Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Report on results of second phase scalability and performance evaluation

Deliverable number	*D5.6*
Dissemination level	*Public*
Delivery date	*June 2014*
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.

Table of Contents

Table of Figures

Index of Tables

List of abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma Separated Values
ETL	Extract Transform Load
GAPP	GATE Application
GATE	General Architecture for Text Engineering
HON	Health On the Net
HTML	HyperText Markup Language
IE	Information Extraction
KB	Knowledge Base
KS	Large Scale Biomedical Knowledge Server
NER	Named Entity Recognition
RDF	Resource Description Language
REST	REpresentational State Transfer
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SSD	Solid State Drive
TOS	Talend Open Studio
UMLS	Unified Medical Language System
URI	Universal Resource Identifier
VM	Virtual Machine
WS	Web Service

1 Executive summary

Deliverable D5.6 reports on the performance and scalability results of Khresmoi Biomedical Knowledge Server (KS) [1]. In the previous evaluation [2] of the KS the underlying RDF database – OWLIM [3], custom developed query APIs and use case datasets loading and query answer performance and scalability were measured. This report provides an evaluation of the old services in the context of increased data load and cloud deployment, and of new applications developed as part of T5.4 [4] and T5.5 [5].

The detailed analysis of the offline service for semantically annotating and indexing documents shows good performance and flexibility. It is able to process the entire document set selected for year four in less than three days. In addition, it is designed to scale both vertically and horizontally. Therefore, we conclude that the approach can be used for processing even billions of documents in a multi-node setup.

All web services evaluated show excellent performance. Even under heavy concurrent load of thousand requests per second the response times fulfil the requirements discussed and defined in this document. Slight changes in performance with increasing number of requests is observed, but does not cause the unresponsiveness. Deployment in the cloud does not appear to affect the services performance either positively or negatively. It is recommended to consider horizontal scaling of the services if concurrent load in the ranges of hundred thousand to million requests is expected.

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 final KS. This includes both the previously evaluated services and new ones developed as results of T5.1, T5.4 and T5.5. Our aim is to not only measure the current state of the system, but also to compare it to the previous evaluation. Better performance is expected not only due to improvements to the software, but also due to the deployment in the Khresmoi Cloud. We have formally divided the services in two, based on whether or not they are exposed online.

The group of offline services consist solely of the document annotation and indexing workflow – a Talend job for semantic information extraction (IE) and semantic indexing from crawled pages, Wikipedia and PubMed (referred to as documents) [5]. The goal of the process is to enable semantic search over the evolving set of documents: crawled HTML pages, Wikipedia articles and PubMed abstracts. The evaluation of this service concentrates on performance with regard of the number of documents processed per second, its stability and scalability options. There are several steps involved in the process depicted in Figure 1:

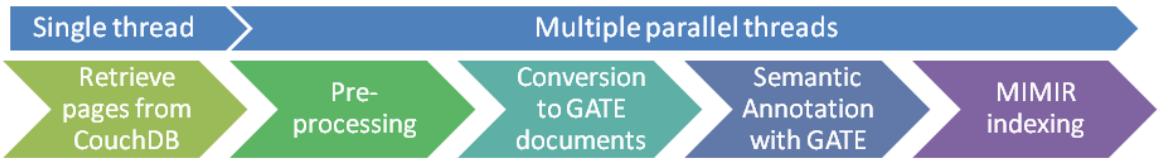

Figure 1: Document annotation and indexing workflow. The figure shows an abstract sequence of the steps performed for each document. First a batch of crawled documents is retrieved from HON CouchDb. Then, preprocessing, e.g. setting encoding and language, is applied, before converting to GATE Documents. Semantic IE is performed on the latter with a pipeline

developed in WP1[7]. Finally, the annotated documents are sent to a MIMIR server for indexing.

The separate steps in the first version of the workflow were sequentially coupled, i.e. a next step could not begin until the previous one is completely finished. In the final version we decoupled different stages to achieve better performance as shown in Section 5.1.

The second group consists of RESTful web-services that expose different information retrieval (IR) methods. There are two such services that were tested in D5.3[2], which we are going to re-evaluate with the current amount of data loaded and cloud deployment:

- SPARQL Endpoint
  - used for querying raw RDF data
- Disambiguator
  - used for disambiguating resources from text input

There is also a set of new services developed that we will evaluate using the same methodology:

- Quick Search
  - basic full-text and resource search
- Co-occurrence Search
  - faceted resource search
- Semantic type-ahead search
  - guided resource search based on type hierarchies and predicate domain and range values

As they can all be evaluated by simulating web request load, the same metrics and infrastructure were used for them, described in Section 3.2 and Section 4.2 respectively. No evaluation of the semantic store OWLIM was performed, as this was tested in D5.3 and no major differences can be reported.

3 Metrics for scalability and performance

In this section we present metrics used to measure the scalability and performance for the different executed tests. Similar to the previous evaluation described in D5.3[2], the aim is to have quantitative measurements of the system's ability to handle large and increasing amounts of data and load for both offline and online services.

3.1 Document annotation and index service

For the document service we have identified the following metrics to quantify its performance:

- T – the total processing time
- N – total processed documents
- P – the average processing speed in [doc/sec]
- ΔP – the change in average speed

The most important parameter is the average processing speed, which will allow us to estimate the required time for future processing. However, the change in average speed can give insights into

bottlenecks and possible improvements with regard to the amount of processed documents and the specific document sets.

3.2 Web services

The performance and scalability metrics for online services are the same as defined in D5.3 [2]. As stated in that report, we are interested not only in the average request response time, but in the corresponding minimum and maximum and variance. Ideally, the services will respond consistently, i.e. the deviation of response times for different requests will be small. Finally, we also need to check the amount of data returned with each response in order to identify bandwidth bottlenecks. The performance metrics are defined as follows:

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

Scalability in the context of web services can be defined as a service's ability to respond to increasing request load in consistent times. Therefore, we need to measure the difference in response times in a series of increasing simultaneous requests. The metrics are defined as follows:

- 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 time series in $G_{0..n}$

In order to calculate these variables with respect to the absolute amount of change, we use the square root of the squared differences:

- $\Delta RP^{i}_{avg} = 1/(n-1)\sum \sqrt{(RP^{i}_{avg}(k) - RP^{i}_{avg}(k-1))^2}$, 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

In addition, we define two response time thresholds for different services based on their perceived usage:

- 100ms - for services that need to react instantaneously
- 1sec - for services that need not react instantaneously, but must not be perceived as unresponsive

The values are based on the research published by Nielses [8].

4 Test infrastructure

4.1 Components for batch process

For measuring the performance and scalability of the document annotation and indexing service, the Talend Performance component was used. It was developed and used for the evaluation of the OWLIM loading workflow in D5.3, section 6.1 [2]. Because of the generic nature of this component – it measures the number of information units that pass through it for a given amount of time – it was used without modification.

4.2 JMeter for Web services

For generating requests to our RESTful web services and simulating concurrent load, we used Apache JMeter – an open-source desktop application for conducting functional and performance tests of web applications [6].

The tool provides a powerful interface to design integration and performance tests, and to monitor the execution times, data transferred and response status. It comes out of the box with components for graphical analysis as well as report generation. Requests inside test plans can be randomized and repeated. Components for authentication are available as well.

Concurrent load is simulated by defining a number of users for the test plan. Each user is executing sampling requests in separate asynchronous thread. Thereby, each user has to perform the entire test plan defined. The user initialization can also be staged so that there is a general increase in the amount of concurrent requests.

All subsequent online query tests presented into the current report are executed with a JMeter test plan, which can be made available on request.

5 Results for performance and scalability

This section presents the results of the performed evaluation.

5.1 Document Indexing and Annotation Workflow

The first version of the Document Indexing and Annotation Workflow was completed in September 2013. With the annotation step estimated as the bottleneck of the process (see Figure 1), it was implemented to process documents on separate threads. The statistics reported in Table 1 were recorded during a full indexing run in the same month on the hon-2 server with 17 cores available. It is evident from the results that almost 11 days were required to process all the relevant resources. While the process was divided in steps based on the crawl date of documents to allow for some flexibility (e.g. recovering after errors), it was still very rigid and cumbersome. The average speed of 11 documents/sec was also much lower than expected from test runs performed while developing the GATE pipeline. Detailed evaluation of the pipeline identified the fetching of documents from CouchDb as the bottleneck. Effectively, new batches of data were retrieved from a single thread only after the previous batch was fully processed. A new version was developed with the retrieving and processing completely decoupled. The full logs of all the runs can be provided on request.

Document set	N [doc]	T [hh:mm:ss]	P [doc/sec]	ΔP [doc/sec]	Version
All	11'211'924	260:28:26	11	0.0039	Sept 2013
Wikipedia	31'703	00:25:17	20	6.5454	May 2014
Pubmed	1'671'633	13:03:17	35	0.1208	May 2014
HTML (en, de, cs)	1'828'766	13:40:19	37	0.1388	May 2014
HTML (fr, es)	1'270'681	17:01:09	20	0.5319	May 2014

Table 1: Measurements for the Document Indexing and Annotation Workflow. The first row is a summarization of a full indexing with the first version of the job. The rest of the table gives detailed statistics about separate document sets processed with the latest version of the workflow.

Rows 2 to 4 in Table 1 present the results from several test runs with the new version based on different document sets. It is evident that a notable improvement in speed of 2 to 4 times was achieved. Thus, the full semantic annotation and indexing could be performed in about 2 days (note that the number of documents to process was also reduced due to changes in classification by HON). However, one can note that there are some differences in performance between the document sets – Wikipedia and French/Spanish HTML pages were processed notably slower. In the first case we must consider that the set consists of relatively few documents and was completed in ~25 minutes total. This time includes the initialization phase, during which Gazetteers are loaded. As mentioned in the paragraph above, the executions were performed on 17 cores. Thereby, for each thread a separate Gazetteer is initialized (with some resource sharing), which is a relatively slow process. From the logs we concluded that the initialization phase requires ~10 minutes to complete, which is about 40% of the total time for processing Wikipedia. As no documents are processed at all during this period, we can conclude that the statistic is skewed and the actual processing speed is *31703*docs*/((25-10))*60+17*sec = 34.57 docs/sec. This is consistent with the measurements for Pubmed and the English, German and Czech pages.

The slow processing speed for the French and Spanish HTML pages can in turn be explained by an interruption of exactly 10 hours due to network outage (between 15:23:31 on 02.05 and 01:23:40 on 03.05). The adjusted processing time is 07h and 01min, or an average speed of 50 docs/sec. The better performance compared to the rest is expected, as the GATE pipeline performs no semantic annotation for French and Spanish pages, but only tokenization and full-text indexing.

In order to assess the scalability of the workflow, we are interested in the change of performance with increasing number of documents processed - ΔP. A high value would indicate instability and possible decrease, which might make the approach unusable for document sets in the billions range. However, the variance in performance is negligibly small for the Pubmed abstracts and HTML pages. Table 1 reports slightly higher values for the Wikipedia pages processing, but we can ignore it due to its small size and long initialization phase. Therefore, we conclude that the amount of documents to be processed is not affecting the performance of the workflow.

Of course, the linear dependency between the number of documents and the time required to process them means that the current setup will take unfeasibly long to process documents in the billions range. Possible solutions to scale up the workflow are to either increase the number of cores on the server (vertical scaling) or to perform indexing on several machines in parallel (horizontal) [9]. The second approach offers more potential, as we have seen that the retrieval speed of documents from CouchDb can be a limiting step. Adding more “processing nodes” can circumvent the single point of retrieval

issue and assuming that CouchDb can efficiently respond to multiple concurrent requests, will yield processing times lower by times the number of nodes. Thereby, document sets from the original collection can be derived by logically sharding them by corpus and/or date and will be indexed in separate MIMIR indices. Multiple nodes are easy to set up, because the workflow requires only JAVA and GATE to be installed on the node and any communication with other services is performed via Web APIs.

5.2 Web services

As mentioned in Section 3.2, we want to test not only the performance of each web service , but also how scalable it is under concurrent load. Therefore, for each service we executed four evaluations – one with a single user and three with increasing number of users (from 10 to 1000). A set of requests to each service are defined and executed in random order multiple times. We report the detailed statistics for the reference single-user evaluation and only the aggregated statistics for the multi-user tests. The KS evaluated is running on a 64-bit Ubuntu VM, with 8 Xeon E5-2620 CPUs (2 GHz), 32GB RAM and SSD disks. Caching of web requests was disabled in the application server and the web application.

5.2.1 SPARQL Endpoint

In this section we present the evaluation results for our SPARQL endpoint. As this is the primary way to accessing the RDF data in our semantic repository it will test primarily the query answering performance of OWLIM [3]. As per recommendation after our previous evaluation we included both simple and complex SPARQL queries in order to get a full understanding of any shortcomings. The queries are selected from real use cases in the Khresmoi project, e.g. for indexing, mapping between dataset etc. The queries are listed in the appendix (Section 8). The results of the performance evaluation are given in Table 2:

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
Radiowiki	100	75	62	236	28.21	71.37	37212
Select all	100	108	64	252	16.00	103.71	53495
RadLex to UMLS mapping	100	131	118	222	12.92	106.01	54775
RadLex labels	100	69	59	202	19.31	59.68	30509
MeSH labels	100	113	106	232	12.98	91.76	47046
TOTAL	**500**	**99**	**59**	**252**	**30.06**	**427.77**	**44607**

Table 2: Performance statistics for SPARQL endpoint for 1 thread

All queries were answered in less than 300ms with average response times around 100ms. This is excellent performance given that the repository contains more than 1.2 billion statements. We must note that the queries, while complex from a semantic point of view, do not perform full-text matching. This was tested in D5.3 and showed worse response times.

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
10 users	5000	185	60	986	56.90	618.59	44607
100 users	5000	200	102	1004	65.90	1532.88	44607
1000 users	5000	228	103	1084	65.68	1776.36	44607

Table 3: SPARQL endpoint performance statistics for 10, 100 and 1000 concurrent threads respectively

The performance of the service did not deteriorate under heavy concurrent load as is evident from Table 3. We can calculate the following values for the change in performance :

- $\Delta RP^{ij}_{avg} = 1/(4-1)\,(\sqrt{(185-99)^2} + \sqrt{(200-185)^2} + \sqrt{(228-200)^2}) =$ **43ms**
- $\Delta RP^{ij}_{min} = 1/(4-1)\,(\sqrt{(60-59)^2} + \sqrt{(102-60)^2} + \sqrt{(103-102)^2}) =$ **14.66ms**
- $\Delta RP^{ij}_{max} = 1/(4-1)\,(\sqrt{(986-252)^2} + \sqrt{(1004-986)^2} + \sqrt{(1084-1004)^2}) =$ **277.33 ms**

Only the maximum response time shows significant change considering that the concurrent load has been increased by 3 orders of magnitude. However, it is interesting that there is a significant increase in response time only between one and 10 threads. Subsequent increase in user count results in only minor deterioration of performance. Hence, we can expect that the service will scale well even beyond our measurements and the average response time will remain below 1 second. Since the results are not expected instantaneously, this is in accordance with the threshold defined in Section 3.2.

5.2.2 Disambiguator

For the Disambiguator service we re-use the sample request from D5.3[2] in order to directly compare the results. Queries in different languages were sampled in order to test for possible inconsistent performance. The results are presented in Table 4:

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
” Diabet” ES	100	80	61	507	61.95	41.25	15341
”Diabete” DE	100	68	61	132	9.77	35.56	13193
”Leukoc” CZ	100	68	58	141	13.26	16.71	6214
”Cancer” EN	100	72	63	178	14.40	37.07	13785
”Klein” DE	100	67	58	112	9.52	20.98	7757

TOTAL	**500**	**71**	**58**	**507**	**30.10**	150.17	11258

Table 4: Disambiguator performance statistics for 1 thread

As in the previous evaluation report there is no significant difference in the response between the different queries. However, there is a notable increase in the response times in comparison with the values in D5.3:

- $RP^{Disambiguator}_{avg}$ = 71ms (was 14 ms)
- $RP^{Disambiguator}_{max}$ = 507ms (was 32 ms)
- $RP^{Disambiguator}_{min}$ = 58ms (was 9 ms)
- $RV^{Disambiguator}$ = 906.01ms (was 6.0025 ms)

The reason for this is increased complexity in the retrieval of results in order to yield more precise suggestions. While previously only one request was made internally to the full-text indexer (Solr), now two are performed. The first is looking only for exact matches of the user input, the second is performing fuzzy string matching. Retrieving and merging these two requests is what caused the increase in response time, as these are expensive operations that operate over the local network. Moreover, while the current VM has more CPUs, their frequency is lower (2GHz compared to 2.5GHz). However, the average response time remains below the defined threshold of 100ms. The improved precision of the suggested results was therefore more beneficial than a slightly delayed response, which is probably not perceivable by human users.

The increased variance is more worrying, as it indicates inconsistent performance. This can be again attributed to the two internal requests performed by the service and network instability, but it can also indicate other problem, e.g. with the server deployment. Because the KS is now in a virtual machine in the Khresmoi Cloud it is possible that resource management by the hypervisor causes additional instability. We need to also acknowledge that significantly more data is transferred with each request. The previous version of the Disambiguator served 3659 Bytes per request compared to 11258 Bytes for the current version. This is more than a double increase and while a low-end bandwidth of 16 Mbit/s will still be able to stream more than 20000 responses it can also explain the increases in the millisecond range.

The changes in performance observed might also have an implication for the scalability of the service. Table 5 gives a summary of the results of the concurrent load evaluations.

Request	**N**	**RP^{i}_{avg}[ms]**	**RP^{i}_{min}[ms]**	**RP^{i}_{max}[ms]**	**$\sqrt{RV^{i}}$[ms]**	**D^{i}[KB/s]**	**D^{i}_{avg} [B]**
10 users	5000	105	57	1403	105.71	353.05	11258
100 users	5000	117	56	1493	140.92	445.07	11259
1000 users	5000	133	57	1306	110.99	449.51	11258

Table 5: Disambiguator performance statistics for 10, 100 and 1000 concurrent threads respectively

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

- $\Delta\mathbf{RP}^{ij}_{avg} = 1/(4-1)\,(\sqrt{(57-58)^2} + \sqrt{(56-57)^2} + \sqrt{(57-56)^2}) =$ **1ms**
- $\Delta\mathbf{RP}^{ij}_{min} = 1/(4-1)\,(\sqrt{(105-71)^2} + \sqrt{(117-105)^2} + \sqrt{(133-117)^2}) =$ **20.33ms**

- ΔRP^{ij}_{max} = 1/(4 -1) ($\sqrt{}$ (1403-507)^2 + $\sqrt{}$ (1493-1403)^2 + $\sqrt{}$ (1306-1493)^2) = **390.66 ms**

It is evident that there is an increase in the maximum and average response times, while the minimum response time remains the same. The response time variance is also increased more than threefold. However, the average response time remains close to the threshold of 100 ms and increases only slightly for concurrent loads increasing by orders of magnitude. Moreover, the maximum response times, while increasing significantly between a single user and multiple users, remain in the range of 1400ms. From these statistics we can conclude that the service itself is capable of handling large numbers of concurrent requests and it comes down to the supporting hardware infrastructure exactly what the performance will be.

5.2.3 Quick Search

The Quick Search service was evaluated similarly to the Disambiguator. However, we do not define a strict threshold of 100 ms for its average response time as it performs a more complex operation and is moreover not required to provide results for every character input of the user. Therefore, response times below 1 second are considered good performance.

We defined 5 sampling requests for different datasets in order to capture possible discrepancies in the performances based on data specifics (composition and size). The performance evaluation results are presented in Table 6:

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
Search ImageClef	100	75	57	494	46.31	6.66	2573
Search UMLS	100	78	57	457	66.41	5.42	2076
Search Drugbank	100	74	59	441	51.46	3.15	1204
Search RadioWiki	100	67	58	128	13.29	3.97	1527
Search Radiology Reports	100	72	58	490	44.66	6.75	2581
TOTAL	**500**	**73**	**57**	**494**	**47.83**	**25.72**	**1992**

Table 6: Quick Search performance statistics for 1 thread

It is evident that the Quick Search service performs very well with average response times below 100 ms and the maximum response time below 500ms. The average response size of 1992 Bytes is also very low and will not impose any limitations with regard to network bandwidth.

From the concurrent load results presented in Table 7 we can also conclude that the service can handle large numbers of users with ease. There is an increase in the average and maximal response times, but the values remain well below one second. This is confirmed by the calculations for the change in response times:

- ΔRP^{ij}_{avg} = 1/(4 -1) ($\sqrt{}$ (107-73)^2 + $\sqrt{}$ (133-107)^2 + $\sqrt{}$ (173-133)^2) = **33.33ms**
- ΔRP^{ij}_{min} = 1/(4 -1) ($\sqrt{}$ (55-57)^2 + $\sqrt{}$ (54-55)^2 + $\sqrt{}$ (67-54)^2) = **5.33ms**
- ΔRP^{ij}_{max} = 1/(4 -1) ($\sqrt{}$ (604-494)^2 + $\sqrt{}$ (551-604)^2 + $\sqrt{}$ (635-551)^2) = **82.33 ms**

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
10 users	5000	107	55	604	88.20	31.46	1992
100 users	5000	133	54	551	113.00	70.01	1992
1000 users	5000	173	67	635	125.08	79.68	1992

Table 7: Quick Search performance statistics for 10, 100 and 1000 concurrent threads respectively

5.2.4 Co-occurrence Search

The co-occurrence search service differs from the quick search service in that it performs queries in specific logical elements of the data(i.e. properties or fields) and calculates occurrence counts for the values in these elements. Therefore, we can assume that it is a more complex operation and the expected times should be higher. Still, we judge that a 1 sec response time is required for a good experience. The performance results are presented in Table 8:

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]
Cooccurrence Radiology Reports	100	195	107	662	99.91	37.28	33512
Cooccurrence Radiology Reports 2	100	146	62	532	72.83	19.96	17950
Cooccurrence RadioWiki	100	68	59	143	14.87	1.65	1471
Cooccurrence ImageClef 2	100	214	116	652	88.41	43.87	39439
Cooccurrence ImageClef	100	250	113	906	147.47	44.22	39560
TOTAL	500	175	59	906	113.93	145.38	26386

Table 8: Co-occurrence Search performance statistics for 1 thread

From the sample data we can again conclude that the service is performing according to the specification. The average response time is higher than for the quick search, but below 200ms and the maximum response time is below the 1sec threshold. It is notable however, that the average response size is an order of magnitude larger with 26386 Bytes. This means that a 16Mbit connection can support only 79 concurrent responses. Table 9 gives a good overview of the performance of the service under concurrent load:

Request	N	RP^i_{avg}[ms]	RP^i_{min}[ms]	RP^i_{max}[ms]	$\sqrt{RV^i}$[ms]	D^i[KB/s]	D^i_{avg} [B]

10 users	5000	193	57	1215	105.52	359.06	26386
100 users	5000	237	57	1727	152.39	888.88	26386
1000 users	5000	249	57	2177	149.86	1036.28	26386

Table 9: Co-occurrence Search performance statistics for 10, 100 and 1000 concurrent threads respectively

Again there is a small increase in the average response times and a larger one in the maximum response times, which goes beyond the 1sec threshold. We can calculate the following values for the change in performance:

- $\Delta RP^{ij}_{avg} = 1/(4-1)\,(\sqrt{(193-175)^2} + \sqrt{(237-193)^2} + \sqrt{(249-237)^2}) =$ **28.33ms**
- $\Delta RP^{ij}_{min} = 1/(4-1)\,(\sqrt{(57-59)^2} + \sqrt{(57-57)^2} + \sqrt{(57-57)^2}) =$ **0.66ms**
- $\Delta RP^{ij}_{max} = 1/(4-1)\,(\sqrt{(1215-906)^2} + \sqrt{(1727-1215)^2} + \sqrt{(2177-1727)^2}) =$ **423.66 ms**

Obviously, the service will perform well on average with increasing concurrent load, but there will be outliers with much larger response times. This could maybe be amended by vertically scaling the service with a higher bandwidth network. However, we expect only very few users to be affected.

5.2.5 Semantic type-ahead search

The semantic type ahead service acts similarly to the disambiguator, but instead of suggesting resources for disambiguation guides the user in the composition of a valid SPARQL query. This imposes additional restrictions on the suggestion in the form of domain and range values for predicates. Four request were defined – one for each operation that the service performs:

- suggest subjects
- suggest predicates
- suggest objects
- assemble SPARQL query and execute

The final query should retrieve all Drugs that are applicable do Diabetes (C0011847) patients. We expect response times below 100ms.

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]
Typeahead Predicate	100	60	56	121	9.11	1.49	536
Typeahead Object	100	155	114	527	54.56	54.37	19306
Typeahead Subject	100	63	57	118	10.35	1.41	499

Typeahead SPARQL	100	61	56	110	6.69	2.85	1007
TOTAL	400	85	56	527	49.37	59.61	5337

Table 10: Semantic Type-ahead Search performance statistics for 1 thread

It is evident from the performance evaluation results in Table 10 that the service handles requests efficiently according to our specification. The average response time is below 100ms and there are only a few outliers with a high maximum response time. However, we should note that the object suggestion query is performing worse than the other operations. This is probably due to having most restrictions placed on it, which makes it computationally more complex. Another possible cause is the large response size – 19306 Bytes – which is an order of magnitude bigger than for the other operations. Finally, similarly to the disambiguator, the object operation executes two queries to the internal full-text index. This might need to be refactored in the future, depending on the scalability evaluation results shown in Table 11:

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]
10 users	4000	82	53	584	50.62	79.26	5337
100 users	4000	85	52	779	55.65	156.57	5337
1000 users	4000	96	52	1269	62.08	170.54	5337

Table 11: Semantic Type-ahead Search performance statistics for 10, 100 and 1000 concurrent threads respectively

Even for a 1000 concurrent users the average response time remains below 100 ms. The maximum response times increase as for other services. We can calculate the following changes in performance:

- $\Delta RP^{ij}_{avg} = 1/(4-1)\,(\sqrt{(82-85)^2} + \sqrt{(85-82)^2} + \sqrt{(96-85)^2}) =$ **5.66ms**
- $\Delta RP^{ij}_{min} = 1/(4-1)\,(\sqrt{(53-56)^2} + \sqrt{(52-53)^2} + \sqrt{(52-52)^2}) =$ **1.33ms**
- $\Delta RP^{ij}_{max} = 1/(4-1)\,(\sqrt{(584-527)^2} + \sqrt{(779-584)^2} + \sqrt{(1269-779)^2}) =$ **247.33 ms**

There is virtually no change in the minimum and average response times. Similar to the Disambiguator service only the maximum values show a significant increase. Handling of these outliers should follow the same recommendations as in Section 5.2.2.

6 Conclusion

In this document we evaluated the performance of both online and offline services developed for the KS. The report set metrics to assess the system capabilities, defines an evaluation approach and controlled infrastructure to execute the tests.

Section 5.1 showed that the Talend component for semantic annotation and indexing do not impede performance. The ETL job generated using these components processes documents limited only by the speed of the source (CouchDb) and the semantic annotation pipeline, which is a complex operation. Still it is possible to update or recreate the entire Khresmoi semantic index in a few days. The workflow is also easy to scale vertically, because of the flexible design of the components and multithread support. Horizontal scaling by running multiple instances of the job on different VMs is also possible if even larger volumes of data need to be processed.

The web services exposed by the KS have been shown in Section 5.2 to have excellent performance with average response times around and below the 100ms range. Especially the SPARQL endpoint showed excellent performance considering the different and complex queries tested. One could argue that the queries tested are not complex enough, as they do not include nested queries and aggregates, but these are the type of requests used in Khresmoi. Therefore, we can conclude that the service fully covers the needs of the project.

However, some of the services' responses under heavy user load take significantly longer. While these are only outliers, it is worth investigating techniques to eliminate these. Possible solutions include hardware and network upgrades, improvements to the application server configuration and caching (which was disabled during the tests).

However, we must also note that the cloud deployment does not improve the performance of the services themselves as is evident from the comparison of the Disambiguator evaluation in Section 5.2.2. The only benefit is that communication with other Khresmoi services will be restricted to the local network. Still, most of the KS services have a small response size, with only the Co-occurrence Search possibly benefiting from this.

7 References

[1] K. Pentchev, V. Momtchev. Report on data source integration. Khresmoi project deliverable D5.1. August 2011

[2] K. Pentchev, V. Momtchev. Scalability and performance evaluation report. Khresmoi project deliverable D5.3. August 2011

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

[4] K. Pentchev, V. Momtchev. D. Markonis, T. Schlegl. Report on consistency checking rules for information extraction. Khresmoi project deliverable D5.4. May 2013

[5] K. Pentchev, V. Momtchev. Sustainable biomedical knowledge infrastructure. Khresmoi project deliverable D5.5. February 2014

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

[7] A. Roberts, J. Petrak, C. Boyer, L. Dolamic, A. Hanbury, M. Dittenbach, J. Gobeil, M. Novak. Prototype and report on semantic indexing and annotation for information retrieval. Khresmoi project deliverable D1.7. February 2014

[8] Jakob Nielsen. Usability Engineering. Morgan Kaufmann Publishers Inc. Chapter 5. ISBN 012-5-184-050. 1993

[9] H. El-Rewini, M. Abd-El-Barr . Advanced Computer Architecture and Parallel Processing. John Wiley & Son. p. 66. ISBN 978-0-471-47839-3. April 2005, Retrieved October 2013.

8 Appendix

RadLex UMLS mapping

```
PREFIX radlex:
<http://bioontology.org/projects/ontologies/radlex/radlexOwlDlComponent#>
SELECT ?radlexEntity ?umlsEntity ?prefLabelEn ?prefLabelSp
WHERE {
   ?radlexEntity a radlex:RID0 ;
   skos:exactMatch ?umlsEntity;
   radlex:Preferred_name ?prefLabelEn .
   GRAPH <http://linkedlifedata.com/resource/umls> {
      ?umlsEntity a skos:Concept
   } .
   ?umlsEntity skos:prefLabel ?prefLabelSp .
   FILTER (lang(?prefLabelSp) = 'es') .
   FILTER (lang(?prefLabelEn) = 'en')
}
```

RadLex labels

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX radlex:
<http://bioontology.org/projects/ontologies/radlex/radlexOwlDlComponent#>

SELECT distinct ?uri ?value WHERE {
    ?uri rdf:type radlex:pathophysiology_metaclass .
    ?uri radlex:Preferred_name ?value .
}
```

MeSH labels

```
PREFIX skos-xl: <http://www.w3.org/2008/05/skos-xl#>
SELECT * WHERE {
    ?s a skos:Concept ;
    skos-xl:prefLabel ?l .
    ?l skos:note ?note .
    filter regex(?note, "german|french|czech|mesh$", "i") .
    ?l skos-xl:literalForm ?literal
}
```

RadLex labels

```
PREFIX radlex:
<http://bioontology.org/projects/ontologies/radlex/radlexOwlDlComponent#>


SELECT distinct ?uri ?value WHERE {
    GRAPH <http://khresmoi.eu/resource/radioWiki> {
        ?uri <http://linkedlifedata.com/resource/lifeskim/mentions> ?value .
    }
    ?value rdf:type radlex:anatomy_metaclass
}
```

Default

```
SELECT * WHERE {
    ?s ?p ?o .
}
```