama".

_:pathFinder a <urn:eu.larkc.plugin.reason.urbancomputing.ubl.OpResPathFinderReasoner> .

<urn:eu.larkc.endpoint.sparql.ep1> a <urn:eu.larkc.endpoint.sparql> .
<urn:eu.larkc.endpoint.sparql.ep1> larkc:links _:path .

_:path a larkc:Path .
_:path larkc:hasInput _:sourceSplitter .
_:path larkc:hasOutput _:pathFinder .
```

C.2 Sample queries

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns/>
PREFIX lud: <http://www.linkingurbandata.org/onto/ama/>
SELECT ?p ?w ?n1 ?l ?n2  WHERE {
?p rdf:type lud:Paths.
?p lud:pathFrom <http://seip.cefriel.it/ama/resource/nodes/node8252>.
?p lud:pathTo <http://seip.cefriel.it/ama/resource/nodes/node8253>.
?p lud:contain ?l.
?l lud:from ?n1.
?l lud:to ?n2.
?p lud:pathWeight ?w.
}
ORDER BY ?w
```

D. GWAS

D.1 Workflow description

```
# Workflow Description
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix larkc: <http://larkc.eu/schema#> .
# Define the plug-ins
_:plugin1 a <urn:eu.larkc.plugin.identifier.gwas.GWASIdentifier> .
_:plugin2 a <urn:eu.larkc.plugin.SOStoVBtransformer> .
# Connect the plug-ins
_:plugin1 larkc:connectsTo _:plugin2 .
# Define a path to set the input and output of the workflow
_:path a larkc:Path .
_:path larkc:hasInput _:plugin1 .
_:path larkc:hasInput _:plugin2 .
_:path larkc:hasOutput _:plugin2 .

# Connect an endpoint to the path
<urn:eu.larkc.endpoint.sparql.ep1> a <urn:eu.larkc.endpoint.sparql> .
<urn:eu.larkc.endpoint.sparql.ep1> larkc:links _:path .
```

D.2 Sample queries

```
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
   PREFIX gwas:<http://www.gate.ac.uk/gwas#>
   PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   SELECT * WHERE {
    gwas:x rdf:type gwas:Experiment .
    gwas:x gwas:hasName "experiment1" .
    gwas:x gwas:hasKeywordGroup gwas:g1 .
    gwas:g1 gwas:hasKeyword "lung" .
    gwas:g1 gwas:hasKeyword "cancer" .
    gwas:x gwas:searchInRif "false" .
    gwas:x gwas:useUMLS "false" .
    gwas:x gwas:searchMode "1" .
    gwas:x gwas:dateConstraint "20110412" .
    gwas:x gwas:hasSnpId "rs1051730" .
    gwas:x gwas:hasSnpId "rs401681" .
    gwas:x gwas:hasSnpId "rs2736100"
   }
```

E. LLD

E.1 Workflow description

```
# Workflow Description
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix larkc: <http://larkc.eu/schema#> .

# Define three plug-ins
_:plugin1 a <urn:eu.larkc.plugin.LLDReasoner> .
_:plugin2 a <urn:eu.larkc.plugin.SOStoVBtransformer> .

# Connect the plug-ins
_:plugin1 larkc:connectsTo _:plugin2 .

# Define a path to set the input and output of the workflow
_:path a larkc:Path .
_:path larkc:hasInput _:plugin1 .
_:path larkc:hasInput _:plugin2 .
_:path larkc:hasOutput _:plugin2 .

# Connect an endpoint to the path
_:ep a <urn:eu.larkc.endpoint.sparql> .
_:ep larkc:links _:path .
```

E.2 Queries description

```
SELECT ?s ?p ?o WHERE { { ?s ?p ?o . ?s ?p "arthritis"} }
```

References

[1] R. Brehar, I. Toma, S. Nedevschi, M. Negru, S. Bota, I. Giosan, A. Vatavu, C. Vicas, and M. Chezan, "State of the art and requirements analysis," LarKC Project Deliverable, Tech. Rep. D11.1.1, 2010.

[2] I. Toma, R. Brehar, S. Bota, M. Chezan, I. Giosan, M. Negru, and A. Vatavu, "Instrumentation and monitoring platform - realization," LarKC Project Deliverable, Tech. Rep. D11.4, 2011.