MPI Realization of High Performance Search for Querying Large RDF Graphs using Statistical Semantics

Matthias Assel[1], Alexey Cheptsov[1], Blasius Czink[1],
Danica Damljanovic[2],
Jose Quesada[3]

[1] HLRS - High Performance Computing Center Stuttgart, University of Stuttgart, Nobelstrasse 19, 70569 Stuttgart, Germany

[2] GATE team, Natural Language Processing Group, Department of Computer Science, University of Sheffield, Regent Court 211 Portobello, S1 4DP, Sheffield, UK

[3] Center for Adaptive Behavior and Cognition, Max Planck Institute for Human Development, Lentzeallee 94, 14195 Berlin, Germany

{cheptsov, assel, czink}@hlrs.de, d.damljanovic@dcs.shef.ac.uk
quesada@gmail.com

Abstract. With billions of triples in the Linked Open Data cloud, which continues to grow exponentially, very challenging tasks begin to emerge related to the exploitation of large-scale reasoning. A considerable amount of work has been done in the area of using Information Retrieval methods to address these problems. However, although applied models work on 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 continually confronting with efficiency and scalability problems on serial computing architectures due to their computational complexity. In this paper, we describe a parallelization algorithm of one such method (Random Indexing) based on the Message-Passing Interface (MPI), that enables efficient utilization of high performance parallel computers. Our evaluation results show significant performance improvement.

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

1 Introduction

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[1], and encouraging others to build useful applications. At the same time, Linked Open Data

[1] http://data.gov.uk and www.data.gov

(LOD) project[2] continues stimulating creation, publication and interlinking the RDF graphs with those already in the LOD cloud [1]. In March 2009, around 4 billion statements were available while in September 2010 this number increased to 25 billion RDF triples, and still continues to grow.

This massive amount of data requires effective exploitation, which 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 they face inconsistencies, incompleteness, but also redundancies. These are partially addressed by approaches for assessing the quality such as through tracking the provenance [2]. 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 exact URIs.

Traditionally, RDF spaces are being searched using an RDF query language such as SeRQL [3] or SPARQL [4]. These languages allow the formulation of fine-grained queries by their ability to match whole graphs and to create complex conditions on the variables to be bound in the query. This level of complexity and flexibility is very useful in many situations, especially when the query is created automatically in the context of an application. However, for end-users who want to explore the knowledge represented in an RDF store, this level of detail is often more of a hindrance: querying the repository is not possible without a detailed knowledge of its structure and the names and semantics of all the properties and classes involved. Another challenge is *reasoning* over this vast amount of data. The languages used for expressing formal semantics (e.g. OWL) use logic, which does not scale to the amount of information and the setting that is required for the Web. In that aspect, the approach suggested by Fensel and van Harmelen in [5] is to merge retrieval process and reasoning by the means of *selection* or *subsetting*: selecting a subset of the RDF graph which 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 an URI. These systems are semantic search engines such as Swoogle [6] or Sindice [7]. They collect the Semantic Web resources from the Web and then index the keywords and URIs against the RDF documents containing those keywords and URIs, using the inverted index scheme [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 Random Indexing [10]. In order to compute similarities, these methods first generate a semantic space model. Both generating the model and searching through it (e.g. using cosine similarity) are quite computationally expensive. Hence these methods have not been used with enormous corpora such as hundreds of millions of documents, which

[2] http://linkeddata.org/

is the case we aim to address in our work. Moreover, from the end-users point of view, the linear increase of the search time through the large semantic space model is a large bottleneck.

In this paper we describe a parallelization approach for the Random Indexing search algorithm for calculating semantic similarities based on the cosine function, and how we reduce this time on the way to achieving Web scale. This is of a significant contribution not only to the Semantic Web community, but also to the Information Retrieval community, due to the size of our corpus, which is considerably larger than what has previously been processed in IR. On the other hand, this work is of great interest for High Performance Computing community, for which Semantic Web has already attracted much attention due to the challenging computationally intensive tasks. A number of European initiatives strive to achieve the goal of increasing the scalability of the Semantic Web algorithms by introducing parallel computing, such as the Large Knowledge Collider[3] project [21]. However, due to the lack of parallelised algorithms developed for Semantic Web, the current HPC support (except some promising examples, refer to Section 5) in this domain is still rather rudimentary.

This paper introduces the results of our recent research, which strives to close those identified gaps. The paper is structured as follows. Section 2 introduces the pilot use case our research was concentrated on. Section 3 presents basics of the distributed parallelization, followed by the detailed description of the parallelization strategy elaborated for the pilot use case. Section 4 concentrates on the performance evaluation of the parallel realization. Section 5 discusses some related works. Section 6 presents directions for further work.

2 Use Case

One of the newest advances in semantic statistics, which enjoys the global data stores for the information retrieval, is Random Indexing. Random Indexing is a novel approach for word space modelling. The word space conception is founded on the distributional hypothesis [10], according to which the two words that tend to constantly co-occur in several contexts have the similar meaning. Typically, a context refers to a multi-word segment, i.e. document. In the context of the data repository, that comprises a set of contexts C^m of size m, which are built upon a set of words X^n occurring in those contexts (obviously, $m \leq n$), it is possible, for each word x, belonging to the word set X^n, to build a so-called context vector (1), whose elements are defined by means of the co-occurrence function f_q:

$$\forall x \in X^n, \exists v = [f_q(x, c_j)] \quad (1)$$

where f_q is a co-occurrence function between the word x and each of the contexts, m is a total number of the contexts, n is a total number of the words in all contexts $c_j \in C^m$, $j = \overline{1..m}$.

[3] http://www.larkc.eu

The co-occurrence function is in the simplest case a frequency counter, specifying how many times the word occurs in the corresponding context, normalized and weighted in order to reduce the effects of high frequency words, or compensate for differences in document sizes. Therefore, the vector (1) represents the context in which the word occurs in the document collection. In the scope of the document base, the full word co-occurrence characteristic is collected by combining the context vectors for each words resulting in the vector space – a two-dimensional co-occurrence matrix of size $m_x n$.

The vector space represents the distributional profile of the words in relation to the considered contexts/documents. The main methodical value of this profile is that it enables calculation of the semantic similarity between the words in scope of the document collection (text corpus), based on the cosine similarity of the given words' context vectors. Cosine similarity is a measure of similarity between the two vectors of *m* dimensions, equals to the cosine of the angle between them.

This context vector property has found a wide application in Semantic Web applications. Prominent examples of algorithms based on semantic spaces are query expansion (e.g. [20]) and subsetting (e.g. [22]). Query expansion is extensively used in IR with the aim to expand the document collection (see Fig. 1a), which is returned as a result to a query thus covering the larger portion of the documents. Subsetting (also known as selection), on the contrary, deprecates the unnecessary items from a data set (see Fig. 1b), in order to achieve faster processing. Hence, the query expansion and subsetting applications are complementary, and are used to change properties of the query process to best adapt it to the search needs.

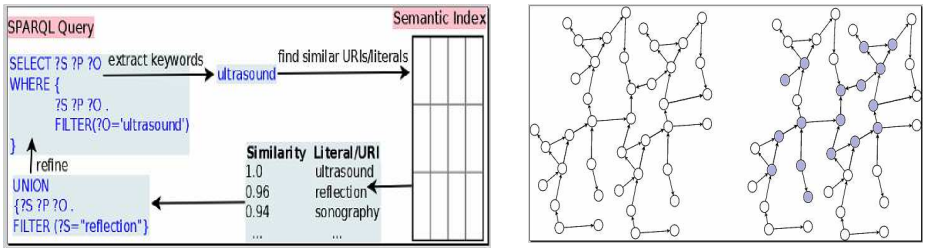


Fig. 1. Examples utilizing Random Indexing: a) Query expansion, b) RDF subsetting.

The pilot application considered in our research is the Airhead Semantic Spaces library [11]. Airhead is an open-source implementation of Random Indexing in the Java language, which has proved its usability for a number of practical tasks, in particular when applied for query refinement or RDF subsetting using Linked Life Data[4] (LLD) or Wikipedia. Nevertheless, the very high dimensionality of the context vectors, which is a direct function of the analyzed data size, results in the hardware requirements going far beyond the capabilities of the current desktop computers. The latter fact makes traditional serial computing architectures increasingly ineffective when complexly addressing those large data amounts, as enabled by LLD.

[4] http://linkedlifedata.com

3 Parallelization strategy

3.1 Basics of Parallelization

Parallelization is a mechanism that allows the single processor's workload to be decomposed and distributed among other compute nodes of the parallel system, thus reducing the total time of the algorithm execution. This basically suggests identification of the concurrent regions of the application work- and dataflow, with further mapping them to the independent processor units of a parallel system, aiming at achieving the positive performance impact for the application.

In software engineering, the most successful parallelization strategies are based on a non-overlapping domain decomposition of data structures as well as code instructions (Fig. 2).

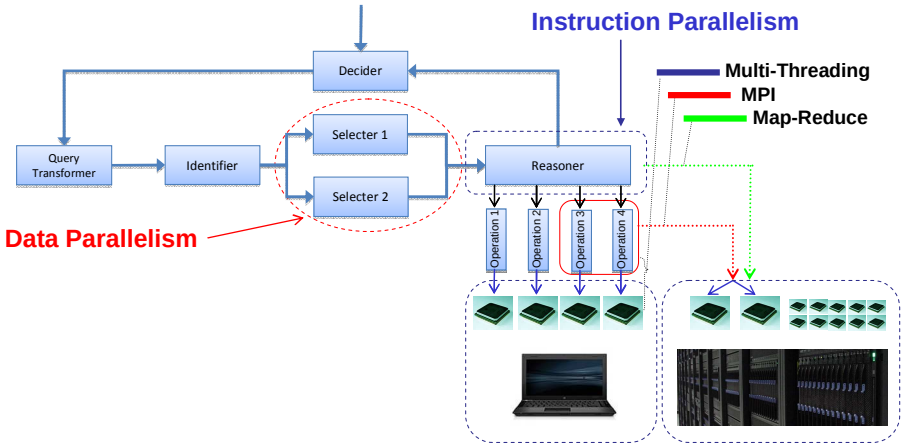


Fig. 2. Examples of parallel data and code constructions in a simple reasoning application workflow, as considered in the LarKC project.

There are several well-established technologies for implementation of parallelization in software applications, which can be subdivided into two big classes:

- thread-based, for systems built on top of tightly coupled parallel CPUs (e.g. Multithreading, OpenMP);
- process-based, for systems built on top of the loosely coupled parallel CPUs (e.g. MPI, MapReduce).

Whereas the thread-based strategies, such as *Multi-Threading* [12], are considered to be very efficient and relatively easy to implement in the application code, they do not allow the application to achieve high performance speed up due to resource limitation of the existing shared-memory compute architecture (the total number of CPU cores provided by such system is usually not higher than 8). On the contrary, the process-based strategies, such as *MapReduce* [13] or *Message-Passing Interface (MPI)* [14], enable distributed compute architectures for application execution, in particular clusters, HPC, or Grid systems. In the following, we concentrate basically on the process-based approaches, in particular MPI.

3.2 Distributed-memory Parallelization Frameworks

Among the most widely utilized and sustainable parallelization approaches, the following two are of special interest for implementation of data-demanding parallel applications:

- MapReduce
- Message-Passing Interface

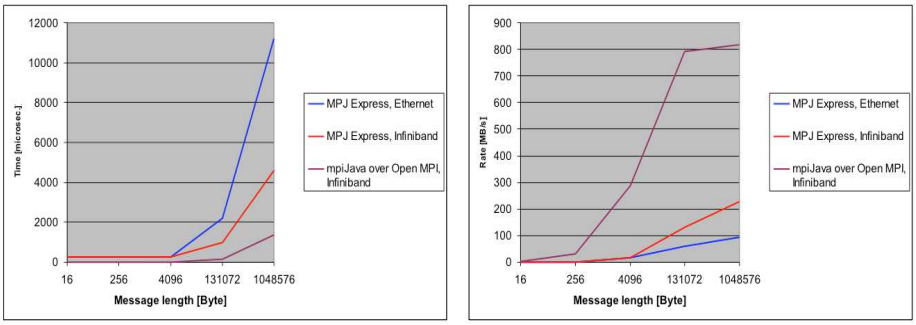
MapReduce is a software framework for distributed computing on large data sets on clusters of workstations. Although MapReduce has proved it efficiency for processing large datasets on certain kinds of distributable problems, in particular for Semantic Web tasks, this technique requires considerable re-thinking of the application algorithms in order to confirm to the Map and the Reduce steps [13]. In the consequence, quite a big effort is required to re-implement existing large applications with the MapReduce framework.

Contrary to MapReduce, MPI does not require such considerable changes in the application code because it is simply a utility library supporting information exchange among the parallel application instances. Moreover MPI is a language-independent standard, so can be re-used at almost any parallel system. For the above-mentioned reasons, we chose exactly MPI for the parallel implementation of the considered algorithm.

As the acronym suggests, MPI is a process-based technique, whereby processes communicate by means of messages transmitted between (a so-called "point-to-point" communication) or among (involving several or even all processes, a so called "collective" communication) the nodes. Normally, one process is executed on a single computing node. If any of the processes needs to send/receive data to/from other processes, it should call a corresponding MPI communication function. Both point-to-point and collective communications available for MPI processes are documented in the MPI standard [15].

There have been several initiatives striving to provide support for Java in HPC environments. One of the most successful MPI implementations is considered to be mpiJava[5] [16], which came out of the HPJava project and is being developed in the framework of the Large Knowledge Collider project. The key feature of mpiJava is that it wraps the calls of the native C library, which mpiJava is installed on top of (e.g. MPICH[6] or Open MPI[7]). This allows mpiJava to substitute MPI operations with calls to the native library (thanks to Java Native Interface - JNI), that ensures better communication performance as in case of the "Java-only" realization, as was for example done in MPJ-Express [17] library (see Fig. 3). Nevertheless, MPJ-Express can be greatly utilized for debugging purposes on the user's local machine before porting to a large HPC system.

[5] https://sourceforge.net/projects/mpijava/
[6] http://www.mcs.anl.gov/research/projects/mpich2/
[7] http://www.open-mpi.org

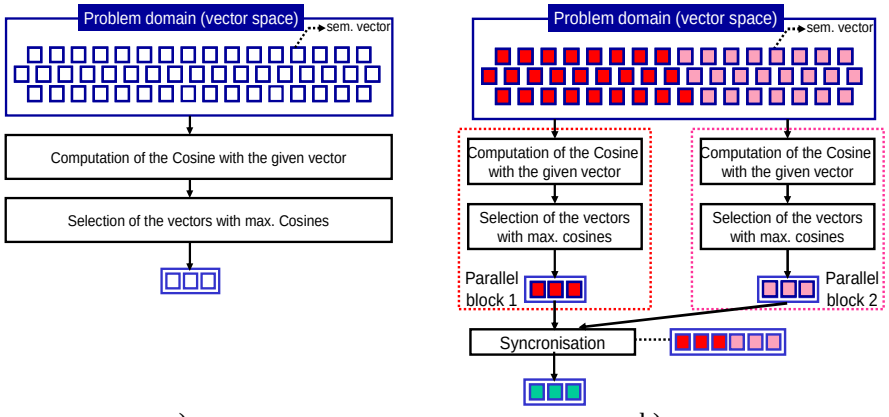


a) b)

Fig. 3. Comparison of time (a) and bandwidth (b) of inter-node communication with different Java MPI libraries on the HLRS Nehalem cluster with Ethernet and Infiniband interconnects.

3.3 MPI Realisation for Random Indexing

In both query expansion and subsetting computing the cosine between a query vector and the rest of the vectors in the space (nearest neighbours) is the operation that benefits the most from paralellization. From now on we refer to the objects we operate with as words, even though they may be all kinds of things such as IDs, genome locations, proteins, and other non-word strings. Fig. 4a shows a schema of selection of the $N=3$ most similar words from the semantic vector space to the given word. Provided that the vectors are analyzed independently of each other and in an arbitrary order, the parallelization can be trivially achieved by applying a data domain decomposition to subdivide the vector space among the processors of the parallel system (Fig. 4b).



a) b)

Fig. 4. The sequential (a) and the parallel (b) realization of the cosine similarity analysis in the Random Indexing algorithm.

In such a parallel fashion, each of the vector subspaces is processed by a separate processor based on the domain boundaries (Listing 1).

```
int my_rank = MPI.Rank();
int subspace_begin = VectorSpace.size() * (my_rank);
int subspace_end = VectorSpace.size() * (my_rank + 1);
```

Listing 1. Calculation of the subspace boundaries for parallel processing with *MPI.Rank()* function.

Each parallel processor is identified by its 'rank' – absolute and unique number in the total processor scope, assigned by the MPI run-time environment. The processor ranking starts from 0 and ends with <*n-1*>, where *n* is a total number of the requested processors. For example, for configuration of two MPI processes, the first one will be assigned the rank "0, and the second one the rank "1". Based on the rank, each process calculates the subdomain to be processed and starts the cosine similarity analysis in the corresponding subdomain. As a result of the analysis, each of the processes produces a list of the most similar words. However those results are incomplete as valid only for a particular subdomain. In order to generalize those partial results for the whole vector space domain, they should be collected in one of the processes (a root) that finalize the search (see the "synchronization" block in Fig. 4b). Usually the process with the rank *0* acts as the root process, however any other process can be assigned by the programmer as a root.

Collection of the results from each subdomain can be achieved by the means of the collective *MPI.Gather* function, according to the schema shown in Fig. 5.

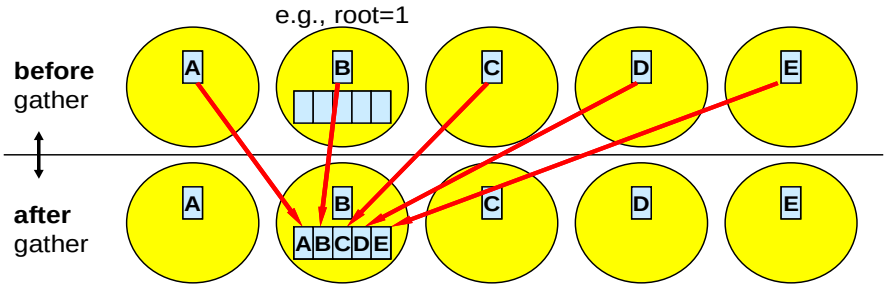


Fig. 5. Schema of the *MPI.Gather* function for collecting the each process's output in the root process.

4 Performance Evaluation and Benchmarking

Performance of the parallel Airhead realisation was evaluated for the four semantic space configurations described in Table 1. The use cases were executed on the

Nehalem[8] cluster of HLRS, whose compute nodes are equipped with Intel Xeon 2.8 GHz processors, interconnected with Infiniband. Configuration of 1, 2, 4, 8, and 16 compute nodes, each equipped with 12 GB of shared RAM space, were benchmarked. The MPI environment was served by mpiJava installed on top of Open MPI 1.4.1.

Table 1. Characteristics of the benchmarked semantic spaces.

Semantic Space	Nr. of vectors	Disk size, GB	Description
LLD1	0,064 M	0,082	Subset of LLD
LLD2	0,5 M	0,65	Subset of LLD
Wiki1	1 M (low density, terms only)	1,6	Term set from 1M most central Wikipedia articles
Wiki2	1 M (high density, entire documents)	16	Document set from 1M most central Wikipedia articles

The quantitative evaluation was focused on the total execution time, the time for loading the vector space from the file on the disk, the duration of the search operation as well as the overhead of the inter-node MPI communication. The obtained benchmarking results are collected in Table 2.

Table 2. Evaluation results on the Nehalem cluster.

Semantic Space	Nr. of compute nodes (processes)	Time, s.				Speed-up
		Loading	Search	MPI comm.	Total	
LLD1	1	-[9]	-	-	2	1
	2	0,75	0,5	0,04	1,61	1,25
	4	0,4	0,4	0,05	1,2	1,7
	8	0,23	0,32	0,1	1,01	1,98
	16	0,17	0,29	0,16	0,94	2,13
LLD2	1	12	6	-	19,5	1
	2	4	3,3	0,03	7,9	2,47
	4	2,4	1,8	0,23	4,6	4,24
	8	1,2	1	0,16	2,9	6,72
	16	0,6	0,7	0,2	2	9,75
Wiki1	1	18	4	-	22	1
	2	8,9	3,8	1	13,3	1,65
	4	4,6	2	0,08	7,4	2,97
	8	2,3	1,3	0,23	4,4	5
	16	1,2	0,75	0,52	2,8	7,86
Wiki2	1	309	83	-	395	1
	2	59	27	0,58	88	4,5
	4	35	13	16	59,1	6,7
	8	20	8	4	32,2	12,3
	16	10	3,7	0,16	14,6	27

[8] http://www.hlrs.de/systems/platforms/nec-nehalem-cluster/

[9] This configuration was not tested due to the RAM limitation on the single node

The evaluation results show that the MPI-parallelized version of Random Indexing scales well on the parallel architecture for the use cases of any complexity, varying from the sparse term vectors (LLD1) to large data sets containing a million of documents (Wiki2), despite the increasing communication overhead. In the best case, the performance speed-up achieved by the parallelized algorithm was approximately 27 times.

5 Related Work

Despite Parallel Computing being recognized in the Semantic Web domain quite recently, there have been some work performed towards high performance reasoning, such as introduced in [18]. However the majority of those approaches only introduce ideas and roadmaps towards developing scalable algorithms rather then describe working prototypes (i.e., concrete applications) that can be quantitatively evaluated.

An outstanding approach is the work presented in [19] that addresses the concrete problem of scalable distributed reasoning by introducing a realization for materialising the closure of an RDF graph based on the Hadoop implementation of the MapReduce framework. This realization is very promising in terms of the performance and scalability that can be achieved on a production-level HPC system. On the other hand, the application scenario described in this work was developed from scratch and it is not clear what the development efforts for an existing application are. Combination of both those approaches, the one presented in [19] and the one elaborated by us, would be of potential interest for building the next-generation reasoning applications and porting them to a large HPC, such as a new Cray computer which will be installed at HLRS[10] in the nearest future and will offer several hundreds of CPU cores.

6 Conclusion and Future Work

The Semantic Web and the High Performance Computing communities have traditionally been somewhat disjoint. However, as the needs and capabilities of these two communities continue to converge, it will be beneficial for both to mutually leverage their respective technologies. This paper presents our approach to parallelise 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 Random Indexing algorithm in order to take advantage of large-scale distributed parallel systems and thus to improve its performance.

[10] http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=1486975&highlight=

Our parallelization technique is based on the domain decomposition. The technique was presented using the simplest constructions from the MPI specification. It required a minimum of implementation efforts in the original application's code and is quite generic, so that it may be easily reused in any other application.

In the future work, we will demonstrate the benefits of MPI for parallelization of further applications. On the other hand, we will investigate the approach proposed in [19] where we are going to look for a trade-off between both MPI and MapReduce based techniques.

7 Acknowledgments

This research has been supported in part by the LarKC EU co-funded project (ICT-FP7-215535). We would also like to thank the main developer of the Airhead library – David Jurgens – for his support.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems 5(3), 1--22 (2009)
2. Hartig, O., Zhao, J.: Using web data provenance for quality assessment. In: Freire, J., Missier, P., Sahoo, S.S. (eds.) Proceedings of the First International Workshop on the role of Semantic Web in Provenance Management (SWPM 2009), CEUR-WS.org (2009)
3. Broekstra, J., Kampman, A.: SeRQL: A Second Generation RDF Query Language. Most, 1-4 (2003)
4. Prud□Hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C working draft 2009. Available at: http://www.w3.org/TR/rdf-sparql-query/
5. Fensel, D., van Harmelen, F.: Unifying Reasoning and Search to Web Scale. IEEE Internet Computing 11(2), 96--95 (2007)
6. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. Proceedings of the thirteenth ACM international conference on Information and knowledge management, pp. 652--659, ACM (2004)
7. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the Open Linked Data. (K. Aberer, K. Choi, N. Noy, D. Allemang, K. Lee, L. Nixon, J. Golbeck, et al., Eds.) The Semantic Web, LNCS, vol. 4825, pp. 552--565, Springer (2007)
8. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data, International Journal of Metadata, Semantics and Ontologies 3(1), 37--52 (2008)
9. Landauer, T. K., Foltz, P. W., Laham, D.: An introduction to latent semantic analysis. Discourse Processes 25(2), 259--284 (1998)
10. Sahlgren, M.: An introduction to random indexing. Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering TKE 2005, pp. 1--9 (2005)
11. Jurgens, D., Stevens, K.: The S-Space Package: An Open Source Package for Word Space Models. Proceedings of the ACL 2010 System Demonstrations, pp. 30--35, Association for Computational Linguistic (2010)

12. Akhter, S., Roberts, J.: Multi-Core Programming: Increasing Performance Through Software Multi-Threading, Intel Press (2006)
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107--113 (2008)
14. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, (1994)
15. The MPI standard version 1.1. Available at http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/mpi-report.htm
16. Getov, V., Gray, P., Sunderam, V.S.: MPI and Java-MPI: contrasts and comparisons of low-level communication performance. ACMIEEE 1999 Conference Supercomputing, pp. 1--16, ACM/IEEE (1999)
17. Carpenter, B., Getov, V., Judd, G., Skjellum, A., Fox, G.: MPJ: MPI-like message passing for Java. Concurrency Practice and Experience 12(11), 1019--1038 (2000)
18. Bock, J.: Parallel computation techniques for ontology reasoning. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) The Semantic Web - ISWC 2008, LNCS, vol. 5318, pp. 901--906. Springer (2008)
19. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable Distributed Reasoning Using MapReduce. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) The Semantic Web - ISWC 2009, LNCS, vol. 5823, pp. 634--649, Springer (2009)
20. Damljanovic, D., Petrak, J., Cunningham, H.: Random Indexing for Searching Large RDF Graphs. Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010), Springer (2010)
21. Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Della Valle, E., Fischer, F., Huang, Z., Kiryakov, A., Lee, T. K., Schooler, L., Tresp, V., Wesner, S., Witbrock, M., Zhong, N.: Towards LarKC: A Platform for Web-Scale Reasoning. In: Proceedings of the 2008 IEEE international Conference on Semantic Computing ICSC, pp. 524--529, IEEE Computer Society (2008)
22. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection, The Journal of Machine Learning Research 3, 1157--1182 (2003)