

The Quest for Parallel Reasoning on the Semantic Web

Peiqiang Li¹, Yi Zeng¹, Spyros Kotoulas², Jacopo Urbani², and Ning Zhong^{1,3}

¹ International WIC Institute, Beijing University of Technology
Beijing 100124, P.R. China.
{lpeiqliang,yzeng}@emails.bjut.edu.cn

² Department of Computer Science, VU University Amsterdam
De Boelelaan 1081, 1081HV, Amsterdam, the Netherlands.
{kot,j.urbani}@few.vu.nl

³ Department of Life Science and Informatics, Maebashi Institute of Technology
460-1 Kamisadori-Cho, Maebashi 371-0816, Japan.
zhong@maebashi-it.ac.jp

Abstract. Traditional reasoning tools for the Semantic Web cannot cope with Web scale data. One major direction to improve performance is parallelization. This article surveys existing studies, basic ideas and mechanisms for parallel reasoning, and introduces three major parallel applications on the Semantic Web: LarKC, MaRVIN, and Reasoning-Hadoop. Furthermore, this paper lays the ground for parallelizing unified search and reasoning at Web scale.

1 Introduction

The Semantic Web provides services for exchanging and utilizing data, information and knowledge in different forms on the Web. Its goal is to organize the web of data through its semantics in a machine-understandable format that can be utilized for problem solving on the Web. In addition, the Semantic Web is dedicated to providing an information sharing model and platform that is convenient for both human and machine to understand and cooperate [1]. Reasoning engines are one of the foundational systems that enable computer systems to automatically reason over Semantic Web data according to some inference rules.

Currently, Semantic Web reasoners are usually deployed on a single machine. This solution is applicable when the datasets are relatively small. In a Web context, it is unreasonable to expect fast enough reasoning mechanisms that work on a single processor, especially when dealing with Web-scale RDF and OWL datasets. Firstly, on the Web, large corpuses of semantically rich data is found, posing new challenges to processing techniques. Secondly, the Web of data is growing very fast and is dynamic. Thirdly, rules might be represented in different forms, which requires a reasoning task to do preprocessing and coordination with other reasoners [2, 3]. In the face of these new requirements, existing reasoning methods have lost their effectiveness. Besides developing new forms of reasoning, another solution is to parallelize the reasoning process [3–5].

The rest of the paper is organized as follows. Section 2 summarizes the current status for parallel reasoning outside the Semantic Web. Section 3 describes in detail the problem of parallel reasoning on the Semantic Web. Section 4 describes LarKC, MaRVIN and Reasoning-Hadoop as three implementations. Finally, based on the analysis of present parallel reasoning techniques on the Semantic Web, we make some preliminary discussion on a related topic which is yet to be fully explored: parallelizing ReaSearch (the unification of reasoning and search) at Web scale.

2 Parallel Reasoning

It is commonly agreed that the goal of parallel reasoning is to accelerate the reasoning process. The major differences between a single processor-based method and a parallel environment-based method can be summarized from two perspectives: platform architecture and algorithm.

2.1 Platform Architecture Perspective

From the perspective of instruction and data streams, three types of architectures are considered, namely, SIMD (single instruction stream, multiple data streams), MISD (multiple instruction streams, single data streams), and MIMD (multiple instruction streams, multiple data streams) [6]. From the perspective of memory, three types of architectures are considered, namely, SMP (Symmetric Multiprocessing), DMP (Distributed Memory Parallel), and HMS (Hierarchical Memory systems) [7]. Systems with an SMP architecture are composed of several interconnected processors that have a shared memory. In DMP, each processor maintains its own memory. Local memory access is fast but distributed memory access has to be done through an interconnect network and is slow [7, 8]. HMS is a hybrid architecture which is an integration of SMP and DMP. In this architecture, clusters of nodes have an SMP architecture internally and a DMP architecture across clusters. Commonly-found clusters of multi-processor or multi-core nodes are also considered to have an HMS architecture [7].

2.2 Algorithm Perspective

Besides the differences on system architecture, a set of parallel reasoning dispatching algorithms have also been developed. The goal for adopting parallel reasoning is to accelerate the reasoning speed. However, as the number of machines increases, the reasoning speed may increase only sublinearly because of communication overhead. Besides, each machine may not be assigned the same amount of processing and thus, not run in its maximum capacity [9]. Thus, one of the key issues for parallel reasoning algorithms is minimizing communication while maintaining an even distribution of load (load balancing) [10]. In order to solve this problem, several algorithms were proposed. Here we introduce two

of them. Although they are proposed before and outside the Semantic Web, we believe they may bring some inspiration.

The load balancing algorithm in [11] uses a matrix to denote the distance from two arbitrary reasoning engines. Another matrix is used to represent the mappings (task transfer from one to another) between two arbitrary reasoning engines [11]. The reasoning engine selection strategy can be summarized as follows: the algorithm adopts the shortest path algorithm. Firstly, it tries to map among the nearest reasoning engines, and after the mapping, remove those which have been occupied. Then, the mapping will try those reasoning engines which have the second shortest path. This progress repeats until all the reasoning engines are assigned. The advantage for this algorithm is that it can drastically reduce communication, its weakness is that it is centralized [11].

The load sharing by state (LDSHBS) algorithm [12] restricts the communication to the nearest reasoning engines. This algorithm uses a tree-structured allocator which has a structure similar to a binary tree. All reasoning engines are at the lowest level. Some nodes supervise reasoning engines in the system. If these nodes detect that some reasoning engines are overloaded, they will remove some tasks from them. If they find some engines are underutilized, the nodes will allocate some additional tasks to them [12]. The advantage of this algorithm is that it can allocate tasks locally and reduce communication. Besides, the algorithm also ensures that each reasoning engine gets a reasonable number of tasks. The disadvantage is that choosing an appropriate number of supervision nodes might be a hard problem [12], especially for Web scale reasoning.

3 Parallelizing Semantic Web Reasoning

Since the data on the Semantic Web is mainly presented using RDF, reasoning on the Semantic Web is mainly focused on RDFS and OWL. We can identify two major goals in Semantic Web reasoning. The first goal is to check the consistency of the web of data so that the data from different sources are well integrated [13]. The second goal is to find new facts (implicit semantic relations) based on existing facts and rules [13, 14].

Compared to traditional parallel processing, the Semantic Web has some additional concerns. Firstly, there are too many nodes in a Web-scale RDF dataset, and each node may have many predicates associated with it. This makes data dependencies complex. Hence, partitioning RDF data is not easy. Secondly, configuring the parallel hardware environment may need to meet new challenges considering Web scale data. Since the dataset is dynamically changing and grows very fast, we cannot assume a static environment. Thirdly, so far, there are not many parallel reasoning algorithms which can be directly imported from other areas to solve Semantic Web parallel reasoning. Finally, load balancing on each machine is still a hard problem to solve, given the very skewed nature of Semantic Web terms. In this light, the current state-of-the art in practical parallel reasoning is rather poorly developed.

For Parallel Semantic Web Reasoning, two major trends are introduced to process reasoning in parallel, namely, data partitioning approaches, and the rule partitioning approaches [15]. In data partitioning approaches, all rules are applied by all reasoners while data is split into smaller partitions and processed in parallel by several reasoners [15]. In rule partitioning approaches, the rules are partitioned into different reasoners to perform the reasoning tasks, and the data has to go through all the reasoners [15, 16]. These studies have also shown that effective partitioning is not easy. Better partitioning methods need to be proposed. Some practical implementations of large-scale parallel reasoning are described in the following section.

4 Some Solutions

Very recently, some practical implementations for parallel Semantic Web reasoning have been published. In this section, we are going to introduce three parallel reasoning approaches for web-scale data: LarKC [3], MaRVIN [5], and Reasoning-Hadoop [4]. While all of these are in the direction of parallel reasoning on the Semantic Web, each of them has a different viewpoint.

4.1 LarKC

The Large Knowledge Collider (LarKC)⁴ is an open architecture and a generic platform for massive distributed reasoning [3]. LarKC currently emphasizes on scalability through parallelization of the execution of an open set of software components. LarKC works as a scalable workflow engine for reasoning tasks. In each workflow, there are several components (plug-ins) which are responsible for diverse processing tasks, for example, identifying relevant data, transforming data, selecting data and reasoning over data. The execution of the workflow is overseen by a decider plug-in [3]. Since several plug-ins are invoked in a workflow, they can be distributed among several nodes and work in parallel [17]. LarKC parallelizes execution in the following ways:

- Invocation of plug-ins that have a parallel implementation;
- Invocation of distinct plug-ins in parallel;
- Execution of several workflows in parallel, or execution of the same workflow with different input in parallel.

Currently, a set of LarKC plug-ins already have a parallel implementation: A Sindice identifiers uses multiple threads (thus exploiting shared memory, multiple processor architectures), and a GATE transformer can run on supercomputers. MaRVIN, which will be introduced in the next section will be wrapped as a parallel and distributed reasoner for LarKC. As work in progress, USER-G (Unifying Search and Reasoning from the perspective of Granularity)⁵ is also a series of methods that aims at working in a parallel environment for LarKC.

⁴ <http://www.larkc.eu>

⁵ <http://www.iwici.org/user-g>

Currently, LarKC is considering some parallel programming models (e.g. OpenMP, HPF (High Performance Fortran), and MPI (Message Passing Interface)) and some frameworks (e.g. the Ibis framework [18]) to offer as an API for developing parallel components. The core mechanism of OpenMP is based on shared memory directives [19] and task decomposition, but does not provide how to do decomposition [7]. Although HPF provides specific data decomposition, the rules of HPF may cause a too high communication overhead [7]. MPI allows the developers to specify the distribution of the work and the data. How and when the communication is done can also be specified [7]. According to [7] LarKC may consider using the Ibis framework [18]. Ibis is a grid programming environment that combines portability, flexibility and high efficiency [18]. The parts which are relevant to LarKC are the Ibis portability layer (IPL) [18] and the MPJ/Ibis [20].

4.2 MaRVIN

As a part of the LarKC project, MaRVIN (Massive RDF Versatile Inference Network)⁶ is a parallel and distributed platform for processing large amounts of RDF data. MaRVIN is the first practical parallel reasoning implementation [5].

The work on MaRVIN is motivated by the observation that it is hard to solve Semantic Web problems through traditional divide-and-conquer strategies since Semantic Web data is hard to partition [5].

MaRVIN brings forward a method named divide-conquer-swap [5] to do inferencing through forward chaining (i.e. calculate the closure of the input data). The main algorithm can be described in the following steps: First, the platform divides the input data into several independent partitions and assigns this partitions to compute nodes. Second, each compute node computes the closure of its partition using a conventional reasoner. Then, old and new data is mixed and new partitions are created in a distributed manner. This process is repeated until no new triples are derived. At this point, the full closure has been calculated.

In the context of MaRVIN, the SPEEDDATE routing strategy has been developed [5]. RDFS and OWL rules are triggered by triples that share at least one term. SPEEDDATE makes partitions in a distributed manner while increasing the triples with the same terms that belong to the same partition. This way, the number of partitioning-reasoning cycles that need to be performed to calculate the full closure is reduced.

The advantages of the MaRVIN platform are the following:

- Since the partitions are of equal size, the amount of data to be stored and the computation to be performed is evenly distributed among nodes;
- No upfront data analysis is required;
- It can support any monotonic logic by changing the conventional reasoner;
- It uses a peer-to-peer architecture. Thus, no central coordination is required;
- It shows anytime behavior, i.e. it produces results incrementally with time.

⁶ <http://www.larkc.eu/marvin/>

The latest experiments on real-world datasets show that MaRVIN can calculate the closure of 200M triples on 64 compute nodes in 7.2 minutes, yielding a throughput of 450K triples per second.

In terms of the definitions in sections 2 and 3, MaRVIN does data partitioning on a HMP architecture. Compared to traditional reasoners, MaRVIN shows higher loading speeds but is limited to calculating the closure of the input.

4.3 Reasoning-Hadoop

Reasoning-Hadoop [4] is a parallel rule-based RDFS/OWL reasoning system built on the top of the Hadoop framework [21]. Hadoop is an open-source framework mainly used for massive parallel data processing initially developed by Yahoo! and now hosted by the Apache foundation.

Hadoop implements the MapReduce programming model. The MapReduce programming model was developed by Google [22] and it requires that all the information is encoded as a set of pairs of the form $\langle key, value \rangle$. A typical MapReduce algorithm takes as input a set of pairs, processes them using two functions, *map* and *reduce*, and returns some new pairs as output. The program execution is handled by the framework which splits the input set in subsets and assigns computation to nodes in the network [22].

In the reasoning-hadoop project⁷ RDFS and OWL-Horst forward reasoning has been implemented with a sequence of MapReduce algorithms. In terms of the definitions in sections 2 and 3, Reasoning-Hadoop does both data partitioning and rule partitioning on a DMP architecture.

In [4], it is shown that a naive implementation for RDFS reasoning performs poorly. Thus, three non-trivial optimizations were introduced:

- the schema triples are loaded in the nodes' main memory and the rules are applied on the fly with the instance triples;
- the rules are applied during the *reduce* function, and the *map* function is used to group together the triples that could lead to a duplicated derivation;
- the rules are applied in a certain order so there is no need to apply the same rule multiple times.

The refined version of the algorithm proved to have very high performance. The RDFS reasoner is able to compute the closure of the 1 billion triples of the 2008 Billion Triples challenge in less than 1 hour using 33 machines [4]. This approach currently outperforms any other published approach.

The performance of the OWL reasoner is not yet competitive. The optimizations introduced for the RDFS semantics do not apply for some of the rules of the OWL Horst fragment. Current research is focused on finding optimizations that apply to this fragment.

Concluding, Reasoning-Hadoop has shown that there are several advantages in using MapReduce for reasoning in Semantic Web. First, the reasoning can

⁷ <https://code.launchpad.net/jrbn/+junk/reasoning-hadoop>

be done efficiently on large datasets because the Hadoop framework can be deployed in networks with thousand of nodes. Second, the execution is handled completely by the framework and the programmer can concentrate on the logic of the program without worrying about the technical problems that are common in distributed systems. The main disadvantage of this approach lies in dealing with more complex logics. For example, the rules of the OWL Horst fragment are more complex and more difficult to encode efficiently. However, given the early stage of this research, a verdict is yet to be reach for the applicability of this approach to other logics.

5 Evaluation of Parallel Semantic Web Reasoning Approaches

Compared to traditional parallel reasoning, parallel Semantic Web reasoning approaches pose unique challenges in their evaluation. Thus, a new set of evaluation criteria needs to be defined. We can make a distinction between functional and non-functional characteristics of such systems.

Functional characteristics refer to the functions a system can perform. In the context of parallel Semantic Web reasoning, we can identify the following functional characteristics:

- *Logic applied*: depending on the logic implemented, various optimizations may be possible. For example, the antecedents of the RDFS match at most one instance triple [4]. The approach presented in the previous section exploits this fact to optimize RDFS reasoning by loading schema triples in memory and processing instance triples as a stream. Nevertheless, this optimization cannot be applied to OWL horst reasoning, because antecedents in the OWL horst ruleset may contain multiple instance triples.
- *Degree of completeness*: tolerating incomplete results can dramatically speed up the reasoning tasks [2]. This is a common compromise made in search engines. Furthermore, sometimes, the number of answers grows sublinearly with time. Depending on the task, it may be acceptable to return a fraction of the total answers spending a (smaller) fraction of the time that would be required to calculate all answers. MaRVIN [5] is an example of a system with this characteristic.
- *Correctness*: similarly to completeness, accepting incorrect answers may also increase performance [2].

Non-functional characteristics refer to constraints in performing the prescribed functions. Some relevant non-functional characteristics for parallel reasoning are:

- *Triple throughput*: a central measure for the performance of a reasoning system is the number of triples it can process per second, indicating its efficiency [4, 5].
- *Query throughput*: another performance measure for reasoning systems doing query answering is the number of queries they can process per second.

- *Query response time*: relevant to the previous characteristic, query latency refers to the amount of time between posting a query and getting the answer.
- *Scalability* refers to the capability of a system to handle larger input and to efficiently use additional computational resources. In parallel systems, computational resources are usually refer to computation nodes.
- *Maximum input size*: depending on the approach, there may be hard limits in the amount of data a system can handle, given some hardware. If these limits are reached, the approach becomes impractical. These limits may be imposed by restrictions in the available memory or hard disk space.

6 ReaSearch and Its Parallelization

Parallelizing reasoning is an attempt to solve the scalability problems for Web scale reasoning by introducing additional computational power. Nevertheless, because of the limitations and assumptions of traditional reasoning methods [2], complementary methods should be developed.

6.1 The Necessity of Parallelizing ReaSearch

ReaSearch⁸, which stands for unifying reasoning and search to Web scale, was proposed in [2] and is further developed in the LarKC project [3]. It emphasizes on an interweaving process of searching an important subset from the Web of data and do reasoning on it. The interweaving process will not stop until the user is satisfied with the reasoning result [2], as shown in Figure 1(a) (Note that this workflow design is a part of the search and reasoning part in LarKC [3]). On the Semantic Web, both the search and reasoning process need to handle massive data. Hence, the ReaSearch process need to be parallelized. ReaSearch is a framework for unifying reasoning and search, and concrete strategies on how to implement this framework can be developed through different methods.

USeR-G (**U**nifying **S**earch and **R**easoning from the viewpoint of **G**ranularity)⁹ is a set of strategies that aim at combining the idea of granularity [23, 24] and ReaSearch to provide concrete implementations of ReaSearch for the LarKC Project. The set of strategies include: unifying search and reasoning through multilevel completeness, multilevel specificity, multiperspective, etc [25]. In this paper, we will not go into details of these strategies, we just mention some processing steps in the implementation of these strategies where parallelization is needed. For the multilevel completeness strategy, the search process needs two parameters for the selection of important sub dataset. Namely, node degree calculation and node number statistics for the whole dataset. In one of our experiments for calculating the number of co-authors from the SwetoDBLP dataset [25], we spend more than one hour on the 1.08G semantic dataset, which is unacceptable, and should be parallelized. As indicated in [2], Web scale data

⁸ <http://www.reasearch.net/>

⁹ <http://www.iwici.org/user-g>

may be over 10 billion triples. Hence, the task of calculating the node number for the multilevel completeness strategy can be parallelized to save time. For the multilevel specificity strategy, nodes that are distributed in different levels of specificity can be assigned to multiple nodes in order to save processing time. For the multiperspective strategy, since the unification of search and reasoning can be done from multiple perspectives to meet the diverse user needs, different perspectives can be processed in parallel so that one can get reasoning results from all perspective almost at the same time. For all of the strategies, all of the reasoning parts can be and should be parallelized.

6.2 A Preliminary Design for the Parallel ReaSearch Architecture

As mentioned above, parallelizing ReaSearch is not only about parallelizing reasoning. Both search and reasoning need to be considered in the parallel environment. In this section, we propose a preliminary parallel ReaSearch architecture.

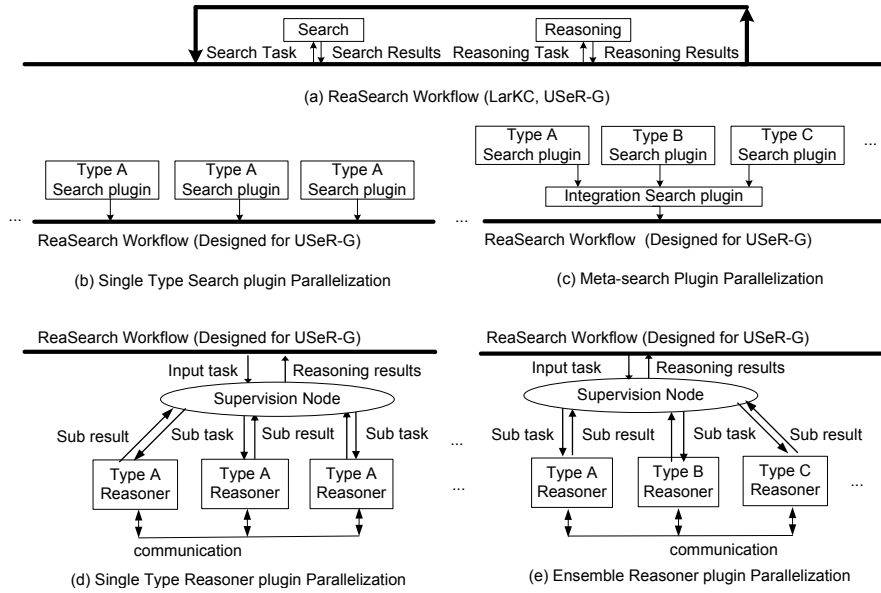


Fig. 1. A Parallel Architecture for ReaSearch.

Following the design principles of parallel reasoning [26], for a parallel ReaSearch architecture, the nodes with different functionalities are distributed physically, but unified logically. As mentioned in Section 4.1, the unification of search and reasoning can be implemented in a workflow, hence the search process can be

done in one node, and the reasoning process can be done in another. An alternative strategy, which scales better, is that the search and the reasoning processes themselves are also parallelized.

For the search part, search plugins are parallelized as shown in Figure 1(b),(c). In Figure 1(b), the search tasks are parallelized by dividing the dataset into several sub datasets and each subset is handled by one search plugin and the search results from different search plugins are independently sent to the ReaSearch workflow for reasoning. Figure 1(c) provides an architecture for a meta-search plugin, which was inspired by the study of meta-search engine [27] for information retrieval. For this type of parallel search plugin, all the RDF/OWL datasets go through different types of the search plugins (Type A,B,C, etc.), and an integration search plugin is used to integrate all the search results from different type of search plugins and select out the most important subset of RDF/OWL data for reasoning. The selection criterion is that if a subset of the original dataset appears in all the search results from different search plugins, then it is considered as the most important subset and is delivered through the ReaSearch workflow for the reasoning task. If a subset appears in only some of the search results, then it is considered as a less important subset. Although this meta-search plugin has a parallel architecture, its aim is not to speed up the search process. Instead, it helps to select out the most important subset for reasoning.

For the reasoning part, two types of architectures are provided, as shown in Figure 1(d),(e). Each of them has a supervision node, hence both of them have centralized architectures. Reasoning is part of the ReaSearch workflow and is working in parallel with the parallel search architecture. In the architecture shown in Figure 1(d), reasoners are identical and apply the same rules. The dataset is divided in several parts to be processed on different reasoners(data partitioning). Its aim is to improve the speed of reasoning, which is similar in spirit with Reasoning-hadoop and MaRVIN. In Figure 1(e), the sub reasoning engines are different. This architecture refers to ensemble reasoning [28], which is inspired from ensemble learning in the field of machine learning. This architecture is not aimed at speeding up the reasoning process. The motivations behind this approach lie in: (1) Reasoning based on Web-scale data may produce too many results. Among these results, only some parts may be useful to a certain user. With the different reasoning plugins involved, various reasoning results will be obtained. If some results appear in the result sets from all reasoners, they can be considered as more important than others. The integration node can provide the most important reasoning results by selecting out the ones which appear in all or most sub result sets. (2) With different reasoning plugins involved, one may get more meaningful results compared to the first type shown in Figure 1(d). In this case, the integration node is responsible for merging all the reasoning results together for user investigation. In a more user-oriented scenario, the architecture also enables users to configure how many reasoners of each type they prefer to use for their own reasoning tasks. In this way, each type of reasoner will have a weight on the whole architecture. When these weights are changed, the reasoning results may also differ to meet different user needs.

7 Conclusion

The Semantic Web brings many new problems and insights to the field of parallel reasoning. New architectures and algorithms need to be developed to fit the context of Web scale reasoning. LarKC, MaRVIN, and Reasoning-Hadoop are three practical systems which have touched this area and are potentially effective. Nevertheless, there is still much more that has not been well explored, such as how to develop concrete strategies for unifying reasoning and search in a parallel environment (i.e. parallelizing ReaSearch).

Moreover, through the preliminary design for the parallel ReaSearch architecture, we notice that for the reasoner part, a parallel architecture can improve the reasoning speed while also producing more fruitful reasoning results to meet diverse user needs. This topic needs further investigation. In this paper, we try to provide some preliminary discussion to inspire more research results in this area. We had some very preliminary discussions on where do ReaSearch (more specifically, the set of methods UseR-G) should be parallelized. In future work, we will go into deeper discussion and concrete implementations.

Acknowledgements

This study was supported by the European Union 7th framework project FP7-215535 LarKC (Large Knowledge Collider) and VU University Amsterdam. This paper was prepared when Yi Zeng was visiting VU University Amsterdam.

References

1. Berners-Lee, T.: The semantic web. *Scientific American* **6** (2001) 1–6
2. Fensel, D., van Harmelen, F.: Unifying reasoning and search to web scale. *IEEE Internet Computing* **11**(2) (2007) 96, 94–95
3. Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E., Fischer, F., Huang, Z., Kiryakov, A., Lee, T., School, L., Tresp, V., Wesner, S., Witbrock, M., Zhong, N.: Towards larkc: A platform for web-scale reasoning. In: *Proceedings of the International Conference on Semantic Computing*. (2008) 524–529
4. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using mapreduce. In: *Proceedings of the International Semantic Web Conference*. (2009)
5. Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., Ten Teije, A., van Harmelen, F.: Marvin: distributed reasoning over large-scale semantic web data. *Journal of Web Semantics* (to appear)
6. Flynn, M.: Very high-speed computing systems. *Proceedings of the IEEE* **54**(12) (1966) 1901–1909
7. Gallizo, G., Roller, S., Tenschert, A., Witbrock, M., Bishop, B., Keller, U., van Harmelen, F., Tagni, G., Oren, E.: Summary of parallelisation and control approaches and their exemplary application for selected algorithms or applications. In: *LarKC Project Deliverable 5.1*. (2008) 1–30

8. Wilkinson, B., Allen, M.: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. 2 edn. Prentice Hall (2005)
9. Robert, B.: Japan's pipedream: The fifth generation project. *System and Software* September **3** (1984) 91–92
10. Ehud, S.: Systolic programming: A paradigm of parallel processing. In: *Proceedings of the international conference on Fifth Generation Computer Systems*. (1984) 458–470
11. Liu, Z., You, J.: Dynamic load-balancing on a parallel inference system. In: *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*. (1990) 58–61
12. Tan, X., Zhang, X., Gao, Q.: Load sharing algorithms for parallel inference machine epim-ldshbs, intlsh. *Chinese journal of computers* (5) (1986) 321–331
13. Allemang, D., Hendler, J.: *Semantic Web for the Working Ontologist*. Elsevier, Inc. (2008)
14. Brachman, R., Levesque, H.: *Knowledge Representation and Reasoning*. Elsevier, Inc. (2004)
15. Soma, S., Prasanna, V.: Parallel inferencing for owl knowledge bases. In: *Proceedings of the 37th International Conference on Parallel Processing*. (2008) 75–82
16. Schlicht, A., Stuckenschmidt, H.: Distributed resolution for alc. In: *Proceedings of the International Workshop on Description Logic*. (2008)
17. Oren, E.: Goal: Making pipeline scale. Technical report, LarKC 1st Early Adopters Workshop (June 2009)
18. van Nieuwpoort, R., Maassen, J., Wrzesinska, G., Hofman, R., Jacobs, C., Kielmann, T., Bal, H.: Ibis: a flexible and efficient java based grid programming environment. *Concurrency and Computation: Practice and Experience* **17**(7-8) (2005) 1079–1107
19. Chapman, B., Jost, G., van der Pas, R., Kuck, D.: *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press (2007)
20. Bornemann, M., van Nieuwpoort, R., Kielmann, T.: Mpij/ibis: a flexible and efficient message passing platform for java. In: *Proceedings of 12th European PVM/MPI Users' Group Meeting*. (2005) 217–224
21. Hayes, P.: Rdf semantics. In: *W3C Recommendation*. (2004)
22. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*. (2004) 137–150
23. Hobbs, J.: Granularity. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence*. (1985) 432–435
24. Yao, Y.: A unified framework of granular computing. In: *Handbook of Granular Computing*. Wiley (2008) 401–410
25. Zeng, Y., Wang, Y., Huang, Z., Zhong, N.: Unifying web-scale search and reasoning from the viewpoint of granularity. In: *Proceedings of the 2009 International Conference on Active Media Technology*. (2009)
26. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. In: *Proceedings of the European Semantic Web Conference*. (2005) 361–376
27. Howe, A., Dreilinger, D.: Savvysearch: a meta-search engine that learns which search engines to query. *AI Magazine* **18**(2) (1997) 19–25
28. Chabuk, T., Seifter, M., Salasin, J., Reggia, J.: Integrating knowledge-based and case-based reasoning. Technical report, University of Maryland (2006)