Abstract

The volume of data exposed on the Web is increasing at a robust pace. Reasoning is a wide-spread knowledge discovery and information retrieval technique, in particular extensively used for developing Web applications. However, most of the reasoning algorithms are dealing with significant challenges when being scaled up to the problem sizes addressed by the modern Semantic Web, breaking the barrier of billions of RDF statements (triples). Unfortunately, reasoning applications are not optimized to be applied to emerging Internet-scale data sets, known as a "big data" problem. In this paper, we introduce a service-oriented approach to facilitate the development of reasoning applications that can scale to big data demands. The approach is based on an incomplete reasoning engine LarKC (the Large Knowledge Collider) as well as parallelization techniques elaborated for big data applications in the frame of the JUNIPER EU project. We discuss the use of the service-oriented approach to develop two exemplarily resource discovery applications - query expansion and subsetting, based on the random indexing technique.

Keywords: semantic web, Java, parallelization, LarKC, JUNIPER.

1 Introduction

The large- and internet-scale data applications are a primary challenge for the Semantic Web, and in particular for reasoning algorithms, which are used for processing exploding volumes of data that are massively exposed on the Web. Reasoning is a process of making implicit logical inferences from the explicit set of facts or statements that are comprised by a knowledge base. The key problem of modern reasoning engines, such as Jena [1], or Pellet [2], is their inability to be effectively applied to newly-emerging data sets, which span over several billions of triples (a unit of the semantically annotated information in the format “resource-property-value”), collecting the amount of digital information that occupies...
petabytes of storage space. Whereas modern advances in supercomputing and virtualization (cloud) technologies allow this limitation to be overcome, the reasoning algorithms and logic that they implement still require adaptation to the demands of rapidly growing data universe in order to be able to take advantages of the modern computing and storage facilities. On the other hand, the algorithmic principals of the reasoning engines need to be reconsidered as well in order to allow for very large volumes of analysed data. Service-oriented architectures (SOA) can greatly contribute to this goal, acting as the main enabler of the newly proposed reasoning techniques, such as incomplete reasoning [3]. This paper focuses on a service-oriented solution for developing new-generation Semantic Web applications by means of the Large Knowledge Collider (LarKC)1 platform. LarKC relies heavily on a Java architecture, which is necessary in order to ensure interoperability with the mainstream data analytics solutions, the majority of which are developed in Java language. The parallelization technologies that are referred in the paper are being explored by us within the JUNIPER project2.

The paper is organized as follows. Section 2 is devoted to the discussion on the big data problem for the Semantic Web. In Section 3, we discuss a concept of Semantic Web reasoning as a service – a fundamentally new approach based on the incomplete reasoning, which was first implemented in the LarKC platform. In Section 4, we discuss a bright spectrum of pilot Semantic Web applications that have been implemented with the proposed incomplete reasoning approach, powered by LarKC. In Section 5, we discuss parallelization technologies for applications developed in Java language, based on the evaluation done by the JUNIPER project. Section 6 presents conclusion and highlights directions for future work in highly-scalable semantic reasoning.

2 Scaling Semantic Web Reasoners to Big Data

The volume of data collected on the Semantic Web has already reached the order of magnitude of billions of triples and is expected to further grow in the future, which positions this Web extension to dominate the data-centric computing in the oncoming decade. Processing (e.g., inferring) such a volume of data, e.g. generated by social networks like Facebook or Twitter, or collected in domain-oriented knowledge bases like pharmacological data integration platform OpenPHACTS3, is a big challenge for the currently available software platforms.

Whereas there is a number of existing highly-scalable software solutions for storing data, the scalable processing of data poses a major challenge for data-centric applications, in particular for the Semantic Web ones. The complex of issues related to scaling the existing data processing techniques to the vast amounts of data is often referred as a ‘Big Data’ problem. With regard to the Semantic Web, such issues as size of collected data, velocity of generating new data, and number of parallel operations (e.g. according to the user request) are the major issues.

1 http://www.larkc.eu/
2 http://juniper-project.org/
3 http://www.openphacts.org/
Semantic Data are massively produced and published at the speed that makes traditional processing techniques (such as reasoning discussed in this publication) inefficient when applied to real-scale data. In the Semantic Web, the data size scaling problem is considered in two its main aspects - horizontal and vertical scaling. Horizontal scaling means dealing with diverse, often unstructured data, acquired from heterogeneous sources. The famous Linked Open Data cloud diagram4 consists of hundreds of data sources, ranging from geospatial cartographic sources, like Open Street Map, to governmental data, opened to the publicity, like data.gov. Vertical scaling implies scaling up the size of similarly structured data. Along the open government data spawns over 851,000 data sets across 153 catalogues from more than 30 countries, as estimated in [32] for the beginning of 2012. Processing data in such big amounts is not straightforward and introduces a lot of challenges for the currently existing frameworks and infrastructures. Whereas there are some known algorithms dealing with the horizontal scaling complexity, such as identification of the information subsets related to a specific problem, i.e., subsetting, the vertical scaling remains the major challenge for all existing algorithms.

Another essential property of the Big Data is the complexity. Semantic applications must deal with rich ontological models describing complex domain knowledge, and at the same time highly dynamic data representing recent or relevant information, as produced by streaming or search-enabled data sources. A considerable part of the web data is produced as a result of automatic reasoning over streaming information from sensors, social networks, and other sources, which are highly unstructured, inconsistent, noisy and incomplete. Despite the majority of data on the Web is available as an unstructured text, e.g. generated from the content kept in RDBM, the application areas of the modern Semantic Web spawn a wide range of domains, from social networks to large-scale Smart Cities projects in the context of the future internet [4][5]. However, data processing in such applications goes far beyond a simple maintenance of the collection of facts; based on the explicit information, collected in datasets, and simple rule sets, describing the possible relations, the implicit statements and facts can be acquired from those datasets.

The latest research on web-scale knowledgebases, combined with the proliferation of SOA infrastructures and cloud computing, has created a new wave of data-intensive computing applications, and posed several challenges to the Semantic Web community. As a reaction on these challenges, a variety of reasoning methods have been suggested for the efficient processing and exploitation of the semantically annotated data. However, most of those methods have only been approved for small, closed, trustworthy, consistent, coherent and static domains, such as synthetic LUBM [6] sets. Still, there is a deep mismatch between the requirements on the real-time reasoning on the Web scale and the existing efficient reasoning algorithms over the restricted subsets.

Whereas unlocking the full value of the scientific data has been seen as a strategic objective in the majority of ICT-related scientific activities in EU, USA, and Asia [7], the “Big Data” problem has been recognized as the primary challenger in

---

4 http://richard.cyganiak.de/2007/10/lod/
Indeed, the recent years have seen a tremendous increase of the structured data on the Web with scientific, public, and even government sectors involved. According to one of the recent IDC reports [10], the size of the digital data universe has grown from about 800,000 Terabytes in 2009 to 1.2 Zettabytes in 2010, i.e. an increase of 62%. Even more tremendous growth should be expected in the future (up to several tens of Zettabytes already in 2012, according to the same IDC report [10]).

The big data problem makes the conventional data processing techniques, also including the traditional semantic reasoning, substantially inefficient when applied for the large-scale data sets. On the other hand, the heterogeneous and streaming nature of data, e.g. implying structure complexity [11], or dimensionality and size [12], makes big data intractable on the conventional computing resource [13]. The problem becomes even worse when data are inconsistent (there is no any semantic model to interpret) or incoherent (contains some unclassifiable concepts) [14].

The broad availability of data coupled with increasing capabilities and decreasing costs of both computing and storage facilities has led the semantic reasoning community to rethink the approaches for large-scale inferencing [15]. Data-intensive reasoning requires a fundamentally different set of principles than the traditional mainstream Semantic Web offers. Some of the approaches allow for going far beyond the traditional notion of absolute correctness and completeness in reasoning as assumed by the standard techniques. An outstanding approach here is interleaving the reasoning and selection [16]. The main idea of the interleaving approach (see Figure 1) is to introduce a selection phase so that the reasoning processing can focus on a limited (but meaningful) part of the data, i.e., perform incomplete reasoning.

**Figure 1:** Incomplete reasoning, general approach (a) and service-oriented vision (b).

Although the interleaving approach somewhat simplifies the reasoning problem in terms of the computation resource power needed for data processing, the incomplete reasoning remains a very computation-intensive process, which requires using high...
performance computing for dealing with the issues of data size and processing complexity.

An outstanding in terms of the uptake in the Semantic Web community effort to develop an service-oriented platform for incomplete reasoning, backed by supercomputing infrastructures, was done by the LarKC (Large Knowledge Collider) [20] project. In the following sections, we discuss the main ideas, solutions, and outcomes of this project.

3 Data as a Service: The LarKC Way of Developing Semantic Web Services

In order to create a technology for creation of trend-new applications for large-scale reasoning, several leading Semantic Web research organizations and technological companies have joined their efforts around the project of the Large Knowledge Collider (LarKC), supported by the European Commission. The mission of the project was to set up a distributed reasoning infrastructure for the Semantic Web community, which should enable application of reasoning far beyond the currently recognized scalability limitations [22], by implementing the interleaving reasoning approach. The current and future Web applications that deal with Big Data are in the focus of LarKC.

The LarKC’s design has been guided by the primarily goal to build a scalable platform for distributed high performance reasoning. Figure 2 shows a conceptual view of the LarKC platform’s architecture and the proposed development life-cycle. The architecture was designed to holistically cover the needs of the three main categories of users – semantic service (plug-in) developers, application (workflow) designers, and end-users internet-wide. The platform’s design ensures a trade-off between the flexibility and the performance of applications in order to achieve a good balance between the generality and the usability of the platform by each of the categories of users.

Below we introduce some of the key concepts of the LarKC architecture and discuss the most important platform’s services and tools for them.

Plug-ins

Plug-ins are standalone services implementing some specific parts of the reasoning logic as discussed previously, whether it is selection, identification, transformation, or reasoning algorithm, see more at [21]. In fact, plug-ins can implement much broader functionality as foreseen by the incomplete reasoning schema (Figure 1), thus allowing the LarKC platform to target much wider Semantic Web user community as originally targeted, e.g. for machine learning or knowledge extraction. The services are referred as plug-ins because of their flexibility and ability to be easily integrated, i.e. plugged into a common workflow and hence constitute a reasoning application. To ensure the interoperability of the plug-ins in the workflows, each plug-in should implement a special plug-in API, based on the
annotation language [23]. Most essentially, the API defines the RDF schema (set of statements in the RDF format) taken as input and produced as output by each of the plug-ins. The plug-in development is facilitated by a number of special wizards, such as Eclipse IDE wizard or Maven archetype for rapid plug-in prototyping. The ready-to-use plug-ins are uploaded and published on the marketplace – a special web-enabled service offering a centralized, web-enabled repository store for the plug-ins5.

**Workflows**

The workflow designers get access to the Marketplace in order to construct a workflow from the available plug-ins, combined to solve a certain task. In terms of LarKC, workflow is a reasoning application that is constructed of the (previously developed and uploaded on the Marketplace) plug-ins. The workflow’s topology is characterised by the plug-ins included in the workflow as well as the data- and control flow connections between these plug-ins.

The complexity of the workflow’s topology is determined by the number of included plug-ins, data connections between the plug-ins (in particular involve several split and join branches, such as in Figure 3a, or multiple end-points, such as in Figure 3b), and control flow events (such as instantiating, starting, stopping, and terminating single plug-ins or even workflow branches comprising several plug-ins). Same as for plug-ins, the input and output of the workflow is presented in RDF, which however can cause compatibility issues with the user’s GUI, which are not


Figure 2: Architecture of LarKC.
obviously based on an RDF-compliant representation. In order to confirm the internal (RDF) dataflow representation with the external (user-defined) one, the LarKC architecture foresees special end-points, which are the adapters facilitating the workflow usage in the tools outside of the LarKC platform. Some typical examples of end-points, already provided by LarKC, are e.g. SPARQL end-point (SPARQL query as input and set of RDF statements as output) and HTML end-point (HTTP request without any parameters as input and HTML page as output).

Figure 3: Examples of LarKC workflows: a) workflow with non-trivial branched dataflow (containing multiple splits/joins), b) workflow with multiple end-points.

For the specification of the workflow configuration, a special RDF schema was elaborated for LarKC, aiming at simplification of the annotation efforts for the workflow designers. Figure 4a shows a simple example of the LarKC workflow annotation. Creation of the workflow specification can greatly be simplified by
using upper-level graphical tools, e.g. Workflow Designer that offers a GUI for visual workflow construction (Figure 4b) [28]. The elaborated schema makes specification of the additional features such as remote plug-in execution extremely simple and transparent for the users and can be used for tuning the front-end graphical interfaces of the applications to adapt them to the user needs.

Platform services
All above-described activities related to plug-in creation, workflow design, and application development are facilitated by an extensive set of the platform services, as shown in Figure 2. A detailed description of the main LarKC services can be found in our previous publication [21].

```larkc
# Define plug-ins
_:plugin1 a <urn:eu.larkc.plugin.LLDReasoner> .
_:plugin1 a <urn:eu.larkc.FilteringPlugin.FilteringPlugin>
_:plugin1 larkc:runsOn _:host1 .

# Define hosts
_:host1 a <urn:eu.larkc.host.Tomcat> .
_:host1 larkc:hostType larkc:JEE .

# Define a path to set the input and output of the workflow
_:path a larkc:Path .
_:path larkc:hasInput _:plugin1 .
_:path larkc:hasOutput _:plugin1 .

# Connect an endpoint to the path
_:ep a <urn:eu.larkc.endpoint.sparql.SparqlEndpoint> .
_:ep larkc:links _:path .
```

Figure 4: Further example of LarKC workflows: a) RDF schema for workflow annotation, b) Workflow Designer GUI with the specification of the remote host.
Applications

Workflows are already standalone applications that can be submitted to the platform and executed by means of such tools as Workflow Designer discussed above. Nevertheless, workflows can also be wrapped into much more powerful user interfaces, adapted to the needs of the targeted end-user communities, e.g., Urban Computing [24], and using LarKC as a back-end engine. The service-oriented approach makes possible hiding the complexity of the LarKC platform, by enabling its whole power to the end-users through such interfaces. We present an exemplarily LarKC application in Section 4.

4 Application Scenario Examples

LarKC is the technology that not only enables the large-scale reasoning approach for the already existing applications, but also facilitates their rapid prototyping with low initial investments, leveraging the advantages of SOA architectures by means of the platform solutions discussed in Section 3. Furthermore, LarKC delivers a complete eco-system in which researches from very different domains can team up in order to develop new challenging applications (mash-ups). Below we discuss some of the most prominent pilot applications developed with LarKC, which challenge the big data problem.

Bottari

BOTTARI [34] is a location-based mobile application that leverages a place of interest recommendation system to support people who find themselves in the new place, which they are not familiar with. The application’s front-end is implemented at Android tablets, whereas the back-end is served by LarKC. BOTTARI is collecting relevant information from social media networks such as Twitter and blog posts, elaborates it and provides contextualized suggestions. At the current stage, the application was implemented for one of the most popular touristic districts in Seoul, South Korea. The recommendations given by BOTTARI include places of interest nearby the current location of the user, reputation ranking of the suggested places according to the other users’ feedback, identification of the most interesting place fitting well the user’s profile. To the main innovations of BOTTARI can be referred offering a location-based service through a simple and intuitive interface, advanced semantic features, and hiding the complexity of reasoning from the end-user. BOTTARI become the winner of the International Semantic Web challenge 2011.

WebPIE

WebPIE (Web-scale Parallel Inference Engine) [35] is a MapReduce-based parallel distributed RDFS/OWL inference engine. Being implemented as a LarKC plug-in, WebPIE can be used for materialization of an RDF graph expressed in the OWL Horst semantics, which is required by a lot of semantic reasoning workflows. The workflows that use WebPIE can take advantages of the distributed and parallel reasoning, facilitated by the underlying MapReduce implementation with Hadoop.
Thanks to the parallel implementation, WebPIE vastly outperforms all the existing inference engines when comparing supported language expressivity, maximum data size and inference speed (according to the benchmarks in [36]). In LarKC, WebPIE can easily be integrated in any forward chaining reasoning workflow and thus improve its scalability. The distributed execution framework takes care of the execution of the WebPIE reasoner on a machine that can take full advantages of the parallel realization, e.g. a cluster of workstations or a parallel supercomputer. The WebPIE research won the first scalability prize at the IEEE Scale Challenge in 2010.

GWAS
Genome-wide association study (GWAS) is a research domain aiming to identify common genetic factors that influence health and disease apparition. GWAS use bio-probes (gene markers) to look for higher levels of association between genes in a diseased subject as opposed to controls. The large numbers of markers mean that huge numbers of samples are needed to achieve sufficient statistical power. Semantic Web helps the GWAS researchers apply common statistical models to raw experimental data to find the relevance of each marker, and then rank them in order of relevance to the disease. Only the genes that are close to the top few markers are then studied in more depth by conventional techniques, to narrow the problem and achieve better results. This last bit is expensive, and improving rankings could improve both the efficiency and the economics of the technique. The WHO’s cancer research unit, IARC, has chosen LarKC as the technology to combine prior knowledge about a gene with experimental data, thus improving statistical power [37]. The modular nature of LarKC plug-ins allowed for combination of those techniques with the modern advances of the Statistical Semantics as random indexing, term frequency inverse document frequency, or term expansion using UMLS. This allowed the researchers to scale knowledge discovery across the large amounts of biomedical knowledge now encoded in the data- and bibli-ome, and to apply it to the millions of data points in a typical GWAS.

Random Indexing
Random indexing [25] is a distributional statistic technique used in resource discovery for extracting semantically similar words from the word co-occurrence statistics in the text data, based on high-dimensional vector spaces (Figure 5).

---

**Figure 5:** Schema of the co-occurrence statistical analysis of text corpora with random indexing.
Random indexing offers new opportunities for a number of large-scale Web applications performing the search and reasoning on the Web scale [26]. Prominent application using random indexing is subsetting and query expansion. Query expansion [30] is used in information retrieval with the aim to expand the document collection returned as a result to a query, thus covering the larger portion of the documents. Subsetting (also known as selection) [31], on the contrary, deprecates the unnecessary items from a data set in order to achieve faster processing. Both presented problems are complementary, as change properties of the query to best adapt it to the search needs.

The implementation as a LarKC plug-in allows random indexing to take advantages of the LarKC data and execution model, being seamlessly integrated with the other plug-ins and building up a common workflow. This allows random indexing to be coupled with reasoners to improve the resource discovery algorithm. On the other hand, the reasoning process can also benefit from the integration, for example by using random indexing to expand the initial query and improve the quality of the obtained results, such as shown in Figure 6.

![Diagram of query expansion in the Linked Life Data reasoning workflow.](image)

**Figure 6:** Implementation of query expansion in the Linked Life Data reasoning workflow.

## 5 Data-Centric Application Parallelization Technology

Although applying incomplete and interleaving approaches allows reasoning algorithms to minimize the amount of the analyzed RDF collection, the size of data remains nevertheless the major challenge for real-time analysis. For example, the random indexing algorithm, discussed in Section 4, requires vector spaces of the terabyte size range to deliver sound results. Porting to a high performance computing infrastructure is the only possibility to achieve the true scale of the Semantic Web applications. In the following we introduce some of the most popular
parallelization technologies and discuss their practical usability to cope with the big data challenge. The evaluation was done in the frame of EU-ICT project JUNIPER6.

5.1 IBIS and JavaGAT

IBIS [38] is a middleware stack used for running Java applications in distributed and heterogeneous computing environments. IBIS leverages a peer-to-peer communication technology based on a proprietary Java RMI (Remote Memory Invocation) implementation, provided by GAT (Grid Application Toolkit) [39]. The Java implementation of GAT (JavaGAT) is a middleware framework, which allows an application to instantiate Java classes remotely on the network-connected resource, i.e., on a remote Java Virtual Machine. Along with the traditional access protocols, e.g., telnet or ssh, the advanced access protocols, such as ‘ssh-pbs’ for clusters with PBS (cluster Portable Batch System)-like job scheduling or ‘gsissh’ for grid infrastructures are supported. IBIS implements a mechanism of multiple fork-joins to detect and decompose the application’s workload and execute its parts concurrently on distributed machines. While [39] indicates some successful Java applications implemented with IBIS/JavaGAT and shows a good performance, there is no clear evidence about the scalability of this solution for more complex communication patterns, involving nested loops or multiple split-joins. Whereas IBIS is a very effective solution for the distributed computing environments, e.g., Grid or Cloud, it is definitively not the best approach to be utilized on the tightly-coupled productional clusters.

5.2 Hadoop

MapReduce framework [40] and its most prominent implementation in Java, Hadoop, has got a tremendous popularity in modern data-intensive application scenarios. MapReduce is a programming model for data-centric applications exploiting large-scale parallelism, originally introduced by Google in its search engine. In MapReduce, the application’s workflow is divided into three main phases: map, process (or shuffle), and reduce. In the map stage, the input data set is split into independent chunks and each of the chunks is assigned to independent tasks, which are then processed in a completely parallel manner (process stage). In the reduce phase, the output produced by every map task is collected, combined and the consolidated final output is then produced.

Hadoop is a service-based implementation of MapReduce for Java. Hadoop considers a parallel system as a set of master and slave nodes, deploying on them services for scheduling tasks as jobs (Job Tracker), monitoring the jobs (Task Tracker), managing the input and output data (Data Node), re-executing the failed tasks, etc. This is done in a way that ensures high service reliability and fault tolerance properties of the parallel execution. In Hadoop, both the input and the output of the job are stored in a special distributed file-system. In order to improve the reliability, the file system also provides an automatic replication procedure,

6 http://www.juniper-project.org
which however introduces an additional overhead to the inter-node communication. Due to this overhead, Hadoop provides much poorer performance than the approaches network, i.e. cluster interconnect, based approaches, such as MPI discussed below. However, unlike the current network-based approaches, Hadoop offers better QoS characteristics related to the reliability and fault-tolerance. Since MPI and MapReduce paradigms have been designed to serve different purposes, it is hardly possible to comprehensively compare them. However they would obviously benefit from a cross-fertilization; e.g., MPI could serve a high-performance communication layer to Hadoop, which might help improve the performance by omitting the disk I/O usage for distributing the map and gathering the reduce tasks across the compute nodes.

5.3 Message-Passing Interface

The Message-Passing Interface (MPI) is a process-based standard for parallel applications implementation. MPI processes are independent execution units that contain their own state information, use their own address spaces, and only interact with each other via standardized communication mechanisms [41]. Every MPI process can be executed on a dedicated compute node of the high performance architecture, i.e., without competing with the other processes in accessing the hardware, such as CPU and RAM, thus improving the application performance and achieving the algorithm speed-up. In case of the shared file system, the MPI processes can effectively access the same file section in parallel without any considerable disk I/O bandwidth degradation. Each MPI process is responsible for processing the data partition assigned to it proportionally to the total number of the MPI processes (see Figure 8). The position of any MPI process within the group of processes involved in the execution is identified by an integer R (rank) between 0 and N-1, where N is a total number of the launched MPI processes. The rank R is a unique integer identifier assigned incrementally and sequentially by the MPI runtime environment to every process. Both the MPI process’s rank and the total number of the MPI processes can be acquired from within the application by using MPI standard functions. Normally, the number of MPI processes corresponds to the number of the compute nodes, reserved for the execution of parallel job. Once the MPI process is started, it can request its rank as well as the total number of the MPI processes associated with the same job. Based on the rank and total processes number, each MPI process can calculate the corresponding subset of the input data and process it. The data partitioning problem remains beyond the scope of this work; particularly for RDF, there is a number of well-established approaches discussed in several previous publications, e.g. horizontal [42], vertical [43], and workload driven [44] partitioning. Since a single MPI process owns its own memory space and thus can not access the data of the other processes directly, the MPI standard foresees special communication functions, which are necessary, e.g., for exchanging the data subdomain’s boundary values or consolidating the final output from the partial results produced by each of the processes. The MPI processes communicate with each other by sending messages, which can be done either in a “point-to-point” (between two processes) or a collective way (involving a group of or all processes).
5.4 Which Parallelization Strategy to Choose

With regard to data-centric application scenarios, parallelization allows applications to handle more data using more computational resources (cores, CPUs, nodes of a supercomputing system) by partitioning data into disjoint subsets, each of those can be processed concurrently (i.e. in parallel). Generally, choosing a concrete strategy is a very non-trivial process. There are a lot of factors influencing the choice, such as internal data structures and dependencies, communication patterns inside those data structures, availability of real-time requirements, hardware architecture’s configuration, etc.

MapReduce and MPI are two parallelization strategies that gained the most popularity for data-centric application development. However they are pretty much different. Whereas MapReduce is a key-value based parallelization strategy, offering fault-tolerance guarantees, MPI is more flexible, but also offers to gain more performance by providing a lot of low-level communication optimization functions. Generally, MPI is advantageous over MapReduce (or, to be more precisely, its most wide-spread implementation of Hadoop) in terms of performance and scalability promises, but losses in terms of flexibility and automation of the parallelization process. JUNIPER is an EU project that is investigating opportunities of both parallelization technologies for the most common data-centric application scenarios. A comprehensive comparison report on the data-centric parallelization strategies is expected to appear in the second half of 2013.

6 Conclusion

LarKC is the technology that not only enables the large-scale reasoning approach for the already existing applications, but also facilitates their rapid prototyping with low initial investments, leveraging the SOA approach through the unique platform solutions. Furthermore, LarKC delivers a complete eco-system where the researches from very different domains can team up in order to develop new challenging mashup-applications, e.g. for the resource discovery, hence having a dramatic impact on domain-specific application development. Supported by the parallelization technologies such as MPI and MapReduce, which are also explored in frame of JUNIPER, LarKC offers a promising outlook on solving the big data problem for the Semantic Web. The availability of an e-Infrastructure that will be data-centric (i.e. offer an extensive support for data management in terms of databases and parallel and streaming data processing), service-oriented (i.e. providing user- and developer friendly interfaces to access the infrastructure resources), highly-customizable (i.e. with a pluggable and extendible architecture that will be capable of meeting needs of a concrete application), and elastic with regard to the infrastructure resource pool (i.e. designed as a high-level cloud solution), is a major milestone towards the further expansion of big data applications. The proposed technology of developing data-centric applications based on LarKC and JUNIPER will surely be an important contribution toward solving
this issue; the pilot application developments discussed in this paper serve as a good prove to this statement.

Acknowledgment

This work was conducted within the JUNIPER project, partially funded by the European Commission’s Seventh Framework Programme for research and technological development (FP7).

References

Attained From The Literature To Improve Ranking In Genome-Wide Association Studies” (2009)


