Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Scalability and performance evaluation report

Deliverable number	*D5.3*
Dissemination level	*Public*
Delivery date	*26 August 2011*
Status	*Final*
Author(s)	*Konstantin Pentchev, Vassil Momtchev*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Abstract

Deliverable D5.3 reports on the performance and scalability results of Khresmoi Biomedical Knowledge Server (KS). KS takes central place in the KHRESMOI architecture by providing a generic and efficient web interface to various types of biomedical datasets required by the use cases. The document evaluates the developed server in D5.2 in terms of the underlying RDF database – OWLIM, custom developed query API and use case datasets performance.

Table of Contents

1 Executive summary

Deliverable D5.3 reports on the performance and scalability results of Khresmoi Biomedical Knowledge Server (KS). KS takes central place in the KHRESMOI architecture by providing a generic and efficient access to various biomedical datasets required by use cases. The document evaluates the developed server in D5.2 [1] in terms of the underlying RDF database – OWLIM [2], custom developed query API and use case datasets loading and query answer performance.

During the evaluation of the KS performance and scalability two separate approaches are undertaken. The measurement of the batch processing and the online service are conceptually different tasks that require separate approaches and sets of metrics in order to quantify them. The tests delivered a series of absolute values for the performance metrics, which not only reported the current status of the system, but allowed us to measure the improvements to the technology.

As this evaluation is performed during year two of the project and has to be concluded before the first code freeze and user-centred evaluation, it will be relevant for the current set and amount of data loaded, which are based on the requirements provided by the use cases. OWLIM is a RDF database with support of lightweight inference. The semantic expressivity is deliberately limited in order to meet the requirements for regular dataset updates, schema and data source evolution. Thus, we can keep the high system scalability, tractability and predictive query response time.

2 Introduction

The scope of task T5.3 "Evaluation of biomedical knowledge server", of which this deliverable is part, is to evaluate the performance and scalability of the KS, with respect to the results of T5.1 and T1.3. We identified two general components that require distinct evaluation approaches due to the principal differences between them: the offline loading of data into the knowledge base (KB) – the batch process; and the online response to user requests – the online services. The former component concerns itself with the management of the KB – loading and maintaining up to date the growing amounts of data in the semantic repository. The OWLIM database integrates forward chaining reasoner that materialize nearly all implicit information with the exception of owl:SameAs equivalence on loading time. The time to load the knowledge base can become a serious constrain depending on the size of the repository. The latter is responsible for online presentation of data, i.e. evaluating SPARQL [3] queries (among others) and delivering results to the users in real time. The important factor here is to measure if the KS would be able to deliver results fast enough in order to meet the use case requirements.

In the context of the batch process performance can be defined as the ability of the service to successfully load a given amount of data into the semantic repository in a certain time frame. This is a critical property of the KS, as it will define when and how we load and update data to the KB. Because it is planned to run only once in a fixed period of time and should be separated from the live system, we can assume that the success margin is in a couple of hours rather than in seconds or milliseconds. The second objective – to test the scalability of the KS – aims at defining the limits of the system. In the context of the batch process this translates to finding the maximum amount of statements that can be added to the KB in reasonable time and investigate how the increasing amount of data affects the performance of the loading process.

The online process has one main objective – to deliver query results and answer user requests in real time. To give a more definite measure of success, this task must be performed in the space of milliseconds to guarantee near to real-time response. The performance can be affected both by the amount and structure of the data in the repository, but also by the availability of resources at a given moment – network bandwidth, CPU time, memory etc. In terms of scalability the limit of the online process can be defined as the maximum number of simultaneous requests that can be served per a specificied time frame.

In order to properly benchmark these two components of the KS we had to devise two sets of metrics and test procedures that will profile their performance and identify possible issues and bottlenecks.

3 Metrics for scalability and performance

In this section we present metrics used to measure the scalability and performance for the different executed tests. The two metrics are closely related and measures how well the system is capable to handle an increased volume of information or a number of concurrent users in terms of loading and query response time.

3.1 Batch process

The performance of the batch loading depends on the scale of the loaded data, the complexity of the used inference and the specific properties of the datasets like having very long literals or URIs. Therefore, the following metrics were specified:

- N^{expl} – the number of explicit statements (triples)
- N_i^{expl} – the number of explicit statements for dataset i
- N^{impl} – the number of implicit (inferred) statements
- N – the sum of N_{expl} and N_{impl}

Additionally, we must consider a special dataset – the mappings generated by the Khresmoi mapping rules. These link entities and relations between datasets and could lead to huge amounts of inferred statements being generated. In order to quantify their significance we defined the following metric:

- N^{map} – the number of instance mapping links

The actual variable that measures the performance of the batch process is the time required to load all data in the repository – T. However, it might be interesting to measure the time required by each dataset, as these differ in size and composition and this may provide useful insights for further optimization. Additionally, we are interested in the speed of loading statements in the repository. This would not only give us a precise performance metric, but will allow us to predict future loading times. Consequently, the following variables are defined:

- T_i – the loading time of dataset i
- T – the total loading time of all datasets; the sum $T_{1..n}$
- P_i – the average loading speed of dataset i; measured in statements per second
- P – the average loading speed of all datasets; the sum $P_{1..n} * N_{1..n}^{expl} / N^{expl}$
- ΔP_i – the change in average speed for dataset over the number of statements loaded
- ΔP - the change in average speed for all datasets

3.2 Online service

The crucial metric for online service performance is the time taken to complete a request by the server in the context of a specific number of concurrent user requests. Thereby, not only the average value is important, but much more the maximum, as it can render the service virtually unusable in certain situations. It is desirable that the response times are stable, i.e. that their variance is small. We have identified the following metrics that will help us quantify the online performance of the KS:

- RP^{i}_{avg} – the average response time for service i
- RP^{i}_{max} – the maximum response time for service i
- RP^{i}_{min} – the minimum response time for service i
- RV^{i} – the response time variance for service i

In addition, we need to monitor the amount of data transferred to the client, as it will have implications for bandwidth requirements. If responses are huge, their transfer can lead to excessive network traffic, exhausting the available bandwidth and effectively blocking the service. We identify two metrics to keep track of:

- D^{i}_{avg} – the average response size for service i
- D^{i} – total response size for service i (in a series)

Currently, there are three online services provided by the KS and integreated in the use cases as defined in [5]:

- Disambiguator Service – accepts as input keyword and suggest ranked concepts that disambiguate its meaning; allow various Khresmoi components to prompt the user to specify the context of the used keyword;
- SPARQL endpoint – accepts as input SPARQL 1.1 query and returns results; the scope of the current tests is limited only to SELECT queries; multiple components loads data to populate gazetteer lists or translate concepts;
- Resource view – exposes the RDF resources as linked data; the scope of the current tests excludes this service since its implemented on top of the SPARQL endpoint;

In order to assess the scalability of the KS online services we need to simulate high request loads, i.e. simultaneous access by multiple users. Therefore, we can define a time series G, of performance tests over a growing number of users N_u.

- ΔRP^{i}_{avg} – the change in average response time for service i over time series $G_{0..n}$
- ΔRP^{i}_{max} – the change in maximum response time for service i over time series $G_{0..n}$
- ΔRP^{i}_{min} – the change in minimum response time for service i over time series $G_{0..n}$
- RV^{i}_{G} – the response time variance for service i over all time series in $G_{0..n}$

Because we are interested in the amount of change between time series rather than its direction, we take the square root of the squared differences. Here are the definitions:

- $\Delta RP^{i}_{avg} = 1/(n-1)\sum \sqrt{(RP^{i}_{avg}(k) - RP^{i}_{avg}(k-1))^2}$, where $RP^{i}_{avg}(0)$ is the single user sampling average response time
- $\Delta RP^{i}_{min} = 1/(n-1)\sum \sqrt{(RP^{i}_{min}(k) - RP^{i}_{min}(k-1))^2}$, where $RP^{i}_{min}(0)$ is the single user sampling minimum response time
- $\Delta RP^{i}_{max} = 1/(n-1)\sum \sqrt{(RP^{i}_{max}(k) - RP^{i-1}_{max}(k-1))^2}$, where $RP^{i}_{max}(0)$ is the single user sampling maximum response time

4 Evaluation of scalability and performance approach

The scalability and performance evaluation of RDF database is a complex and well researched topic. [6] provides a very sophisticated overview of the existing benchmarks, benchmark configurations and results for different RDF engines. The OWLIM database publishes regular update of the latest performance and scalability metrics in terms of inference and data loading speed. Lehigh University Benchmark (LUBM) defined in [7] is a popular benchmark to measure the scalability of very large RDF stores. The test evaluates the loading time of a synthetically generated dataset including the calculation of the inference closure and the query answering time. At present distributed Hadoop based databases are able to break 100 billion RDF statements barrier [8] for datasets with limited semantic expressivity to OWL-Horst reasoning. In [9] there are also published query answering results and discussions on various optimizations. Unfortunately, the LUBM query results are suboptimal because for very large datasets some of the queries return million of results, which is unrealistic in a real life use case. That is why both [8] and [9] perform experiments with real datasets like Ontotext's LLD, where different optimization approaches depending on the specific dataset must be considered. As a general conclusion we can summarize that the RDF databases scale well beyond the size of all existing RDF statements in the linked data cloud. Nevertheless, the query answering and inference approach should be adepted to the various use case scenarios in order to yield maximum performance and scalability. The current Khresmoi KB is in the range of 10 billion statements. This scale of data is considered processeable by a single OWLIM server and that is experimentally proven in the latest OWLIM 5.x benchmark results [10].

The current effort to evaluate the KS scalability and performance is limited only to the Khresmoi KB specifics. It primary aims to test the batch and online service performances with respect to the requested datasets. The processes are described in the following subsections and have been designed to best reflect the specifics of KS information loading including semantic mappings and required web service interfaces.

4.1 Batch process

In the context of the Khresmoi project we would apply a much more narrow evaluation approach in relation to the use case dataset and the execution of the required semantic mapping. In order to precisely measure the performance metrics defined, a full reload of the KB was necessary. We setup the job described in section 4.2 "Process and behaviour" of D5.2 [5] with the addition of a tFlowRateMeter component (see chapter 5), which records statistics about the work performed at every 10'000 statements processed. The recorded values by this generic component are:

- Time
- Total number of rows
- Number of rows for the interval
- Average speed
- Average speed for the interval

The values are recorded in parallel for each loaded dataset and allow us to derive from the metrics N_i^{expl}, N^{expl}, Pi, P, T_i, T. The batch test was run on a 64-bit Open Solaris VM, with 2 Xeon 5420 CPU (2.5 GHz), 64GB RAM and SSD disks in RAID 0. KS is updated to OWLIM-SE 5.1-20120518.113230 for the execution of all performance tests.

The setup described in the previous section also allows us to measure the changes in speed with increasing statements - ΔP_i and ΔP.

4.2 Online service

As previously mentioned, the KS actually offers two distinct online services for accessing the KB. One is a default implementation of a SPARQL endpoint for querying and manipulating the data using SPARQL 1.1 query language. It allows the user to perform arbitrary database-like searches within the KB, extracting explicit and implicit relations between arbitrarily removed entities. However, there are tasks for which this endpoint is sub-optimical like prefix-based literal matching of resources against the input string. This is a much more classical text search problem and consequently RDF engines are not really the tool of choice. Therefore, we added an additional full-text index based on Apache Lucene [11] to the KB, which stores references from labels, synonyms and descriptions to URIs. This index is exposed for querying through the Disambiguator service.

The test will investigate the scalability and performance of the system by comparing different approaches of executing the following queries:

Because the disambiguator has to return suggestions for rapidly changing user input, possibly for each entered character, its performance has a critical upper boundary of 100ms. In order to evaluate the service's ability to complete in the defined boundary, we identified five exemplary queries from our user log. These queries are to be repeatedly executed against the service in a random order each for 100 times. This will allow us to measure the predefined performance metrics on a large enough set of requests. Note that this approach is valid only because neither the online service nor the index have caching enabled. The five queries are:

- Resources for "Cancer"
- German resources for "Klein"
- German resources for "Diabete"
- Spanish resources for "Diabet"
- Czech resources for "Leukoc"

Table 1 lists all used test queries in two formats, one as HTTP request for the Disambiguator service and second as a SPARQL query. The SPARQL queries are created as close as possible to the Disambiguator search semantics. Although a small difference is that every SPARQL query will match not only word prefix, but also suffixes. In order to compensate this incompatibility all samples queries were tweaked to return once and the same number of concepts.

Query String	Language	Disambiguator service request	Equivalent SPARLQ query
Cancer	English	autocomplete.json ?q=Cancer&limit=10	SELECT * WHERE { ?s a skos:Concept ; skos-xl:prefLabel ?l . ?l skos:note "Mesh". ?l skos-xl:literalForm ?literal. filter regex(?literal, "cancer", "i") } LIMIT 10
Leuko	Czech	autocomplete.json ?q=Leko@cz&limit=10	SELECT * WHERE { ?s a skos:Concept ; skos-xl:prefLabel ?l . ?l skos:note "MeSH Czech". ?l skos-xl:literalForm ?literal. filter regex(?literal, "Leukoc", "i") } LIMIT 10
Klein	German	autocomplete.json	SELECT * WHERE {

		?q=Klein@de&limit=10	?s a skos:Concept ; skos-xl:prefLabel ?l . ?l skos:note "MeSH German". ?l skos-xl:literalForm ?literal. filter regex(?literal, "Klein", "i") } LIMIT 10
Diabet	Spanish	autocomplete.json ?q=Diabet@sp&limit=10	SELECT * WHERE { ?s a skos:Concept ; skos-xl:prefLabel ?l . ?l skos:note "SNOMED Terminos Clinicos". ?l skos-xl:literalForm ?literal. filter regex(?literal, "Diabet", "i") } LIMIT 10
Diabete	German	autocomplete.json ?q=Diabete@de&limit=10	SELECT * WHERE { ?s a skos:Concept ; skos-xl:prefLabel ?l . ?l skos:note "MeSH German". ?l skos-xl:literalForm ?literal. filter regex(?literal, "Diabete", "i") } LIMIT 10

Table 1 Disambiguator requests and equivalent SPARQL queries

In order to measure the scalability of the service we simulate concurrent requests in a time series of increasing user numbers. We would like to measure the change in performance for increases by an order of mangitute. Therefore, the same queries are executed for 10, 100 and 1000 concurrent users. Each user has to execute the five queries in a random order. The sampling experiment with 10 users will be adjusted so that each user executes the queries 5 times in order to get a more statistically significant number of measurements. Users start sending requests at different times, but they should all be active after a certain time when the test started. To make the simulation more realistic, we introduce a small delay between queries based on a normally distributed variable with mean 3 seconds and standard deviation of 1 sec. From the results of these time series we should be able to calculate the change in average, minimum and maximum response times and their variance.

5 Test infrastructure

The chapter describes the used software to perform all benchmark tests of the KS. All the batch processes are executed with code generated by the Talend environment. In [1] section 6.1 is presented a more detailed introduction of the Talend environment. The online performance results are completed with the help of JMeter and a modification of the Berlin SPARQL Benchmark tool client introduced into this document.

5.1 Components for batch process

The loading of the Khresmoi KB is implemented via a multi-staging loading process. The loading process passes through multiple steps:

- Prepare the dataset for import by removing all redundant statements and translate the RDF resources to OWLIM internal identifiers. The redundant statements positively affect the loading speed and would deliver faster results because they would not write to the disk new information; thus, we isolate the specifics how the different datasets are generated i.e. often they are transformed from a non-fully normalized data source.

Dataset	RDF [statements]	N_i^{expl} [unque statements]	Redundant statements (%)
UMLS (MeSH, FMA, SNOMED)	121'585'973	82’524’114	32%
RadLex DL and Full (English and German)	599'488	313’065	48%
ClinicalTrials.gov	24'653'008	24’651’326	0%
HON Certified Websites	216'780	138’424	36%
Geonames	107'834'991	107’834’991	0%
DrugBank	766'920	517’023	33%
PubMed	586'980'736	402’210’882	31%
Instance Mappings	37’707	37’707	0%
Semantic Annotations	259’252’739	158’432’633	39%
TOTAL	**1’101’928’342**	**776’660’165**	**30%**

Table 2 The number of unique statements in the RDF files per dataset

- Apply for every dataset the procedure of creating a new empty repository and loading all preprocessed statements to it. So, we gain statistics about the inference closure materialized for every individual dataset;
- Merge all isolated repositories into a single RDF warehouse that provides efficient access to all information
- Generate instance mappings and semantic annotations as a new dataset, which is later imported via the same loading methodology

Although, the focus of WP5 is not text analysis a close collaboration with WP1 exists in the context of T1.3 "Coupling semantic annotation to Biomedical Knowledge Resource" in order to enable more interesting analytical queries the knowledge base is enriched with semantic annotations between PubMed and UMLS (FMA, SNOMED-CT, MeSH, RadLex). The semantic annotations help bridging the ambiguity of the natural language when expressing notions and their computational representation in a formal language (see Figure 1).

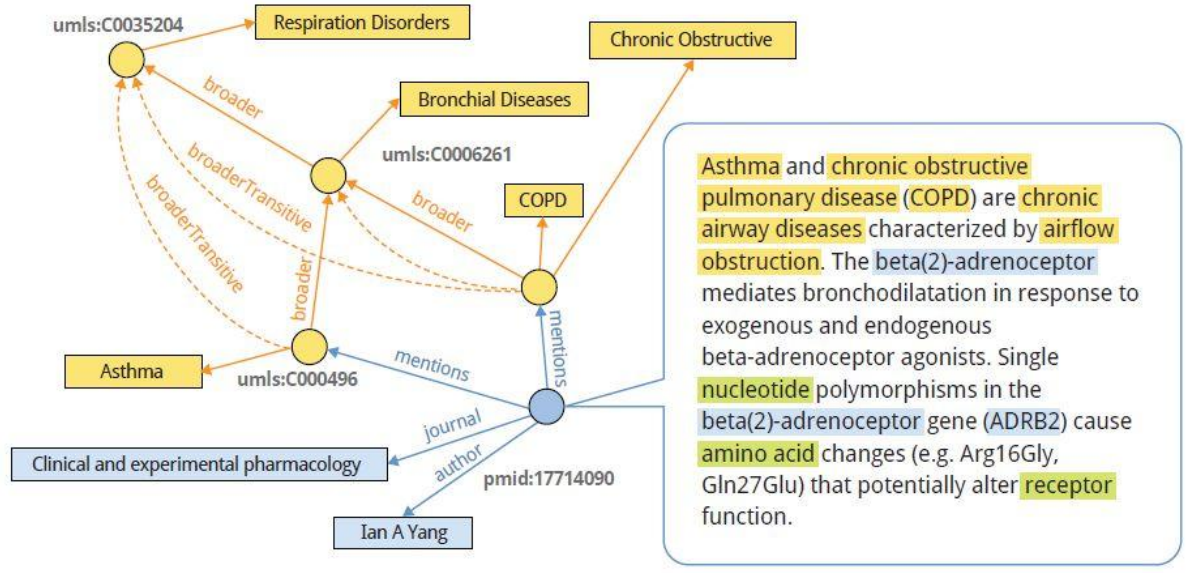

Figure 1 Semantic annotations interlinking PubMed with UMLS concepts

The semantic annotations dataset contains statements linking PubMed citations (subjects of type http://khresmoi.eu/resource/pubmed/Citation) to UMLS concepts (subjects of type skos:Concep in skos:inScheme http://linkedlifedata.com/resource/umls). All relations are generated by Semantic Biomedical Tagger 1.2 [12], which is a commercial text-analysis product of Ontotext capable of recognizing over 133 biomedical types of information. In order to better type the generic nature of the mention relation two predicates are used:

- http://linkedlifedata.com/resource/mentions/title - if the match is in the PubMed title
- http://linkedlifedata.com/resource/mentions/abstract - if the match is in the PubMed body

The total number of semantic annotation is 158'432'633 that link 5'018'528 PubMed abstract with UMLS.

5.2 JMeter for Online process

For implementing the evaluation approaches defined in sections 4.2 and, we used Apache JMeter – an open-source desktop application for conducting functional and performance tests of web applications

[13]. It can be used to test both static and dynamic resources and simulate heavy load for testing service strength and performance under concurrent load. It comes out of the box with components for graphical analysis as well as report generation.

The tool provides a powerful interface to design integration or performance tests and monitor the execution time. All subsequent online query tests presented into the current report are executed with its assistance.

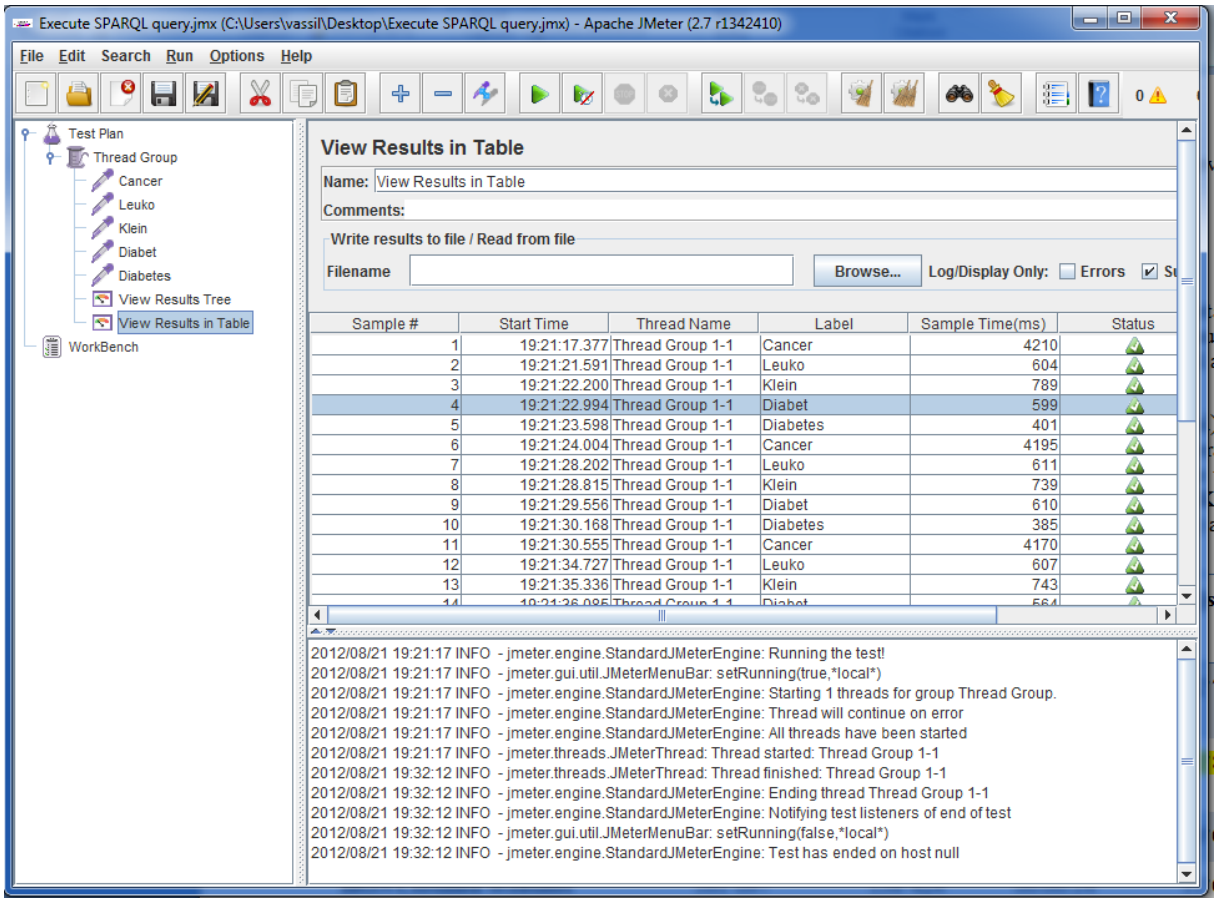

Figure 2 The result view of JMeter

6 Results for performance and scalability

The chapter focuses over the details and mechanics of the benchmark execution tests and online service performance figures.

6.1 Batch process

The batch process evaluates the time needed to load the Khresmoi KB from scratch. The details around the process setup are described in sections 3.1 and 4.1. The preprocessing steps is measured, but not considered as part of the execution tests since the implemented algorithm is suboptimal and aimed to deduplicate the input RDF statements.

The total time to load every datasets part of Khresmoi KB (776'660'165 statements in total) in separate OWLIM repositories is less than 10 hours on a standard server. Table 3 presents the average loading speed and the time needed to process every individual dataset. OWLIM is configured to use custom inference rule set just enough to express the RDFS semantics extended with rules for SKOS transitive (skos:broader and skos:narrower), symmetric (skos:exactMatch and skos:closeMatch) and reflexive (skos:broaderTransitive and skos:narrowerTransitive).

Dataset	N_i[statements]	N_i^{expl} [statements]	T_i [hh:mm:ss]	P_i [st/s]
UMLS (MeSH, FMA, SNOMED)	110'044'591	82'524'114	00:56:00	32'751
RadLex DL and Full (English and German)	697'716	313'065	00:00:32	21'803
ClinicalTrials.gov	27'508'674	24'651'326	00:12:43	36'053
HON Certified Websites	161'097	138'424	00:00:14	11'056
Geonames	245'042'640	107'834'991	02:13:05	30'687
DrugBank	611'630	517'023	00:00:18	33'979
PubMed	477'361'349	402'210'882	04:22:37	30'295
Instance Mappings	37'707	37'707	00:00:01	37'707[1]
Semantic Annotations	163'530'190	158'432'633	01:39:35	27'369
TOTAL	**1'020'045'594**	**776'660'165**	**09:25:05**	**30'085**

Table 3 Loading time in OWLIM for the different datasets

[1] The value has too big statistical error because of the short measurement period.

In order to trace the scalability, the average loading speed is monitored every 10 seconds. The test demonstrates if the RDF engine is capable to scale in terms of increased volume of persisted data and how it affects the ΔP and ΔP_i values.

Figure 3 indicates the average loading speed per 10 second periods during the complete load of PubMed dataset. The axis indicates the timestamp of the log entry and how many statements were processed during the given period. Dynamics of the diagram shows a close to random peaks and drops in the loading speed that is likely to be caused by dataset specifics like ordering of the RDF resources, length of literals, garbage collector cycles or the interaction with another operating system processes.

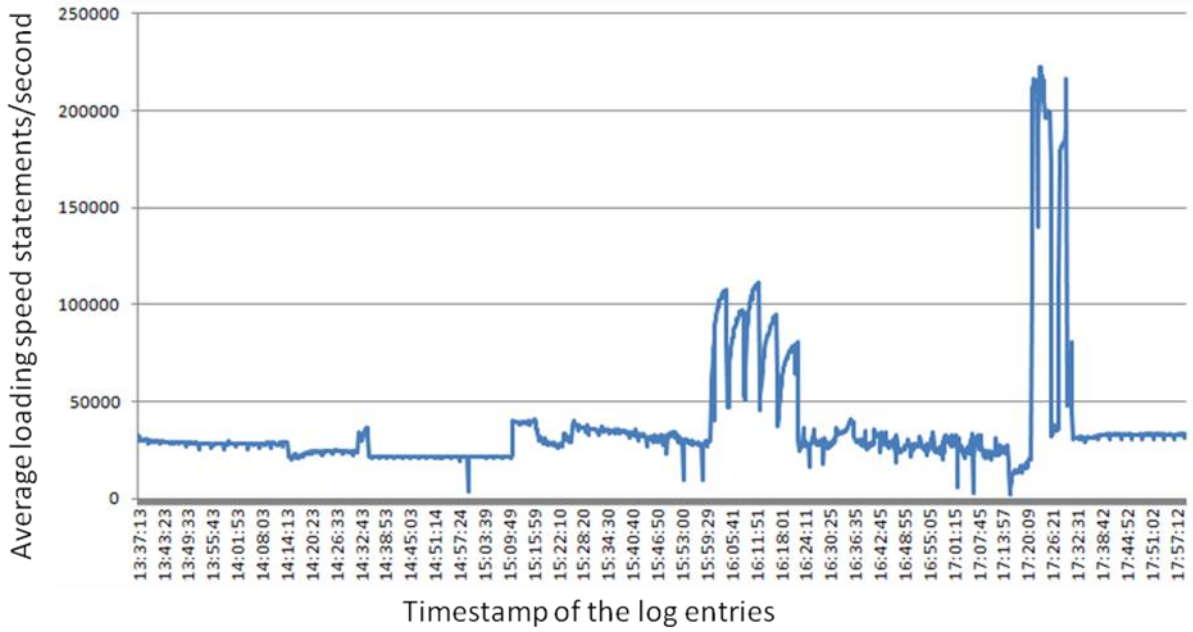

Figure 3 The average loading speed per 10 second periods for PubMed (P_{PubMed} [st/s])

indicates the same type of diagram for Geonames another dataset with significant size. The curve has similar fluctuation like for PubMed (the chart covers smaller period). Once again we cannot observe any definitive trend in decreasing speed.

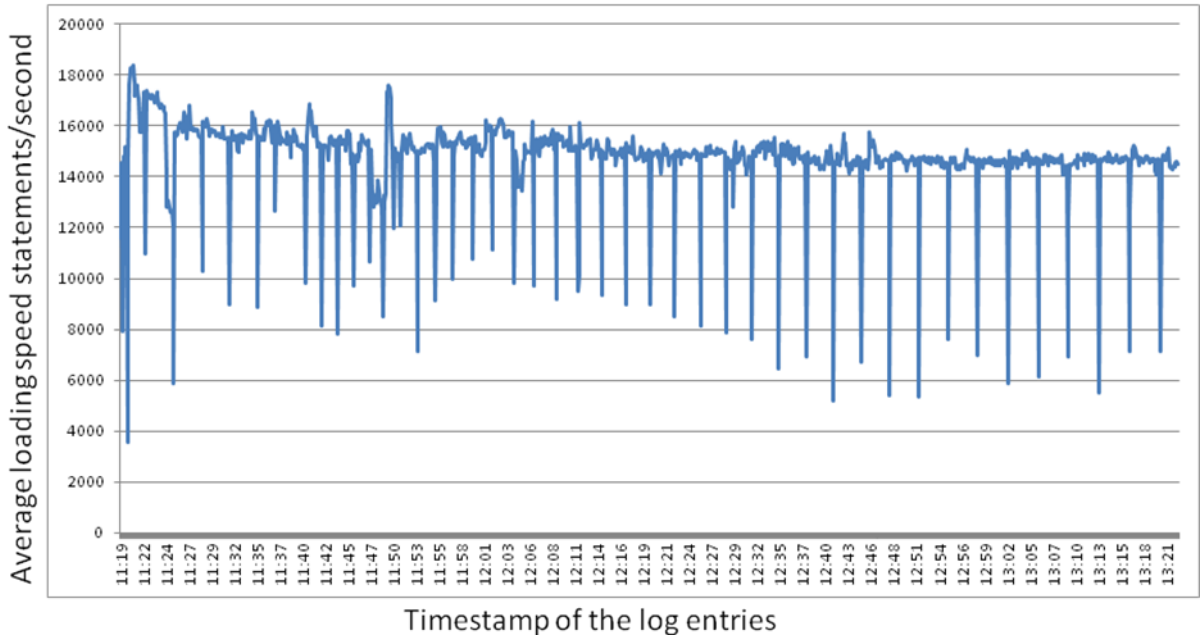

Figure 4 The average loading speed per 10 second periods for Geonames ($P_{Geonames}$ [st/s])

Once all datasets are loaded into separate OWLIM repositories it remains the last step to merge them into a single queryable store, which does not requires federation and therefore better query-time performance. The time to aggregate all datasets into a single RDF warehouse is relatively very quick and completed for less than 1 hour on the same server environment. Thus, the total time to introduce a new dataset or substitude an existing one will be roughly equal to the time to load it into individual repository and then merge it. The results look very promising and future investigation will be done in the context of the upcoming delibverable D5.5 "Sustainable biomedical knowledge infrastructure".

6.2 Online service

In this section we present the results of the Disambiguator service scalability evaluation and compare it to SPARQL query execution. The results of the evaluation approach defined in section 4.2 are presented in Table 4 for the SPARQL queries and Table 5 for the improved indexing. The values are measured in milliseconds unless otherwise specified.

Request	N[requests]	RP^i_{avg} [ms]	RP^i_{min} [ms]	RP^i_{max} [ms]
"Cancer"	100	4170	4116	4261
"Leukoc", language=cz	100	617	596	705
"Klein" language=de	100	758	726	848
"Diabet" language=sp	100	570	555	610
"Diabete" language=de	100	431	385	464

Table 4 Disambiguator sampling simulated with SPARQL queries 500 requests with 1 user

The database query optimizer chooses one and the same execution plan for all five SPARQL queries. The "Cancer" query returns the first 10 matched results for 4.17 seconds (average), which is over 5 times longer than the second slowest query. All execution values are consistent with the expected behaviour – the time increases with the number of executed regular expression tests; the terms in English are 320'738 compared to 38'184 (German) and 31'213 (Czech). Another factor is the selectivity of the of the searched term – "Klein" has longer execution time than "Diabete" because the term is much less frequent and more literals must be evaluated.

Table 5 presents the same test results but executed against the Disambigator service. Not surprisingly, the use of full-text indexes dramatically reduces the execution time and all sorts of fluctuation. All variances are now close to the statistical error.

Request	**N**	**RP^{i}_{avg} [ms]**	**RP^{i}_{min} [ms]**	**RP^{i}_{max} [ms]**	**$\sqrt{RV^{i}}$ [ms]**	**D^{i} [KB]**	**D^{i}_{avg} [B]**
"Cancer"	100	14	10	30	2.245	2.50	4143.92
"Leukoc", language=cz	100	16	13	32	2.35	4.35	7199.94
"Klein", language=de	100	13	10	17	1.13	1.05	1733.9
"Diabet", language=sp	100	15	11	26	1.86	2.95	4864.92
"Diabete", language=de	100	12	9	24	1.95	0.22	356.92
TOTAL	**500**	**14**	**9**	**32**	**2.45**	**11.00**	**3659.92**

Table 5 Disambiguator sampling with 1 user for 500 requests

From the last row we can read the following values for the metrics specified in section 3.2 :

- $RP^{Disambiguator}_{avg}$ = 14 ms
- $RP^{Disambiguator}_{max}$ = 32 ms
- $RP^{Disambiguator}_{min}$ = 9 ms
- $RV^{Disambiguator}$ = 6.0025 ms[1]

It is obvious that for a single user the service easily satisfies the requirement to respond bellow 100ms for a single user. Moreover, based on the measured variance the response times under these conditions are significantly proven to be within the defined boundary. The metrics for network data transferred are also quite low – $D^{Disambiguator}_{avg}$ = 3659.92 B/response and $D^{Disambiguator}$ = 11 KB for 500 responses. A simple calculation shows that with a low-end bandwidth of 16 Mbit/s it will be theoretically

[1] derived from the measured standart deviation, denoted as $\sqrt{RV^{i}}$

possible to stream the results for ~50000 requests to the Disambiguator service. Consequently, we can ignore the response size from further consideration with respect to the service. Tables 6, 7 and 8 measures the scalability of the service and how the performance is affected with respect to an increased number of concurrent query requests.

Request	N	RP^i_{avg} [ms]	RP^i_{min} [ms]	RP^i_{max} [ms]	$\sqrt{RV^i}$ [ms]	D^i [KB]	D^i_{avg} [B]
"Cancer"	50	15	11	42	5.47	15.24	4864.92
"Leukoc", language=cz	50	13	10	30	3.14	5.45	1733.94
"Klein", language=de	50	19	11	271	36.04	21.75	6926.22
"Diabet", language=sp	50	16	11	63	10.02	12.82	4107.96
"Diabete", language=de	50	12	8	41	5.45	1.17	356.94
TOTAL	**250**	**15**	**8**	**271**	**17.34**	**52.54**	**3597.99**

Table 6 Disambiguator sampling with 10 concurrent users (5 repeats)

Request	N	RP^i_{avg} [ms]	RP^i_{min} [ms]	RP^i_{max} [ms]	$\sqrt{RV^i}$ [ms]	D^i [KB]	D^i_{avg} [B]
"Cancer"	100	17	10	207	21.51	4.77	1720.22
"Leuko", language=cz	100	17	10	92	11.50	11.54	4143.91
"Klein" language=de	100	17	11	77	9.35	20.96	7127.09
"Diabet" language=sp	100	14	8	80	11.73	1.10	356.97
"Diabete" language=de	100	16	10	67	7.11	15.53	4774.83
TOTAL	**500**	**16**	**8**	**207**	**13.24**	**49.77**	**3624.60**

Table 7 Disambiguator sampling with 100 concurrent users

Request	N	RP^i_{avg} [ms]	RP^i_{min} [ms]	RP^i_{max} [ms]	$\sqrt{RV^i}$ [ms]	D^i [KB]	D^i_{avg} [B]
"Cancer"	1000	16	10	232	11.64	22.33	4874.75
"Leuko", language=cz	1000	12	8	108	8.34	1.64	364.98
"Klein" language=de	1000	16	9	222	12.54	19.11	4105.54
"Diabet" language=sp	1000	14	9	184	10.32	8.04	1725.07
"Diabete" language=de	1000	18	11	182	12.06	32.79	7157.93
TOTAL	**5000**	**15**	**8**	**232**	**11.22**	**81.64**	**3645.65**

Table 8 Disambiguator sampling with 1000 concurrent users

From these test results we calculate the following values for the metric defined in section 3.2 :

- $\Delta RP^{ij}_{avg} = 1/(4-1)\,(\sqrt{(15-14)^2} + \sqrt{(16-15)^2} + \sqrt{(15-16)^2}) =$ **1ms**
- $\Delta RP^{ij}_{min} = 1/(4-1)\,(\sqrt{(8-9)^2} + \sqrt{(8-8)^2} + \sqrt{(8-8)^2}) =$ **0.33 ms**
- $\Delta RP^{ij}_{max} = 1/(4-1)\,(\sqrt{(271-32)^2} + \sqrt{(207-271)^2} + \sqrt{(232-207)^2}) =$ **109 ms**

It is clear from the values for the differences of minimal and average response times that the changes for increasing amounts of users are not significant. Only for the maximum response time do we see an increase of ~100ms for increasing the number of concurrent users by an order of magnitude. For a common search service this would not be a big concern because theoretically[1] you could support 10^9 concurrent users. However, we defined that for our service we require response times maxima around 100ms, which will be exceeded easily. A closer look at the values themselves reveals that there is actually a huge increase between $N_u = 1$ and $N_u = 10$, but actual decrease for subsequent increases in the number of users and $RP^{Disambiguator}_{max}$ stays in the boundaries of [200 ms, 300 ms]. These small fluctuations can be attributed to network stability, server io time or other hardware performance variations. Still, we can conclude that the service performance is stable within these boundaries for increasing users and while it is above 100ms it will be relevant only for a small fraction of requests and with a difference that will be hardly noticed by human users.

A more interesting aspect of the online query performance is the ability to answer complex knowledge base questions. For example in the context of task T5.4 "Consistency checking rules for information extraction" and the clinical radiology use case it is interesting to know if an extracted pathology can fit to the identified anatomical location:

[1] In practice this will be limited by other factors, e.g. network bandwith and server hardware resources.

```
SELECT ?provenance ?s1Label ?radLexAnatomy ?s2Label ?radLexPathology
WHERE {
 # Relation properties
 ?r <http://linkedlifedata.com/resource/umls/relationQualifier> "finding site of".
 ?r <http://linkedlifedata.com/resource/umls/relationDatasource> ?provenance.
 ?r rdf:type <http://linkedlifedata.com/resource/umls/Relation> .
 ?s1  <http://linkedlifedata.com/resource/umls/relation> ?r.
 ?r <http://linkedlifedata.com/resource/umls/relatedConcept> ?s2.

 # Disease concept properties
 ?s1 rdf:type <http://linkedlifedata.com/resource/semanticnetwork/id/T047>. #Disease or Syndrome
 ?s1 <http://www.w3.org/2008/05/skos-xl#prefLabel> ?s1Atom.
 ?s1Atom <http://www.w3.org/2008/05/skos-xl#literalForm> ?s1Label.
 ?s1Atom skos:note ?s1Note.
 OPTIONAL { ?radLexAnatomy skos:exactMatch ?s2. }

 # Pathology concept properties
 ?s2 rdf:type <http://linkedlifedata.com/resource/semanticnetwork/id/T029> . #Body Location or Region
 ?s2 <http://www.w3.org/2008/05/skos-xl#prefLabel> ?s2Atom.
 ?s2Atom <http://www.w3.org/2008/05/skos-xl#literalForm> ?s2Label.
 ?s2Atom skos:note ?s2Note.
 OPTIONAL { ?radLexPathology skos:exactMatch ?s2. }

 # Filter the non-english terms
 FILTER (regex(?s1Note, "Czech|German|French|Terminos") = false && regex(?s2Note,
"Czech|German|French|Terminos") = false)
}
```

The current query yields 1'878 unique pairs between concepts of type "Disease or syndrome" and "Body Location or Region", where 431 has direct connection to RadLex [14]. The returned result on its own can be used as a trusted source. However, it is interesting to double check statistically what average co-occurence frequency according the SBT semantic annotations and measure its average performance:

```
# Count semantic annotations and concept cooccurence generated by SBT
SELECT (count(?abstract) as ?abstractCount) (count(?title) as ?titleCount)
WHERE {
 {
  ?abstract <http://linkedlifedata.com/resource/sbt/mentions/abstract>
<http://linkedlifedata.com/resource/umls/id/C0006131> .
  ?abstract <http://linkedlifedata.com/resource/sbt/mentions/abstract>
<http://linkedlifedata.com/resource/umls/id/C0027530> .
 }
UNION
 {
  ?title <http://linkedlifedata.com/resource/sbt/mentions/title>
<http://linkedlifedata.com/resource/umls/id/C0006131> .
  ?title <http://linkedlifedata.com/resource/sbt/mentions/title>
<http://linkedlifedata.com/resource/umls/id/C0027530> .
 }
}
```

From the result statistics are excluded all pairs, which do not generate any hits in PubMed abstract. Only 83 combinations between "Disease or syndrome" and "Body Location or Region" deliver more

than one hit, which is suggest an initial negative evaluation for the feasibility of the approach. The low co-occurence between the terms could be explained with the generic nature of the PubMed abstract texts and the very specific concepts that are rarely found in summary descriptions. A more promising source will be to investigate the RadLex dataset, which specially addresses the radiologist needs of less detailed entity modelling. The average execution time is close to 88ms, the maximum and minimum values range between 67ms and 467ms. If the number of concurrent request is increased 5 parallel users to execution time drops only to 108ms, which is more than acceptable performance to implement index time repository checks.

7 Conclusion

This document reports on the first performance and scalability evaluation of the Khresmoi Biomedical Knowledge Server (KS) delivered by D5.2 [1]. The report set metrics to assess the system capabilities, defines an evaluation approach and controlled infrastructure to execute the tests. Chapter 6 demonstrates linear system scalability during the data preprocessing, where the loading speed is not affected by the already processed information. The total time to load, merge all datasets and deliver the Khresmoi KB (over 1 billion RDF statement) is less than 12 hours. The flexibility of the batch loading allows fast incremental replacement of the used datasets, whereas the unmodified graphs are not preprocessed again.

KS and the underlying OWLIM database scale well also with respect to the query performance. Section 6.2 compares the execution time between native SPARQL queries and the Disambiguator service – a service optimized to autocomplete or suggests concepts based on user keyword input. The performance improvement in favour to the Disambiguator service is more than an order of magnitude, which satisfy the very demanding user requirements of virtually immediate server response. The execution speed remains stable even with 100 concurrent user requests and satisfies the use case requirement of 100ms second response.

Lastly, but not least the KB demonstrates solid performance in matching pathologies with anatomical location. This topic is a subject of future investigation and work that will be presented in D5.4 “Report on consistency checking rules for information extraction”.

8 References

[1] Konstantin Pentchev, Vassil Momtchev "D5.2 Large Scale Biomedical Knowledge Server".

[2] OWLIM Documentation, http://owlim.ontotext.com

[3] Steve Harris and Andy Seaborne, "SPARQL 1.1 Query Language", W3C Working Draft 24 July 2012, http://www.w3.org/TR/sparql11-query/

[4] K. Pentchev and V. Momtchev "D5.1 Report on data source integration".

[5] Emmanuel Jamin, Vassil Momtchev, Konstantin Pentchev, revised by Allan Hanbury "D6.3.1: State of the art, Concepts, and specification of the Early Prototype", Section 8.1.1 "Components overview for the Search System".

[6] A. Kiryakov et al. "D5.5.2 Validation goals and metrics for the LarKC platform".

[7] Guo, Y; Pan, Z; and Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. Journal of Web Semantics, 3(2), 2005, pp. 158-182, 2004.

[8] J. Urbani, S. Kotoulas, J. Maassen, F. V. Harmelen and H. Bal, "OWL reasoning with WebPIE: calculating the closure of 100 billion triples," in Proceedings of the 2010 Extended Semantic Web Conference, 2010, vol. 6088, pp. 213-227.

[9] J. Urbani, F. V. Harmelen, S. Schlobach and H. Bal: QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases. International Semantic Web Conference (1) 2011: 730-745

[10] OWLIM Documentation: Benchmark Results at http://www.ontotext.com/owlim/benchmark-results

[11] The Apache Software Foundation, "Lucene 3.6.1 Documentation", http://lucene.apache.org/core/3_6_1/index.html

[12] Semantic Biomedical Tagger 1.2 at http://www.ontotext.com/life-sciences/semantic-biomedical-tagger

[13] The Apache Software Foundation, http://jmeter.apache.org/

[14] Langlotz CP. RadLex: a new method for indexing online educational materials. Radiographics. 26 (6): 1595-7. doi:10.1148/rg.266065168