ICT Seventh Framework Programme (ICT FP7)

Grant Agreement No: 318497

Data Intensive Techniques to Boost the Real – Time Performance of Global Agricultural Data Infrastructures

D5.3.2b Automatic Rigorous Testing Components

Deliverable Form	
Project Reference No.	ICT FP7 318497
Deliverable No.	D5.3.2b
Relevant Workpackage:	WP5: Semantic Infrastructure
Nature:	P
Dissemination Level:	PU
Document version:	Final
Date:	04/03/2016
Authors:	UAH, NCSR-D and SWC
Document description:	This report documents the implementation and deployment of the first version of the *Automated Rigorous Tester (ART)*. ART implements the methodologies developed in *WP4 Rigorous Experimental Testing* by collecting and analysing measurements from the operation of the *SemaGrow Stack*.

Document History

Version	Date	Author (Partner)	Remarks
Draft v0.8	16/06/2014	UAH, NCSR-D, UNITOV, SWC	Final version
Draft v0.9	23/06/2014	AK, FAO	Internal review
Final v1.0	30/06/2014	NCSR-D, SWC	Delivered as D5.3.1
Final v2.0	30/06/2015	NCSR-D, SWC	Delivered as D5.3.2
Draft v2.1	03/03/2016	NCSR-D	Newly added Section 3.2 that demonstrates the usage and expected output of the JMeter tool
Draft v2.2	04/03/2016	UAH	Internal review
Final v3.0	04/03/2016	NCSR-D, SWC	Delivered as D5.3.2b

The reviewers should note that this document only has minor updates wrt. D5.3.1 reviewed during the 2nd review. This deliverable is a prototype and work in Task 5.3 is reflected in the software. This deliverable was marked a Report by mistake in the original Part B of the proposal.

EXECUTIVE SUMMARY

This report documents the implementation and deployment of the first version of the *Automated Rigorous Tester (ART)*. ART implements the methodologies developed in *WP4 Rigorous Experimental Testing* by collecting and analysing measurements from the operation of the *SemaGrow Stack*. This incudes automatically measuring: (a) compression, reactivity, and throughput of the POWDER store that supports resource discovery with data summaries; and (b) the growth rate of the data summaries needed by the source selection algorithm and the efficiency and accuracy of the source selection algorithm that uses them, using the metrics defined in Section B1.1.2, Bullet 1, of the Technical Annex.

Besides measurements during the stack's regular operation of serving user queries over current SemaGrow data, ART is also used with test queries, automatically requested using user traffic projections and over current and realistic synthetic data. These test querying generates data for scalability analysis and for experimenting with different scenarios and data and usage growth projections.

Besides being a tool that facilitates the evaluation of SemaGrow technologies during the project, ART is also integrated in the *SemaGrow Stack* distribution as a tool for stack maintenance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF TERMS AND ABBREVIATIONS

Term/Abbreviation	*Definition*
Automatic Rigorous Tester (ART)	The *Automatic Rigorous Tester (ART)* is the component of the *SemaGrow Stack* that automatically benchmarks SemaGrow over current and projected data sets and usage loads.
cron job	Job scheduled for execution in *cron*, the time-based job scheduler.
JMeter	Open source, Java application designed to load test functional behaviour and measure performance. URL: http://jmeter.apache.org
Listener	*Listeners* store and/or visualize the output of performance tests.
Ontology Alignment or Ontology Matching	*Ontology alignment*, or *ontology matching*, is the process of determining correspondences between ontology entities. A set of correspondences is also called an *alignment*.
Resource Discovery	The component of the *SemaGrow Stack* that maintains and serves metadata about the contents of each data source that is federated under the *SemaGrow Stack*.
Sampler	*Samplers* tell JMeter about the requests that should be sent to the server that is being tested and the corresponding *listeners* can be used to store and/or visualize the output of the performance test.
Test Plan	A test plan describes a series of steps JMeter will execute when run. A complete test plan consists of, among other elements, *listeners*, *samplers*, and configuration elements.
Vocabulary Transformation	The component of the *SemaGrow Stack* that serves Ontology Alignment results.

1. INTRODUCTION

1.1 Purpose and Scope

This report documents the implementation and deployment of the second version of the *Automated Rigorous Tester (ART)*. ART implements the methodologies developed in *WP4 Rigorous Experimental Testing* by collecting and analysing measurements from the operation of the *SemaGrow Stack*. Besides measurements during the stack's regular operation, ART can also be used with generated data and user traffic scenarios in order to allow the infrastructure hosts to experiment with different projections of data and traffic growth.

Besides being a tool that will facilitate the evaluation of SemaGrow technologies during the project, ART is also integrated in the *SemaGrow Stack* distribution as a tool for stack maintenance.

1.2 Approach to Work Package and Relation to other Deliverables

The aim of WP5 is to carry out all development, integration, and deployment required in order to produce a complete and robust SemaGrow system based on the methodological and research prototyping work in WP3 and WP4; including both the *SemaGrow Stack* and the associated off-stack tools.
With respect to this deliverable in particular, this approach is refined as follows:

- Task 5.3 receives as input the automated testing methodology from Task 4.1, in order to implement the appropriate measurement collection functionality.
- Task 5.3 provides the *Automated Rigorous Tester (ART)* to the integrated prototype prepared in Task 5.4.

1.3 Methodology and Structure of the Deliverable

The rigorous testing component will automatically measure: (a) compression, reactivity, and throughput of the POWDER store that supports resource discovery with data summaries; and (b) the growth rate of the data summaries needed by the source selection algorithm and the efficiency and accuracy of the source selection algorithm that uses them, using the metrics defined in Section B1.1.2, Bullet 1.

We have analysed the testing requirements stemming from the Rigorous Testing methodology and have identified the Apache JMeter and Apache logging frameworks as appropriate basis. We will then implement the testing methods developed in WP4 [5]. Furthermore, test queries that have been produced through elicitation from domain experts can be added to those extracted from query logs, allowing administrators to estimate the impact of functionality extensions in the client applications.

1.4 Big Data Aspects

Most of the testing requirements can be addressed by on-line measurements that do not require for any data to be stored for processing, but there are also some requirements that could become challenging if the *response* to a query is so voluminous that it constitutes big data (cf. Requirements R6 and R10, Section 2.1). This issue is addressed by using a log rotation scheme and ensuring that a ceiling is enforced on the total space taken up by the logs. The processing of the log files for testing system status is carried out as soon as each log file in the rotation is closed, so that older files in the rotation can be deleted as soon as possible to make space (cf. Section 3.2).

2. MEASUREMENT REQUIREMENTS AND TESTER DESIGN

Rigorous testing in SemaGrow is supported by the *Automatic Rigorous Tester (ART)*, a component of the *SemaGrow Stack* that automatically benchmarks SemaGrow over current and projected data sets and usage loads. In this section we elicit requirements for the ART component from the SemaGrow success criteria and the experimental methodology, and then proceed to present an architecture that satisfies such requirements.

2.1 SemaGrow Success Criteria

In SemaGrow, we develop a *distributed querying infrastructure* that supports the interoperable and transparent application of data-intensive techniques over heterogeneous data sources. This deliverable assumes from Part B, Section 1.1.2 of the *Technical Annex* and from Section 3 of *D4.1: Scalability and Robustness Experimental Methodology* [6] the SemaGrow success criteria and the analysis of their relevance to the different components. From that, we derive here requirements on the information that ART should collect from the execution of the *SemaGrow Stack*:

- Requirements for the different *SemaGrow Stack* components on the metrics that they must provide over their internal processes and data structures. These requirements will be satisfied by work in the relevant tasks in WP3. It should be noted that these requirements do not pertain to the testing of each component (to be carried out within the respective WP3 task), but to information that each component must provide for the rigorous testing of the overall *SemaGrow Stack*.
- Requirements for the integrated system on the metrics that can only be collected from the integrated system and cannot be measured by testing any individual component alone.
- Requirements for the testing components themselves.

The remainder of Section 2.1 presents these requirements, and concludes by collecting and organizing them as requirements relevant to the design of ART and requirements for other deliverables.

2.1.1 Requirements related to Resource Discovery

The *Resource Discovery* component (*D3.1: Techniques for Resource Discovery*) provides advanced source selection methods over distributed databases based on efficient indexing of metadata that concisely describes the data sources federated under a SemaGrow Stack deployment; including instance-level metadata about the content of these data sources.

Resource Discovery is evaluated in terms of:

- The *size of the metadata* as a function of the *total size of the federated repositories*.
- The *time to retrieve* the instance-level metadata.
- The accuracy of the source selection in *predicting which sources hold data that satisfy a given query*, as found by exhaustively running user queries over all sources during source selection evaluation.
- The *overhead of the method* as a function of the *time it would take to query without any metadata available*.

The first two metrics can be fully measured within Task 3.1 and do not require any external information except for the overall size of the federated repositories, which is a static parameter for the duration of a given evaluation run. Furthermore, they are more relevant to the technical evaluation of the methods implemented in the component and is of little value to the administrators of *SemaGrow Stack* deployments.

On the other hand, the third and fourth metrics, and specifically the results and time to query all repositories by-passing resource discovery, cannot be measured outside the context of a full deployment. Furthermore, it is of value to the administrators of SemaGrow Stack deployments, as they can decide to refine the metadata maintained for resource discovery or to disable resource discovery altogether, depending on the natural and structure of the data federated by a given deployment of the SemaGrow Stack.

Given the above, we place on *Resource Discovery* have the requirement to record:

R1 The size of the metadata.

Furthermore, we place on ART the requirements to record:

R2 The time needed to query the metadata.

R3 The sources that were predicted to have relevant data but *did not*.

R4 The sources that were predicted to *not* have relevant data but did.

The time it would take to execute the user query without taking advantage of metadata is the same as R11, Section 2.1.3.

2.1.2 Requirements related to Query Transformation

The *Vocabulary Transformation* component (*D3.2: Techniques for Ontology Alignment*) serves alignment results in order to rewrite queries (or query fragments) from the query schema into the data source schema and results from the data source schema back into the query schema.

Vocabulary Transformation is evaluated in terms of:

- The time needed to apply a mapping
- The accuracy of the confidence level that the component reports

Given the above, we have the requirement to record:

R5 The time needed

R6 The results obtained under a transformation, to be compared against hand-crafted golden standards

2.1.3 Requirements related to Query Decomposition

The *Query Decomposition* component (*D3.4: Techniques for Heterogeneous Distributed Semantic Querying*) formulates the querying strategy over the federation, taking into account information served by Resource Discovery (*D3.1: Techniques for Resource Discovery*) and *Vocabulary Transformation* (*D3.2: Techniques for Ontology Alignment*) to allow queries in any schema to be executed at all and only those repositories that might hold relevant information, regardless of the schema these repositories use.

Query Decomposition is evaluated in terms of the optimality of the query decomposition along the following dimensions:

- Execution time: comparing the execution time of the selected decomposition with the execution time of other possible decompositions and the querying time of unoptimized baseline methods.
- Integrity of results: query decomposition is an optimization strategy and integrity of the results will be compared with those returned by unoptimized baseline methods.
- Vocabulary transformations: query rewriting should avoid low-confidence mappings and decompositions will be compared on the overall confidence of the results returned.

Given the above, we have the requirement to record:

R7 The time needed to compute a querying strategy

R8 The time needed to execute a querying strategy.

R9 The time spent waiting for query results from the federated data sources.

R10 The difference of the results obtained using a given strategy and the results obtained without using metadata to optimize query execution.

R11 The difference in time execution using a given strategy and without using metadata to optimize query execution.

R12 The overall confidence of a strategy (as calculated from the confidence reported by Query Transformation for the transformations needed to execute the strategy) as compared to maximum-confidence strategy identified by unoptimized baseline methods.

2.1.4 Requirements related to the Automated Rigorous Tester

The *Automated Rigorous Tester* must foresee the following modes of operation:

- **Normal optimized operation:** only recording what can be recorded from client queries. The testing framework has no control over the queries posed, and the testing overhead must be minimal.

- **Normal baseline operation:** bypassing Resource Discovery and querying without the benefit of data source metadata. The testing framework has no control over the queries posed, and the testing overhead must be minimal.
- **Test querying:** executing queries that have not been requested by any client but are useful for testing the system. Might be either optimized or baseline. The testing overhead may be significant.

Given the above, we have the requirement to record:

R13 The time needed to log measurements.

2.1.5 Overview

Table 1 provides an overview of how the requirements above are broken down into what information needs to be recorded, which components of the *SemaGrow Stack* architecture and interactions between them can provide this information, and the mode of operation of the *Automated Rigorous Tester* during which this information is available.

The *Point of Application* refers to the components shows in Figure 1.

Table 1: Requirements on the automated rigorous tester

Req.	Recorded Information	Point of Application	Mode of Operation
R1	Metadata size	Provided by Resource Discovery	–
R2	Invocation and return timestamps	Resource Discovery, as invoked by Query Decomposition	Any
R3 and R4	List of data sources predicted to have relevant data.		Any
	List of data sources (among those predicted) that actually had relevant data.	Query Manager, passing results back towards the SemaGrow endpoint	Any
	Complete list of data sources that have relevant data.		Blind
R5	Invocation and return timestamps	Vocabulary Transformation, as invoked by Query Execution Engine	Any
R6	Result set	Query Manager, passing results back towards the SemaGrow endpoint	Any
R7	Invocation timestamp	Query Decomposition, as invoked by the SemaGrow endpoint	Any
	Invocation timestamp	Query Manager, as invoked by Query Decomposition	Any
R8	Invocation and return timestamps	Query Manager as invoked by Query Decomposition and Query Execution Engines	Any
R9	Invocation and return timestamps	Query Execution Engines	Any
R10	Result Set	Query Manager, as invoked by Query Decomposition	Test Querying
R11	Invocation and return timestamps		
R12	Querying strategy and confidence reported by transformation	Vocabulary Transformation, as invoked by Query Execution Engine	Any
R13	Invocation and return timestamps	ART Logger	Normal

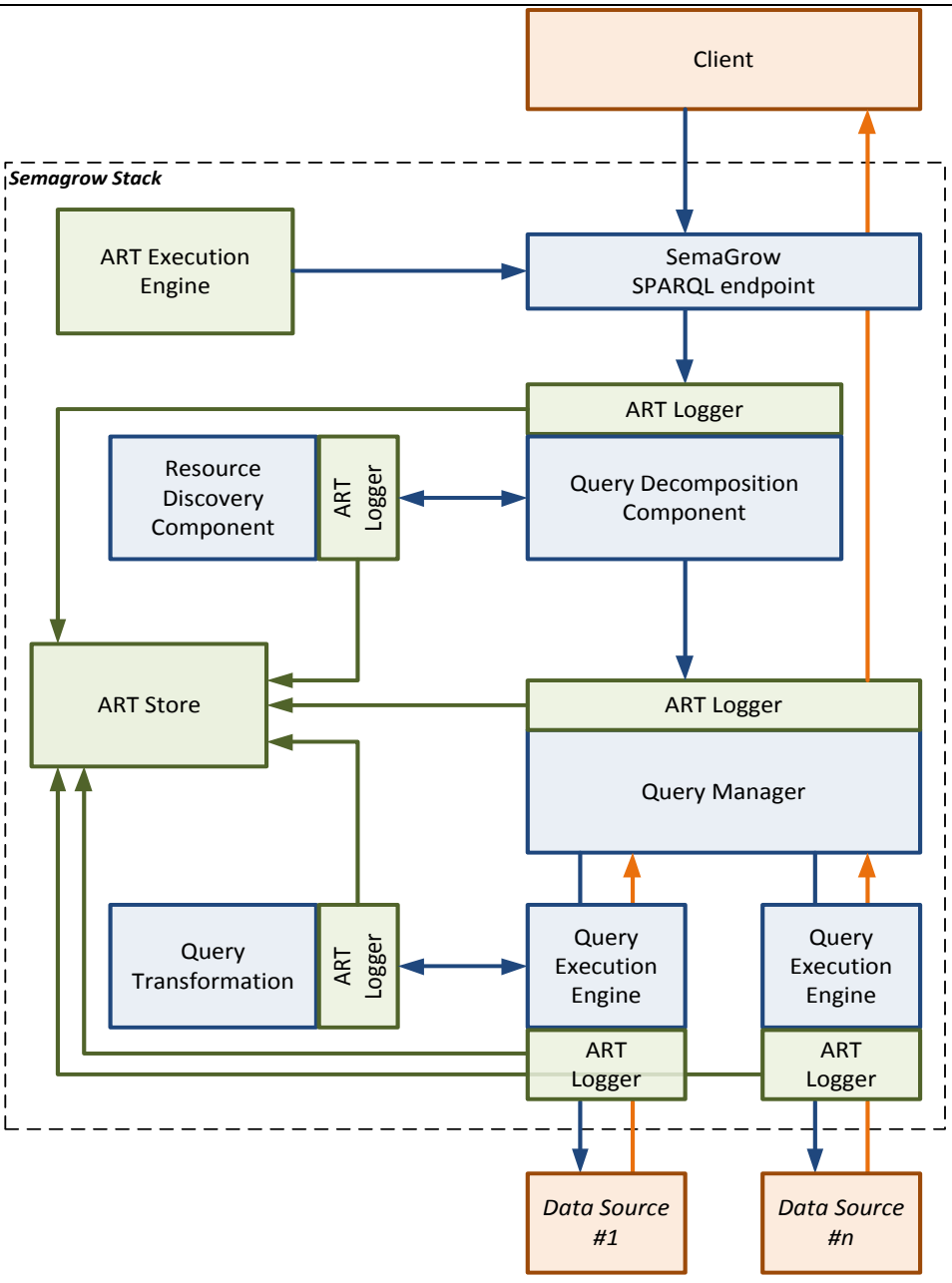

Figure 1: Automated Rigorous Testing components inside the overall architecture of the SemaGrow Stack

2.2 Automated Rigorous Tester Architecture

The Automated Rigorous Tester architecture (Figure 1) comprises the following components:

- The *execution engine* that is responsible for setting up and executing *test querying*, including populating repositories with realistic synthetic data and scaling up usage logs to perform size and usage scalability testing.
- The *loggers* that are responsible for logging timestamps and/or intercepting information passed between components of the *SemaGrow Stack*. The loggers persist this information in the store as efficiently as possible, with minimal overhead for the information passing they are logging.
- The *store* where information is deposited to be analysed without affecting the operation of any other component of the *SemaGrow Stack*.

3. IMPLEMENTATION

3.1 ART Execution Engine

The ART Execution Engine implementation is based on *performance testing tools* [1], which offer the capabilities we require from the Execution Engine, including:

- *Load testing* and *stress testing*, trying the system for different numbers of client connections, to identify the upper limits of concurrent clients that can be served.
- *Spike testing*, trying the system's reaction to sudden increases in the number of connections
- *Soak testing*, trying the system's endurance to sustained load, to verify robustness and memory usage.

Apache JMeter [2] is one of several performance testing tools publicly available. Apache JMeter has been chosen for SemaGrow due to the wide range of monitoring tools that can be integrated and its versatility in being configured for different needs. For SemaGrow testing in particular, Apache JMeter is capable of using current query logs to test scalability against a heavier user load users. Combined with the SemaGrow RDF Triple Generators [5], Apache JMeter covers SemaGrow's requirements for testing system scalability against future data and usage growth.

Key JMeter concepts are the *test plan* and the pairs of *samplers* and *listeners*. The *test plan* is an XML file that describes a series of steps JMeter will execute when run [3, Section 3]. The *samplers* [3, Section 4] tell JMeter about the requests that should be sent to the server that is being tested and the corresponding *listeners* [3, Section 14] can be used to store and/or visualize the output of the performance test. Main monitoring values include the overall throughput of bytes, the average, minimal and maximal duration of a single request as well as the response code of the request.

Apache JMeter will be fed with a list of SPARQL Queries derived from logs collected during piloting [4] and transformed to a CSV file for easy maintenance. These SPARQL Queries will be issued against a settable SemaGrow SPARQL Endpoint and run concurrently by the defined number of users. The default configuration of the testing environment uses the Apache JMeter components shown in Table **2**.

Apache JMeter offers two ways of running the performance tests defined in a workspace:

- Inside Apache JMeter's native graphical user interface; and
- As standalone Java application that can be started from the console using the Java Runtime Environment.

During development, the console performance test will be run periodically on the *SemaGrow Stack Build Server* [7] as a *cron job*. The results will be published on the build server along with the defined federation of SPARQL Endpoints, the SPARQL queries and the hardware used on the build server. **Error! Reference source not found.** gives an overview of the ecessary steps to run the SemaGrow JMeter Performance test inside Apache JMeter's graphical user interface.

Table 2: Apache JMeter Configuration Components

Thread Group	Main Configuration Component for number of users and requests.
HTTP Request Defaults	Used to define the SPARQL Endpoint against which all requests are issued.
HTTP Request	The definition of the HTTP Request.
SPARQL Queries as CSV	Configuration node to define SPARQL Queries input via a CSV file.
Monitor Results	General Server Performance Monitoring
Aggregate Graph	Aggregated Results for Server Performance including average, minimal or maximal response times.
View Results in Table	Similar to the above monitoring component including bytes transferred and latency.
View Results in Tree	The HTTP Response including response code and the results of the SPARQL Queries.
Graph Results	The results from Aggregate Graph displayed graphically.

Table 3 Installation of Apache JMeter

Download Apache JMeter 2.11	http://tweedo.com/mirror/apache//jmeter/binaries/apache-jmeter-2.11.zip
Unzip the downloaded zip file to any directory	This directory is referred to as ${jmeter-home}
Make the binaries executable	This is an optional step that might be necessary to be able to execute the binaries. On Linux this can be achieved by issuing: chmod+x ${jmeter-home}/bin/jmeter
Run Apache JMeter	Start Apache JMeter by calling either ${jmeter-home}/bin/jmeter or ${jmeter-home}/bin/jmeter.bat on windows environments respectivly.

Besides the build server deployment that will be used to evaluate SemaGrow performance and to measure the extent to which SemaGrow has achieved its objectives, the ART will also be bundled as part of the *SemaGrow Stack* distribution in order to assist deployment administrators.

Development on the SemaGrow Automated Rigorous Tester takes place at:

- The main SemaGrow Stack repository, working logging functionality into the stack: https://github.com/semagrow/semagrow.git
- The SemaGrow JMeter repository, using the JMeter framework for executing automated tests: https://github.com/semagrow/semagrow-stack-jmeter.git
- The repositories of specific SemaGrow tools that use the ART framework. STRHist, for example, extracts query workloads from the SemaGrow logs in order to refine histograms (cf. D3.1):
- https://bitbucket.org/acharal/strhist

3.2 JMeter Monitoring Tool

To demonstrate the capabilities of the JMeter tool we provide an example of stress tests that can be performed by the administrator. The scenario involves a federation of two data sources, namely the AGRIS and the AGRIS Crawler databases. The administrator aims to stress tests the current deployment of this SemaGrow federation to multiple users and as a result uses JMeter to simulate multiple concurrent users.

Each row of Table **4** corresponds to a single run of the JMeter tool. The first column shows the number of the number of the users. Each user issues a set of 19 queries, which were randomly selected from the FAO 1st pilot workload. The second column shows the total number of queries that were issued in the server. The third column shows the size in bytes that were returned from the server to all users, while the fourth column shows the total execution time of all queries. In some cases, the server didn't return any results due to a 500: Internal Server Error. The number of errors are shown in the fifth column. Finally, in the last column shows the time (in msec) / size (in MB) ratio.

The raw results produced by JMeter are presented in Table **4**. Moreover, to facilitate further analysis of potential problems, JMeter can also produce charts (see, for example, Figure 2 and Figure 3).

Table 4 Results produced from multiple runs of JMeter

# of users	# of total queries	total size (bytes)	total time (msec)	# of errors	time / size
1	19	108047	44116	0	0.41
2	38	216067	103956	0	0.48
5	95	540177	558421	0	1.03
10	190	1080654	2194226	0	2.03
20	380	2158233	11038902	1	5.11
50	950	5401894	89205414	2	16.51
100	1900	10806022	374421610	3	34.65
200	3800	21683158	1540363774	4	71.04

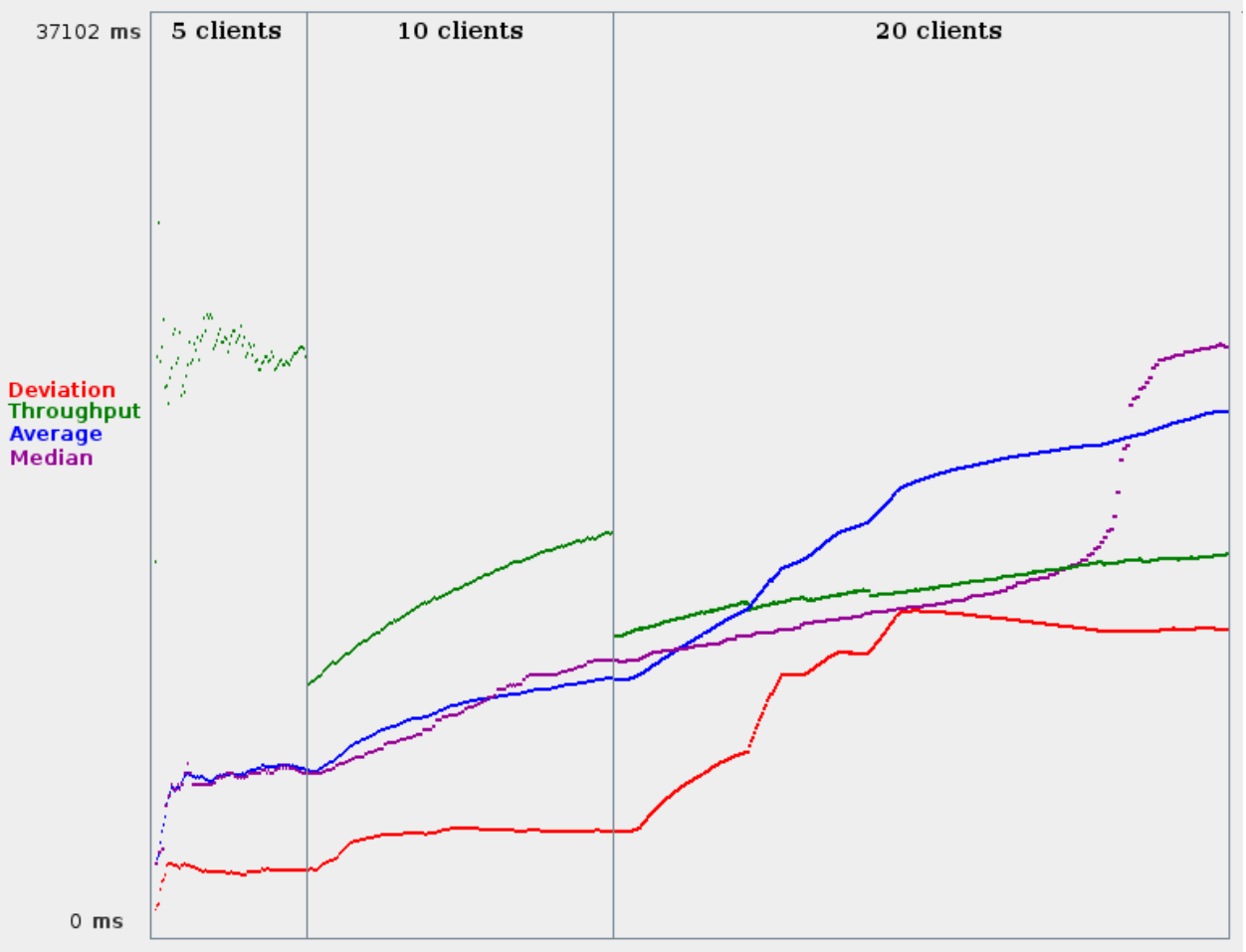

Figure 2. The plotted statistics are 1) the Average query execution time (in blue), 2) the Median (in purple), 3) the Standard Deviation (in red) and 4) Throughput (in green) for runs of 5, 10, 20 clients respectively.

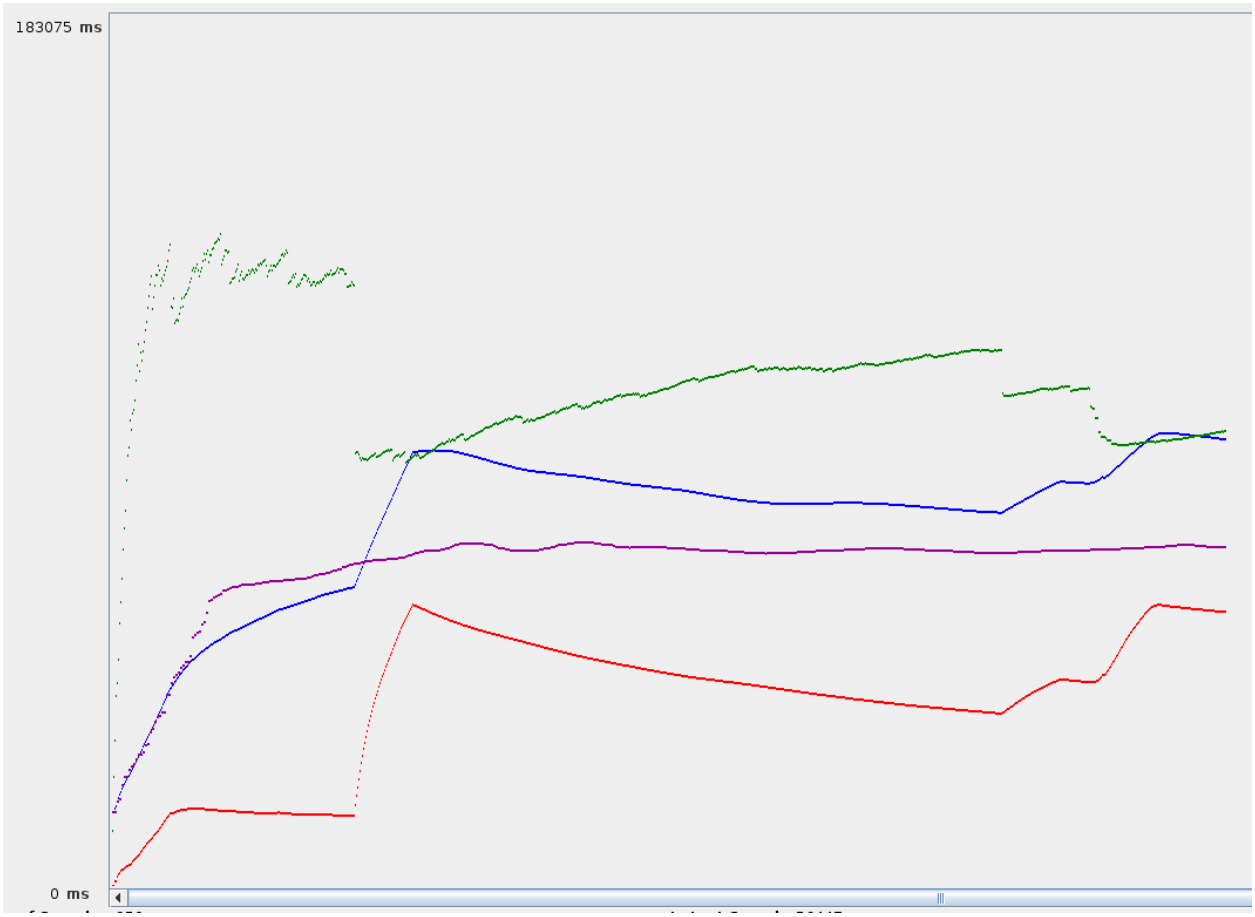

Figure 3. The plotted statistics are 1) the Average query execution time (in blue), 2) the Median (in purple), 3) the Standard Deviation (in red) and 4) Throughput (in green) for runs of 50 clients.

3.3 ART Logging and Store

The *ART Logger* is implemented *Apache Logging Services* [8], configuring a PostgreSQL server as the *ART Store* backend. This solution has the limitation that the Apache loggers are geared towards writing out text messages and not structured information (times, sizes, etc.) in a database. This has been circumvented by parsing

On the other hand, Apache (and similar, such as SLF) logging mechanisms are efficient and convenient to use, so we will pursue a design that is similar, but retains the distinction between the log message/type and the parameters that will be stored in separate database columns. Such a design will offer the best of both worlds:

- Retaining the API of widely used logging services, giving ease of use of the testing framework;
- Storing in a database schema from which the conventional serialized form can be easily produced, if needed; but at the same time
- Storing in a database schema that can be efficiently queried for the structured parts of the log, such as times, sizes, etc.

We will strive to embed this design into a currently use logging framework, but satisfying the requirement of structured representation will take precedence if need be.

REFERENCES

[1] Performance Test Tools Survey, http://www.opensourcetesting.org/performance.php

[2] Apache JMeter, http://jmeter.apache.org

[3] *Apache JMeter User Manual*, http://jmeter.apache.org/usermanual

[4] *Controlled Pilot Trials*. SemaGrow Public Deliverable D6.3.1. November 2014.

[5] *RDF Triple Generator of Realistic Data Sets.* SemaGrow Public Deliverable D4.3. November 2014.

[6] *Scalability and Robustness Experimental Methodology*. SemaGrow Public Deliverable D4.1, October 2013.

[7] *Integrated SemaGrow Stack API Components*. SemaGrow Public Deliverable D5.4.2, April 2014 (M18).

[8] Apache Logging Services, http://logging.apache.org