Parallel Search Through Statistical Semantic Spaces Leveraging Linked Open Data

Alexey Cheptsov

High-Performance Computing Center Stuttgart, Nobelstr. 19,
70569 Stuttgart, Germany
cheptsov@hlrs.de

Abstract. With billions of triples in the Linked Open Data cloud, which continues to grow exponentially, challenging tasks start to emerge related to the exploitation and reasoning of Web data. A considerable amount of work has been done in the area of using Information Retrieval (IR) methods to address these problems. However, although applied models work on the Web scale, they downgrade the semantics contained in an RDF graph by observing each physical resource as a 'bag of words (URIs/literals)'. Distributional statistic methods can address this problem by capturing the structure of the graph more efficiently. However, these methods are computationally expensive. In this paper, we describe the parallelization algorithm of one such method (Random Indexing) based on the Message-Passing Interface technology. Our evaluation results show super linear improvement

Keywords: Statistical Semantics, Random Indexing, Parallelization, High Performance Computing, Message-Passing Interface.

1 Introduction

We live in a big data world, which is already estimated to be of the size of several Zetta (10^{21}) Bytes. However, the most considerable growth has seen the linked (open) data domain. Recent years have seen a tremendous increase of structured data on the Web with public sectors such as UK and USA governments opening their data to public (e.g., the U.S.'s *data.gov* initiative [1]), and encouraging others to build useful applications. At the same time, Linked Open Data (LOD) [2] project continues stimulating creation, publication and interlinking the RDF graphs with those already in the LOD cloud. In March 2009, around 4 billion statements were available in Resource Description Framework (RDF) format [3], while in September 2010 this number increased to 25 billion, and continues to grow every year exponentially. This massive amount of data requires effective exploitation and is now a big challenge not only because of the size but also due to the nature of this data. Firstly, due to the varying methodologies used to generate these RDF graphs there are inconsistencies, incompleteness, but also redundancies. These are partially addressed by approaches for assessing the quality, such as through tracking the provenance [4]. Secondly, even if the quality of the data would be at a high level, exploring and searching through large RDF graphs requires familiarity with the structure, and knowledge of the used

ontology schema. Another challenge is reasoning over these vast amounts of data. The languages used for expressing formal semantics (RDF etc.) use the logic that does not scale to the amount of information and the setting that is required for the Web. The approach suggested by Fensel and van Harmelen [5] is to merge retrieval process and reasoning by means of selection or subsetting: selecting a subset of the RDF graph that is relevant to a query and sufficient for reasoning.

A considerable amount of work has been done in the area of using Information Retrieval (IR) methods for the task of selection and retrieval of RDF triples, and also for searching through them. The primary intention of these approaches is location of the RDF documents relevant to the given keyword and/or a Unified Resource Identifier (URI). These systems are semantic search engines such as Swoogle [6] or Sindice ([7], [8]). However, although these models work on the Web scale, they downgrade the semantics contained in an RDF graph by observing each physical resource as a 'bag of words (URIs/literals)'. More sophisticated IR models can capture the structure more efficiently by modelling meaning similarities between words through computing the distributional similarity over large amount of text. These are called statistical semantics methods and examples include Latent Semantic Analysis [9] and a more modern technique – Random Indexing, which is based on the vector space concept [10]. In order to compute similarities, these methods first generate a semantic space model. Both generating this model, and searching through it (e.g., using cosine similarity), are computationally expensive. The linear feature of searching through the large semantic space model is a huge bottleneck: for the model representing 300 million documents calculating cosine similarity in order to find similar terms can take as long as several hours, which is currently not acceptable for the problem domain specialists.

In this paper, we describe a parallelization approach for the Random Indexing search algorithm, suggested by Sahlgren [10]. We also discuss some techniques that allowed us to reduce the execution time down to seconds on the way to achieving a Web scale. The paper is structured as follows. In Section II, we present the use cases in which this work has been applied. An explicit description about the applied parallelization strategy and the modifications made to the Random Indexing algorithm are presented in Section III. Moreover, we give a thorough evaluation about the algorithm's performance and scalability on a distributed shared-memory system in Section IV. Finally, Section V presents conclusions and discusses main outcomes as well as future work directions.

2 Use Cases

In this section, we briefly discuss two use cases that are taking advantage of the parallelization of the cosine similarity algorithm used by statistical semantics methods, which is the main topic of this paper. Cosine similarity [10] is a measure of similarity between two vectors of n dimensions, which is finding the cosine of the angle between them. If the cosine is zero, the documents represented by vectors are considered dissimilar, while one indicates a high similarity. We present the query expansion use case, which is used to improve the recall when searching, e.g., Linked Life Data (4 billion statements), followed by a subsetting scenario used to reduce the execution time when reasoning over the FactForge repository [11], which contained 2 billion statements at time of performing the experiment.

2.1 Query Expansion

Query expansion is used in Information Retrieval extensively with the aim to expand the document collection that is returned as a result to a query. This method employs several techniques, such as including lemmas and synonyms of the query terms, in order to improve precision and recall. It works by expanding the initial query thus covering larger portion of documents. In this context, finding synonyms is a very important step and one way to achieve this is by employing statistical semantics methods. These methods operate on a set of documents and therefore, we need to lexicalise an RDF graph in a way that will preserve the semantics and "relatedness" of each node with those in its neighbourhood, into an abstraction, which we call a virtual document.

In order to generate virtual documents from an RDF graph, we first select the relevant part of the original graph and subdivide it into a set of potentially overlapping subgraphs. The next step is lexicalisation in order to create virtual documents from these subgraphs. Finally, we generate the semantic index from the virtual documents. The details of how each of these steps is performed significantly influences the final vector space model. For example, in the selection and subdivision step, all or just a part of the ontology could be selected; the subgraphs could be individual triples, or RDF molecules (the set of triples sharing a specific subject node), or more complex/bigger subgraphs. In the lexicalisation step, the URIs, blank nodes, and literals from an RDF subgraph are converted to a sequence of terms. When generating the semantic index, different strategies for creating tokens and performing normalisation have to be applied to typed literals, string literals with language tags, and URIs.

Once the semantic index has been generated, it can be used to find similarities between URIs and literals. We use the ranked list of similar terms for URIs/literals that occur in certain kinds of SPARQL queries [5] to make the query more generic and also return results for entities that are semantically related to those used in the original query. Therefore, the application of query expansion through the use of statistical semantics method is feasible for those SPARQL queries that are not returning all relevant hits. In other words, query expansion here is aimed to improve recall, which is done by adding terms that are similar to the given ones in the original query.

2.2 Subsetting

For reasoning at web-scale, subsetting becomes a key, because most well-known reasoning algorithms can only operate on sets several orders of magnitude smaller than the Web. Getting subsetting algorithms to work is then of capital importance. There is evidence that by sticking to smaller datasets, computer and cognitive scientists may be optimizing the wrong type of models. Basically, there is no warranty that the proven best performing model on thousands of entities is also the best performing model when datasets are four orders of magnitude larger [12].

3 Basics of Parallel Random Indexing Algorithm

Random Indexing and other similar algorithms can be broken down into two steps:

- generating a semantic index (vectors), and
- searching the semantic index.

Both parts are quite computationally expensive, however, the first part is a one-off step, which does not have to be repeated and the semantic index can be updated to follow changes in the documents if they happen. The second step, however, affects the end user, and therefore is a huge bottleneck for real-time applications. Hence, our focus is optimisation of the search part of the Random Indexing algorithm. Usually, search is performed over all vectors in the semantic index. Thereby the vectors are analysed independently of each other, i.e., in the arbitrary order.

This basically means that the search can be efficiently improved, when performed on several computing nodes in parallel instead of the "vector-by-vector" (i.e., sequential computation) processing in the current realisation. Practically, the whole vector space domain is decomposed into sub-domains each of which is processed in a separate block/program instance on a different machine. The division of the vectors between the blocks is defined by the domain decomposition [13] (Figure 1). Depending on the realisation, a synchronisation is required among the blocks, e.g., to collect the partial outputs of each block and produce the final result. Generally, the synchronisation step is expensive, and much attention should be paid to the correct implementation of the synchronisation in order to ensure the minimum overhead. In the next section, we describe the major parallelization strategies, enabling the full utilisation of multiple computing nodes as well as the optimal synchronisation between the distributed tasks.

Although a simple multi-threading approach would be extremely efficient in terms of the performance and easy in terms of the implementation efforts [14], it is not sufficient for achieving the Web-scale due to the limited number of CPU cores/nodes interconnected by a shared-memory bus in the currently available computing architectures (current shared-memory architectures offer a maximum of 8 to 16 interconnected cores).

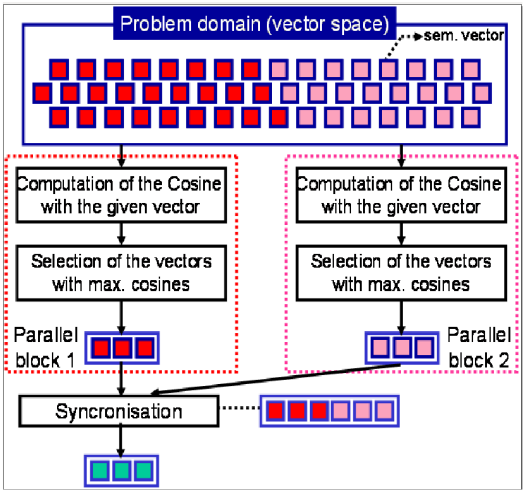

Fig. 1. Domain decomposition based parallelisation of the Random Indexing algorithm

Considering the arguments discussed above, a distributed-memory parallelisation strategy is needed for the implementation of Big Data scenarios. There are several parallelization strategies, differentiating in ways the synchronisation between the processes is implemented. The most promising for the Semantic Web in terms of performance gains are however the Message-Passing Interface (MPI) [15] and MapReduce [16].

MPI is a wide-spread implementation standard for parallel applications, implemented in many programming languages, including Java. As the name suggests, the MPI processes communicate by means of the messages transmitted between two (a so called "point-to-point" communication) or among many (involving several or even all processes, i.e., a collective communication) compute nodes. Normally, one process is executed on a single computing node (however, the MPI standard does not limit the number of processes on one node). If any process needs to send/receive data to/from other processes, it calls a corresponding MPI function. Both point-to-point and collective communications available for MPI processes are documented in the MPI standard [15].

MapReduce is another popular framework for processing big datasets on certain kinds of distributable problems, originally introduced by Google [16] and currently followed by Yahoo in its Hadoop implementation. MapReduce is a promising parallelisation model for data centric applications. However it is quite restrictive with regard to the range of applications that it can be applied to. In this publication, we are focusing on practical aspect of applying the MPI-based distributed memory parallelization for the Random Indexing search algorithm. Due to the algorithmic complexity of splitting the execution workflow according to the map and reduction operation, the MapReduce-based approach [16] will be presented in a separate publication.

4 Implementation With The Message-Passing Interface and Evaluation

4.1 Parallelisation of Airhead Search

Airhead is an open source implementation of Random Indexing in the S-Space package by University of California [17]. Parallelization of the search operation in Airhead was performed by applying the domain decomposition to the semantic vector space, whereby number of domains corresponds to the number of computing nodes the application is running on. Thus, each process performs computation only on a part (sub-domain) of the vector space of size (m/n), where m is the size (dimensionality) of the vector space, and n is the number of processes (and sub-domains, accordingly). The boundary elements of the vector space to be computed by each process are calculated dynamically based on the process rank and the total number of processes provided by MPI, as demonstrated in Figure 2.

```
int num_domains = total_proc_num;
int vector_space_size = VS.size();
int sub_domain_size = vector_space_size / num_domains;

int domain_bound_max = sub_domain_size * (my_rank+1);
int domain_bound_min = sub_domain_size * (my_rank);

// main computation cycle
for (int i=domain_bound_min; i<domain_bound_max; i++) {
... // each process operates only on a sub-domain:
            // VS[domain_bound_min; domain_bound_max]
}
```

Fig. 2. Specification of sub-domains. Each process calculates its respective sub-domain of the vector space based on its Rank and the number of processes in the group.

4.2 Performance Evaluation on Cluster

For the evaluation, a testbed based on the BW-Grid [18] cluster (Intel Xeon CPU architecture, 2 Quad-Core CPUs and 16 GB RAM per computing node), provided by the High Performance Computing Center Stuttgart, was used. Configuration of 1, 2, 4, 8, and 16 computing nodes were benchmarked to evaluate the scalability of the developed algorithms on the target architecture. In our tests, we mainly considered two different datasets coming from well-known semantic repositories (see test sets' parameters in Table 1):

- Linked Life Data (LLD) repository: a large integrated repository, which contains over 4 billion RDF statements from various sources covering the biomedical domain. We investigated two subsets of the LLD that contain major terms (for pharmacological scenarios) and relations between them.
- FactForge repository: contains schemata and ontologies from DBPedia, lingvoj, the CIA Factbook, Wordnet, Geonames, Freebase and musicbrainz. After full materialisation, it contains 404 million resources. We used the DBpedia/Wikipedia section of this space (we will refer to it as Wikipedia from now on, since they are parallel and have the same number of concepts at around 4 M). After filtering out redundant concepts, we kept only 1M documents. After some parameter exploration, we settled on n=1000. That is, the random vectors are 1000-dimensional.

Table 1. Benchmark datasets.

Dataset	Nr. of documents	Nr. of terms	Size on disk	Description
LLD1	0.064 M	0.42 M	0.082 GB	Subset of LLD
LLD2	0.5 M	1.7 M	0.65 GB	Subset of LLD
Wiki term space (Wiki1)	1.0 M	0.3 M	1.6 GB	Subset of Wikipedia
Wiki document space (Wiki2)	1.0 M	0.3 M	16 GB	Subset of Wikipedia

As the first step, we investigated the scalability and stability of the parallelised algorithm on the cluster, increasing the number of nodes involved in the computation, for different problem (dataset) sizes. The time for loading the datasets from the disk (i.e., the whole vector file has to be loaded into the memory of each node), the actual search operation as well as the overhead of the inter-node communication was in the focus of our measurements (Table 2).

The evaluation reveals that our concern about the large impact for loading file from the disk time on the overall application performance was feasible. For all investigated use cases, the load time was considerably higher than the search time. Providing the bad scale of the load operation, the maximum speed-up achieved on 16 clusters computing nodes was only 1.29. Moreover, the experiments with the largest available semantic space (Wiki2) were impossible to be conducted due to exceeding the available RAM on the test bed. For the first test case, although the parallelization has been properly implemented, its usability for datasets with the large number of referenced documents and small amount of dependencies has not been proved. This is because the amount of computation for the search operation was relatively small as compared with the total execution time. Nevertheless, the second variant based on the split of datasets (Figure 1), demonstrated its value in terms of both performance and scalability for the diverse problem sizes (Table 3). The LLD1 set has been excluded because of its small size.

Despite the increasing communication overhead (caused by MPI operations), which is due to more complex communication pattern (as described in the previous publication [21]), the evaluation reveals a significant performance improvement for both load and search operations (see Figure 3). Generally, the use cases taking advantage of the dataset fragmentation show an improvement in time of approximately 85% (i.e., and average speed-up of approx. 7.0 has been achieved) over the non-parallel realisation. This clearly shows that our parallelization technique can be used to benefit Random Indexing applications significantly. Moreover, the technique facilitates applying Random Indexing for the datasets that have not been analysed before due to the limitations of non-parallel test beds.

Table 2. Performance characteristics grouped by dataset and number of computing nodes.

Dataset	Nr. of nodes	Time (s)				Speed-up
		Load	Search	MPI	Total	
LLD1	1	2.0	0.8	0.0	3.4	1.0
	2		0.6	0.025	3.1	1.1
	4		0.5	0.027	3.0	1.13
	8		0.42	0.031	3.0	1.13
	16		0.3	0.034	2.9	1.17
LLD2	1	14.7	4.7	0.0	21.0	1.0
	2		2.8	0.025	18.0	1.16
	4		1.7	0.027	17.0	1.21
	8		1.2	0.031	16.4	1.28
	16		1.0	0.034	16.3	1.29
Wiki1	1	28.5	1.5	0.0	30.5	1.0
	2		1.4	0.025	30.1	1.01
	4		0.84	0.027	30.0	1.02
	8		0.7	0.031	29.7	1.03
	16		0.6	0.034	29.5	1.03
Wiki2	1	Tests could not be conducted due to memory (RAM) limitation on the computing nodes				
	2					
	4					
	8					
	16					

Table 3. Performance characteristics for fragmented datasets, the number of fragments corresponds to the number of computing nodes.

Dataset	Nr. of nodes	Time (s)				Speed-up
		Load	Search	MPI	Total	
LLD2	1	14.7	4.7	0.0	21.0	1.0
	2	7.7	2.5	0.028	10.8	2.0
	4	4.1	1.4	0.20	6.2	3.4
	8	2.3	0.9	0.22	3.8	5.5
	16	1.6	0.65	0.375	2.8	7.5
Wiki1	1	28.5	1.5	0.0	30.5	1.0
	2	15.4	0.99	0.027	16.9	1.8
	4	7.8	0.76	0.039	9.1	3.4
	8	4.4	0.6	0.42	5.5	5.6
	16	2.7	0.54	0.64	3.9	7.8
Wiki2	1	n.a.				
	2	81.0	4.8	0.35	89.9	1.0
	4	67.0	2.7	0.28	71.0	1.25
	8	33.3	1.5	0.22	35.0	2.5
	16	16.8	0.9	0.20	18.4	4.8

4.3 Discussion and Future Directions

As described in the previous section, the performance of the complex search algorithm greatly benefits from the "correct" implementation of the corresponding parallelization paradigm. Correct, here, does not solely mean that MPI has been successfully applied to the Random Indexing search algorithm in order to enable usage of large shared-memory systems, but rather that the algorithm itself has been modified in order to obtain a high performance and scalability - the concept of domain decomposition [13] has been applied to the algorithm to allow the processing of large vector space files (>= 16 GB) and (2) to obtain scalable computation through processing (i.e., the search and in particular the load operation) of smaller subsets of the vector space file concurrently, i.e., distributing the processing to multiple nodes. However, there are many other factors, which influence the overall performance of a parallel application.

Developers can also tune their applications at runtime by using advanced settings for the Java Virtual Machine (JVM) [19]. For this reason, we have also experimented with different settings for the JVM during our tests. We have performed 30 runs of the parallel Airhead search using 6 different JVM settings (each setting has been repeated 5 times) to estimate the optimal configuration for our machines. Due to the fact that all our machines are equipped with equivalent hardware and software, the explicit tests were solely carried out on one particular node with the assumption that the settings are optimal for all other nodes within the cluster, too. A summary of our test runs and the speed-ups achieved is provided in the Table 4.

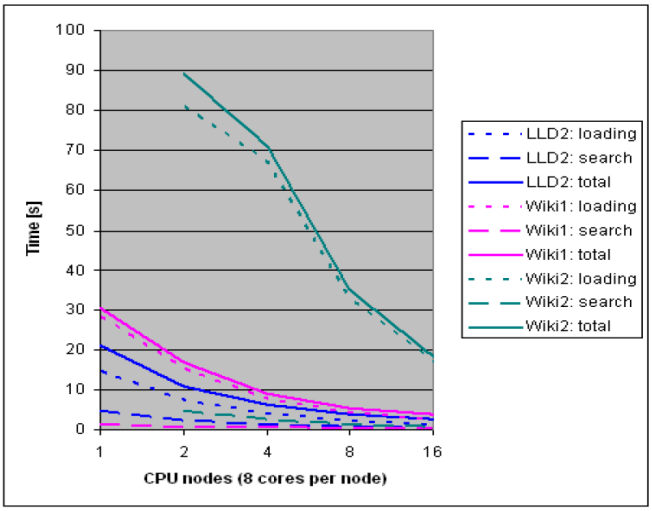

Fig. 3. Performance results for decomposed datasets.

Table 4. Performance results for parallel Airhead search algorithm with varying JVM setting on Wiki1 dataset.

JVM options	Time (s)			Speed-up
	Load	Search	Total	
-Xms4000M -Xms4000M	47.0	2.2	49.2	1.0
-Xms8000M -Xms8000M	43.0	2.1	45.1	1.09
-Xms12000M -Xms12000M	37.0	1.8	38,8	1.27
-Xms12000M -Xms12000M -XX:+AggressiveOpts	30.3	1.6	31.9	1.54
-Xms12800M -Xms12800M -XX:+AggressiveOpts -XX:+UseParallelGC -XX:ParallelGCThreads=16	29.0	1.5	30.5	1.61
-Xms12800M -Xms12800M -XX:+AggressiveOpts -XX:+UseParallelGC -XX:ParallelGCThreads=16 -XX:MaxPermSize=256M -Xmn5120M	28.5	1.5	30.0	1.64

As shown in Table 4, a properly configured JVM significantly improves the overall performance of an application. In our scenario, we were able to optimize the performance of the parallelized application for approximately 40% using the proper JVM settings. Based on these tests, we were also able to determine the best suited JVM configuration for our environment as well as learned more about the optimal values for individual JVM parameters(e.g., the total amount of heap size, the number of threads used for garbage collection, etc.), which can be used for other Java applications as well.

An alternative promising approach is suggested by the JUNIPER (”Java platform for hIgh PErformance and Realtime large scale data management”) project [20]. JUNIPER is an EU-FP7 project that aims to establish a development platform for new-generation data-demanding applications. The JUNIPER approach is to exploit synergies between all major parallelization technologies (such as MPI, MapReduce, COMPSs, etc.) and elaborate new paradigms in data centric parallel processing that will balance flexibility and performance of data processing applications in heterogeneous computing architectures. A possibility to combine diverse parallelization technology within a single application, as offered by JUNIPER, would also be of a huge advantage for Random Indexing algorithms, e.g., to implement the semantic space generation with MapReduce and the search with MPI. In our following research, we are going to investigate the benefits of this “heterogeneous” approach for the Airhead package.

5 Conclusion

This paper presented an evaluation of our approaches to parallelize the Airhead library for Random Indexing, which can be used to significantly improve information retrieval methods, in particular those that use the cosine similarity for searching a large vector space model. We use an effective parallel programming paradigm, namely MPI, to exploit parallelism for the RI algorithm in order to take advantage of large-scale distributed shared-memory systems and thus to improve its performance. We evaluated the parallelized algorithm on different hardware and software configurations (i.e., we varied the amount of computational nodes as well as the input datasets) with promising results. The algorithm improves performance in all of the presented experiments. However, if each process (i.e., node) has to load the full dataset at once, the overall speed-up is relatively small and the algorithm does not scale very well while increasing the number of machines. For this reason, we implemented a way to split the input dataset into smaller junks, which can be independently and concurrently processed by each node. This feature significantly decreased the processing time of the load operation and thus improved the overall performance. Moreover, we are now able to process datasets with billions of statements because we are not directly limited by the system’s memory anymore. In addition, we experimented with different Java Virtual Machine settings in order to optimize and fine-tune the application for the given runtime environment. Most importantly, these results suggest that we need both parallel algorithms and Java Virtual Machine optimizations to effectively utilize machines (not necessarily parallel systems but any common personal computer) for our Semantic Web applications. Finally, we demonstrated the effectiveness of the parallelized algorithm and its usage and benefits within an interesting for biomedical domain application scenario. In the future, we will investigate further possibilities to optimize our code (e.g., using a different MPI implementation for Java) as well as compare the actual MPI-based parallelization with the MapReduce implementation. In particular, methods to cross-fertilize the advantages of diverse programming models in a common application workflow will be explored. Data modelling techniques, investigated by the JUNIPER platform [22], will be the major technology to enable such across-fertilization of the

parallelization technologies. The new technologies that JUNIPER works out will be applied to the most challenging Big Data domains, in particular to Semantic Web.

5 Acknowledgment

This research has been supported in part by the EU-FP7 projects JUNIPER and DreamCloud.

References

1. U.S.'s data.gov intiative website. [Online]. http://www.data.gov/. [retrieved: April, 2013].
2. Linked Data project website. [Online]. http://linkeddata.org. [retrieved: April, 2013].
3. C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far", International Journal on Semantic Web and Information Systems (IJSWIS), 5(3), 2009, pp. 1-22.
4. O. Hartig and J. Zhao, "Using web data provenance for quality assessment", in Int. Workshop on Semantic Web and Provenance Management, Washington D.C., USA, 2009, pp. 29-34.
5. D. Fensel and F. van Harmelen, "Unifying reasoning and search to web scale", IEEE Internet Computing, vol. 11(2), 2007, pp. 95-96.
6. L. Ding et al., "Swoogle: a search and metadata engine for the semantic web", in Proc. the thirteenth ACM international conference on Information and knowledge management CIKM '04, New York, NY, USA, 2004, pp. 652-659.
7. G. Tummarello, R. Delbru, and E. Oren, "Sindice.com: Weaving the open linked data", in Proc. the 6th International Semantic Web Conference, Busan, Korea, 2007, pp. 552-565.
8. E. Oren et al., "Sindice.com: A document-oriented lookup index for open linked data", International Journal of Metadata, Semantics and Ontologies, vol. 3, 2008, pp. 37-52.
9. T. K. Landauer, P. W. Foltz, and D. Laham, "Introduction to latent semantic analysis", Discourse Processes, vol. 25, 1998, pp. 259-284.
10. M. Sahlgren, "An introduction to random indexing", in Proc. Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005, 2005, pp. 1-8.
11. Fact Forge semantic repository website. [Online]. http://factforge.net/. [retrieved: April, 2013].
12. A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data", IEEE Intelligent Systems, vol. 24, 2009, pp. 8-12.
13. T. F. Chan and T. P. Mathew, "Domain decomposition algorithms", Acta Numerica, vol. 3, 1994, pp. 61-143.
14. S. Akhter and J. Roberts, "Multi-core programming: Increasing performance through software multi-threading", Intel Press, Santa Clara, Tech. Rep., 2006.
15. W. Gropp and A. S. E. Lusk, Eds., Using MPI: Portable Parallel Programming with the Message-Passing Interface. Cambridge: MIT Press, 1994.
16. R. Lammel, "Google's mapreduce programming model - revisited", Science of Computer Programming, vol. 70,1, 2008, pp. 1-30.
17. D. Jurgens, "The S-Space package: An open source package for word space models", in Proc. the ACL 2010 System Demonstrations, 2010, pp. 30–35.
18. High Performance Computing Center Stuttgart's BW-Grid cluster description. [Online]. https://wickie.hlrs.de/dgrid/index.php/Hardware. [retrieved: April, 2013].
19. J. Shirazi, Ed., Java Performance Tuning. Sebastopol: O'Reilly & Associates, Inc, 2002.

20. Juniper project website. [Online]. Available: http://juniperproject.org. [retrieved: April, 2013].
21. A. Cheptsov and M. Assel, “Distributed Parallelization of Semantic Web Java Applications by Means of the Message-Passing Interface”, M. Resch et al. (eds.), High Performance Computing on Vector Systems 2011, Springer Verlag Berlin Hedelberg 2012, pp. 51-64.
22. A. Cheptsov and B. Koller, “JUNIPER takes aim at Big Data”, inSiDE - Journal of Innovatives Supercomputing in Deutschland, vol. 11, No. 1, Spring 2011, pp. 68-69.