Big Data to Enable Global Disruption of the Grapevine-powered Industries

D4.2 - Methods and Tools for Distributed Inference

DELIVERABLE NUMBER	D4.2
DELIVERABLE TITLE	Methods and Tools for Distributed Inference
RESPONSIBLE AUTHOR	Milena Yankova (ONTOTEXT)

Co-funded by the Horizon 2020
Framework Programme of the European Union

GRANT AGREEMENT N.	780751
PROJECT ACRONYM	BigDataGrapes
PROJECT FULL NAME	Big Data to Enable Global Disruption of the Grapevine-powered industries
STARTING DATE (DUR.)	01/01/2018 (36 months)
ENDING DATE	31/12/2020
PROJECT WEBSITE	http://www.bigdatagrapes.eu/
COORDINATOR	Pythagoras Karampiperis
ADDRESS	110 Pentelis Str., Marousi, GR15126, Greece
REPLY TO	pythk@agroknow.com
PHONE	+30 210 6897 905
EU PROJECT OFFICER	Mr. Riku Leppanen
WORKPACKAGE N. | TITLE	WP4 | Analytics & Processing Layer
WORKPACKAGE LEADER	CNR
DELIVERABLE N. | TITLE	D4.2 | Methods and Tools for Distributed Inference
RESPONSIBLE AUTHOR	Milena Yankova (ONTOTEXT)
REPLY TO	milena.yankova@ontotext.com
DOCUMENT URL	http://www.bigdatagrapes.eu
DATE OF DELIVERY (CONTRACTUAL)	30 September 2018 (M9)
DATE OF DELIVERY (SUBMITTED)	28 September 2018 (M9)
VERSION | STATUS	1.0 | Final
NATURE	DEM (Demonstrator)
DISSEMINATION LEVEL	PU (Public)
AUTHORS (PARTNER)	Milena Yankova (ONTOTEXT), Boyan SImeonov (ONTOTEXT), Atanas Kiryakov (ONTOTEXT), Vladimir Alexiev (ONTOTEXT)
CONTRIBUTORS	Nicola Tonellotto (CNR), Raffaele Perego (CNR), Raffaele Perego (CNR), Pythagoras Karampiperis (Agroknow)
REVIEWER	Antonis Koukourikos (Agroknow)

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First draft	14/08/2018	Milena Yankova (ONTOTEXT)
0.2	Final draft for review	17/09/2018	Milena Yankova (ONTOTEXT)
0.3	Comments from CNR	18/09/2018	Nicola Tonellotto (CNR), Raffaele Perego (CNR), Raffaele Perego (CNR)
0.4	Added Implementation Section	19/09/2018	Milena Yankova (ONTOTEXT)
0.7	Input from partners	21/09/2018	Nicola Tonellotto (CNR), Raffaele Perego (CNR), Raffaele Perego (CNR), Pythagoras, Karampiperis (Agroknow)
0.9	Internal Review	24/09/2018	Antonis Koukourikos (Agroknow)
1.0	Final edits after internal review	28/09/2018	Milena Yankova (ONTOTEXT), Boyan SImeonov (ONTOTEXT), Atanas Kiryakov (ONTOTEXT), Vladimir Alexiev (ONTOTEXT)

PARTICIPANTS		CONTACT
Agroknow IKE (Agroknow, Greece)		Pythagoras Karampiperis Email: pythk@agroknow.com
Ontotext AD (ONTOTEXT, Bulgaria)		Todor Primov Email: todor.primov@ontotext.com
Consiglio Nazionale Delle Ricerche (CNR, Italy)		Raffaele Perego Email: raffaele.perego@isti.cnr.it
Katholieke Universiteit Leuven (KULeuven, Belgium)		Katrien Verbert Email: katrien.verbert@cs.kuleuven.be
Geocledian GmbH (GEOCLEDIAN Germany)		Stefan Scherer Email: stefan.scherer@geocledian.com
Institut National de la Recherché Agronomique (INRA, France)		Pascal Neveu Email: pascal.neveu@inra.fr
Agricultural University of Athens (AUA, Greece)		Katerina Biniari Email: kbiniari@aua.gr
Abaco SpA (ABACO, Italy)		Simone Parisi Email: s.parisi@abacogroup.eu
APIGAIA (APIGEA, Greece)		Eleni Foufa Email: foufa-e@apigea.com

ACRONYMS LIST

BDG	BigDataGrapes
DL	Description Logic
LP	Logical Programming
OWL	Ontology Web Language
QSQ	Query-subquery
RDDs	Resilient Distributed Datasets
RDBMS	Relational Database Management Systems
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
MSC	The Most Significant Change Technique
SWRL	Semantic Web Rule Language
SPARQL	Symantec Protocol and RDF Query Language
W3C	World Wide Web Consortium

EXECUTIVE SUMMARY

The objective of this deliverable is to develop inference methods that support efficient information selection from heterogeneous data pools. There are many challenges in data reasoning and inference based on distributed data. The first one is addressing data security and access rights to both original data and inferred information. The second challenge is how the actual inference over distributed sources can be performed and implemented. We address the main principles applied to data inference and different types of inference – rule-based, query-based, model-based and fuzzy inference – and their application in BigDataGrapes project. The Final section is dedicated to state of the art with standard theoretical approach to inference from descriptive logic stand point, as well as related work in implementing those approaches.

TABLE OF CONTENTS

EXECUTIVE SUMMARY 5
1 INTRODUCTION 7
2 Types of Inference 8
2.1 Rule-based inference 8
2.2 Distributed rule-based inference 10
2.3 Query-based inference 10
2.4 Model-based inference 11
2.5 Fuzzy Inference 11
3 STATE-OF-THE-ART 12
3.1 OWL Dialects Suitable for Scalable Inference 12
3.2 Advantages of the Different Distribution Approaches 15
3.3 Related work 16
3.3.1 WebPIE by VU Amsterdam 16
3.3.2 RDFox by Oxford / Boris Motik 17
4 IMPLEMENTATION OF DISTRIBUTED INFERENCE 18
4.1 GraphDB Plugin API 18
4.2 RDF4J API extension of GraphDB 19
4.3 GraphDB remote notifications 19
5 SUMMARY 20
6 REFERENCES 21

LIST OF FIGURES

Figure 1: Rule based inference of transitive relations "is located in" 9
Figure 2: Diagram of expressivity of OWL dialects 13

1 INTRODUCTION

The objective of this deliverable is to develop inference methods that support efficient information selection from heterogeneous data pools. Further specification will enable implementation on top of the BigDataGraph database layer including a semantic graph database (a type of NoSQL graph database engine). The final goal is to enable efficient retrieval of data, considering different criteria and implementing mechanisms, which go beyond the capabilities of today’s database and search engines.

There are many challenges in data reasoning and inference based on distributed data. The most prominent one is addressing data security and access rights to both original data and inferred information. To address data security, we follow the industry business need of building the missing piece is the universal semantic data layer. Dave Mariani, co-founder and CEO of startup AtScale and former vice president of development, user data and analytics at Yahoo formulates it:

"You can define security on the data lake itself … anyone who logs in and runs queries on the data lake is going to be secured at the data bit level rather than at the application that's using it. Now data is being secured as it's written as opposed to as it's used. You can't do that if you're sending data extracts out to the business and the business is dealing with it on its own."

The second challenge of how the actual inference over distributed sources can be performed, in BigDataGrapes project we do not limit ourselves to any specific reasoning technique. Approximate reasoning is a non-standard reasoning approach based on the idea of sacrificing soundness or completeness for a significant speed-up in reasoning. This is done in such a way that the loss of correctness is at least outweighed by the obtained speed-up. Parallel reasoning and distributed reasoning are considered to be essential for Web-scale reasoning to improve scalability. Stream reasoning provides the reasoning support in which memory overload is avoided by operating on streams of data instead of statically available sets. Granular reasoning is a non-standard reasoning approach in which multiple perspectives/views can be selected for reasoning by using knowledge at various levels of specificity and data at variable levels of granularity.

We aim to explore the state of the art and construct possibly several reasoning plug-ins, based on insights from both generic inference methods and non-standard reasoning, and invite third parties to contribute further components to the BigDataGrapes ecosystem.

2 TYPES OF INFERENCE

This section addresses the main principles applied to data inference and it is an attempt for drawing a roadmap including their major characteristics, related design and performance issues, the state of the art in the field and future directions. The major objectives are:

- to clarify the principles of operation of the inference and the potential of its distribution;
- to explain the facets of their performance, because we believe that their understanding this is a key factor for the successful adoption of distributed inference.

The context in which we review types of inference and their distribution potential is addressed in one or more of the following goals:

- to handle efficiently larger volumes of data;
- to speed up the data loading and indexing and to improve the performance for updates;
- to lower the query evaluation time for complex queries (e.g. analytical Business Intelligence reports);
- to better handle concurrent query loads and large numbers of users and
- to ensure failover, e.g. to surmount failure of one or more nodes and repositories.

The reminder of this section provides discussion on the different approaches, their advantages and disadvantages and appropriateness with respect to different scenarios and goals.

2.1 RULE-BASED INFERENCE

Broadly speaking, inference can be characterized by discovering new relations (see Fig1.). On the Semantic Web, data is modeled as a set of (named) relations between resources. “Inference” means that automatic procedures can generate new relations based on the data and some additional information in the form of a vocabulary - a set of rules. Whether the new relations are explicitly added to the set of data or returned at query time is matter of implementation. Inference is a tool of choice for improving the quality of data integration by discovering new relations, automatically analyzing the content of data, or managing knowledge in general. Inference-based techniques are also important for discovering possible inconsistencies in the data.

Inference is performed by semantic repositories - database management systems - which are capable of handling structured data, taking into consideration their semantics as well as rules for interpretation. To foster their realization, the World Wide Web Consortium (W3C) developed a series of metadata, ontology, and query language standards. The standardization efforts related to the Semantic technology, most notably RDF(S), OWL, and SPARQL, provided a solid ground for development and good minimal levels of interoperability. Following the enthusiasm and the wide adoption of the related standards, today, most of the semantic repositories are database engines, which deal with data represented in RDF, support SPARQL queries, and can interpret schemata and ontologies represented in RDFS and OWL. Naturally, such engines take the role of web servers of the Semantic Technology.

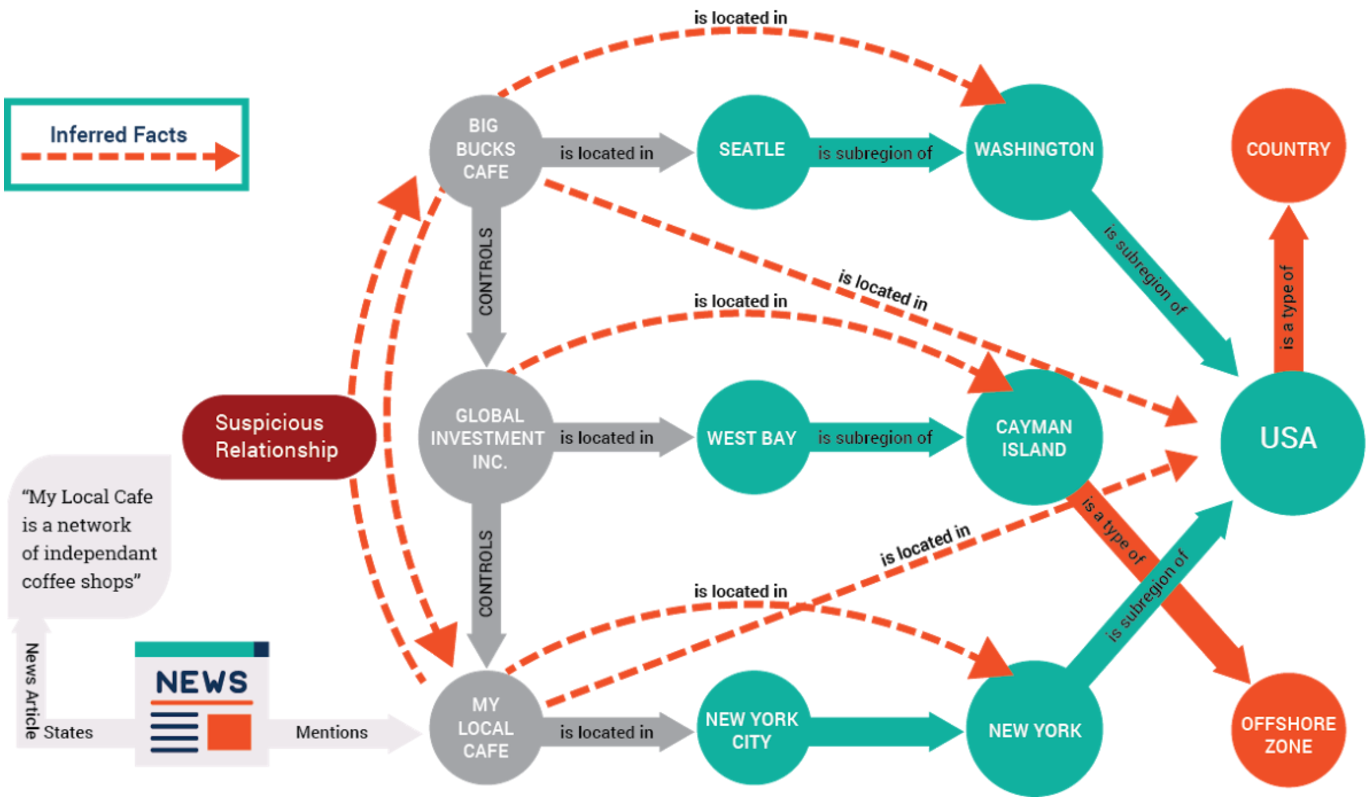

Figure 1: Rule based inference of transitive relations "is located in"

The logical inference over RDF datasets and their implementation in RDF triple stores or semantic graph databases follow one of the two principle strategies for rules application:

- *Forward-chaining*: to start from the known facts (the explicit statements) and to perform inference in an inductive fashion. Typically, the goal is to compute the Inferred Closure.
- *Backward-chaining*: to start from a particular fact or a query, and to verify it or get all possible results. In a nutshell, the reasoner decomposes (or transforms) the query (or the fact) into simpler (or alternative) facts, which are available in the knowledge base or can be proven through further recursive transformations.

The forward-chaining strategy applies the rules over the available facts in order to infer new facts, which are added to the dataset, and then recursively applies the rules over the new dataset. The result is the so-called inferred closure: an extension of a knowledge base (the RDF dataset or the graph of RDF triples) with all implicit facts (RDF triples) that can be inferred from it. The notion materialization is defined as a procedure that keeps an up-to- date inferred closure of the knowledge base.

Materialization is known as a technique for applying inference before query evaluation. This allows for many query optimization approaches to be forward-chaining as querying is realized by lookups in the database. The main drawback of materialization is that the database changes, additions, and updates are generally slow operations. In many scenarios, the materialization of such frequent changes does not affect the querying process, as many of the materialized facts are not used in the answers.

In such cases, an alternative to forward-chaining is a backward-chaining strategy for inferencing over knowledge bases. Here, answering a query requires only partial materialization over the knowledge base.

Unfortunately, backward-chaining is inefficient for large knowledge bases, as many optimizations for the materialization of a knowledge base are not possible.

The most advanced approaches to implementing hybrid reasoning for a fraction of OWL in RDF databases as presented in the work of Urbani et al (2013). They implement backward-chaining based on the QSQ (query-subquery) algorithm for Datalog databases modified to support reasoning over OWL RL. In the application of the algorithm, the facts are divided in two sets: one over which the forward-chaining is applied and the materialization over the set is stored in the semantic graph database in an optimal way. The other is used to support a backward-chaining strategy. It applies the materialization only when it is necessary.

2.2 DISTRIBUTED RULE-BASED INFERENCE

Distributed architecture and multi-threaded reasoning provide very appealing techniques for processing RDF knowledge bases consisting of an enormous amount of statements (usually several billions). The main reasoning strategy for RDF knowledge bases - forward-chaining, faces two problems: (i) maintenance of huge number of URIs, and (ii) inferring new RDF statements via inference rules applied to existing, in the knowledge base, RDF statements.

Some of the obstacles in distributing inference at scale come from the data volume. Data in the semantic representation paradigm are made of terms that are either URIs or literals. Since these terms usually consist of long sequences of characters, an effective compression technique must be used to reduce the data size and increase the application performance. In order to define a more compact representation of RDF statements, the URIs are represented in dictionaries, where each URI is identified by a numeric value, which is then used for the internal representation of the RDF statements. One of the URI terms' characteristics in an RDF knowledge base is their uneven distribution, i.e. many URI terms appear only a few times. One of the best-known techniques for data compression is dictionary encoding and MapReduce algorithm efficiently compresses and decompresses a large amount of Semantic Web data, giving a compression ratio of about 1:6 to 1:8. This compression approach allows for using parallel processing.

The expressiveness of the ontology language and complexity of the rules is another challenging area for distribution. For example, Oren et. al (2009) shows that partitioning of an RDF database into independent parts is not trivial in regard to soundness and completeness of the reasoning or results in communication overload between the different partitions.

Some authors propose additional restrictions on the language expressivity to cope with the problem. For example, Priaya et. al (2014) define an ABox independent partitioning, which supports reasoning in OWL Lite knowledge bases. Further work in this direction by Shrinoshita et. al (2017) evaluates enhanced MSC method over random graph theory that results in very small tractable concepts provided that the number of role assertions are removed from consideration is large enough.

2.3 QUERY-BASED INFERENCE

The ability to abstract the query syntax from the data syntax bears important advantages in data access scenarios where one has to deal with complex relationships or with schema diversity. As long as the semantic repositories can interpret the semantics in a recursive fashion, one can enjoy interpretations of the data, which combine results from previous interpretations and explicit assertions. In other words, depending on the data patterns and the semantics, one can retrieve facts, which are results of multiple steps of interpretation, and this way to uncover relationships which would otherwise remain hidden.

The standardized way of distributed query inference is to use SPARQL 1.1 Federated Query extension for executing queries distributed over different SPARQL endpoints. The SERVICE keyword extends SPARQL 1.1 to support queries that merge data distributed across the Web, and the inference should follow the backward-chaining strategy implemented on the query level. This feature is very powerful and allows integration of RDF data from different sources using a single query. It is also possible to use the federation mechanism to do distributed querying over several repositories on a local server for managing security on data level.

The query-level inference is the most expressive mechanism for inference as it can use the full power of SPARQL and for defining rules (as SELECT) statements with filtering and exceptions. It can include custom functions and potentially wrap complex machine learning models as well.

2.4 Model-based inference

Scientists derive insights from models of complex systems by applying the models to address various types of prognostic queries. This can include, for example:

- Prediction: How will the system evolve in the near future?
- Conditional forecasting: How will the system respond if X changes?
- Counterfactual analysis: What would have happened if X had been Y?
- Comparative impact: What is the difference in utility between strategy X and strategy Y?
- Optimal planning: What is the optimal amount of X to introduce to maximize utility Y?
- Risk assessment: What is the risk of X?
- Outcome avoidance: What is the optimal action or intervention to reduce the risk of X decreasing more than Y?

Model-based inference can also be used diagnostically to test models against available data or knowledge through model checking, validation, and calibration. Automation of model-based inference procedures could increase the speed and accuracy with which these models can be used to address key questions of national security by orders of magnitude. Applications will include frequent update of user-specified queries as new data becomes available, rapid response to emerging natural disasters or other real-time threats, and even fully automated inference with machine-generated queries.

Model-based inference is predominantly based on machine learning techniques and depend very much on the available data features. As part of the initial research in BigDataGrapes will explore the available data sets in deliverable *D2.1 Use Cases & Technical Requirements Specification* before drawing any conclusions on the relevant techniques. This work is closely related to *D4.3 Methods and Tools for Scalable Distributed Processing*.

2.5 Fuzzy Inference

Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic. It is classically applied in Fuzzy control systems to formalize the reasoning process of human language by means of fuzzy logic. It uses the "IF…THEN" rules along with connectors "OR" or "AND" for drawing essential decision rules.

Although alternative approaches such as genetic algorithms and neural networks can perform just as well as fuzzy logic in many cases, fuzzy logic casts to terms that human operators can understand and makes it easier to automate tasks that are already successfully performed by humans. State of the art implementation of Fuzzy Inference is provided by Mathworks.

It is still unclear if Fuzzy logic can be applied to any of the use cases in BigDataGrapes. Such decision can be made based on deliverable D2.1 Use Cases & Technical Requirements Specification and initial experiments using actual data provided by the use case partners.

3 STATE-OF-THE-ART

3.1 OWL Dialects Suitable for Scalable Inference

In order to match the expectations for the next generation global Web of data, the Semantic Web requires scalable high-performance storage and reasoning infrastructure. One challenge towards building such an infrastructure is the expressivity of its schema and ontology definition standards RDFS and OWL. RDFS (Brickley and Guha, 2004) is the schema language for RDF, which allows for the definitions of subsumption hierarchies of classes and properties; the latter being binary relationships defined with their domains and ranges. While RDFS is generally a fairly simple knowledge representation language, implementing semantic repositories which support its semantics and provide performance and scalability comparable to those of relational database management systems (RDBMS) is very challenging.

The semantics of RDFS is based on Logical Programming (LP) – a declarative programming paradigm, in which the program specifies a computation by giving the properties of a correct answer. The LP languages like PROLOG emphasize the logical properties of a computation, using logic and proof procedures to define and resolve problems. Most logic programming is based on the Horn-clause logic with negation-as-failure to store the information and rule entailment to solve problems. Datalog is a query and rule language, a simplified version of PROLOG, meant to enable the efficient implementation of deductive databases. The semantics of RDFS is defined by means of rule entailment formalism, which is a simplification of Datalog.

OWL[1], (Dean and Schreiber, 2004) is an ontology language, which supports more comprehensive logical descriptions of the schema elements (see Fig.2), for instance: transitive, symmetric, and inverse properties; unions and intersections of classes; and property restrictions. The first version of the OWL specification, which was published as W3C standard in year 2004 has three dialects: OWL Lite, OWL DL and OWL Full. They range in their levels of expressivity. OWL Lite is a subset of OWL DL, and OWL DL is a subset of OWL Full. The OWL language is based on description logics (Baader et al, 2003).

The reasoning procedures of DLs are decision procedures that are aimed to always terminate – in mathematical logic terms this means that DLs are decidable. Compared to other logical languages DLs are relatively inexpressive. Still reasoning with DLs is based on satisfiability checking, which means that, in order to prove or to reject a specific statement, a DL reasoner needs to check whether it is possible or not to build a model of the world which satisfies a "theory" which includes this statement or its negation. For instance, suppose that there is a semantic repository which contains one billion statements and a client makes a query, checking whether specific resource is an instance of a specific class. In order to validate this, with respect to the semantics of OWL DL, a repository should add to its current contents the statement that the resource is not instance of the class and check whether the new state of the repository is consistent. It is clear that such semantics is impractical to implement for large volumes of data. Even the simplest dialect of OWL, OWL Lite is a DL formalism which does not support algorithms enabling efficient inference and query answering over reasonably large knowledge bases.

Logic programming and description logics support semantics and data interpretation capabilities of a different nature: LP uses rules to infer new knowledge, whereas DL employ descriptive classification mechanisms. None of these is more powerful or expressive than the other one – there are meaning aspects which can be expressed in each one of them, which cannot be expressed in a language from the other paradigm. As result, the semantics of OWL Lite and DL are incompatible with that of RDFS[2]. Although OWL was meant to be layered on top of RDFS in the Semantic Web specification stack, there is no “backward compatibility”. In practical terms, this means that it may be impossible to “upgrade” an application that uses RDFS schemata to OWL, without replacing the schemata with OWL ontologies. The latter may require considerable changes in the semantics of the classes and the properties and in the data modeling principles used in the application.

To bridge the gap of expressivity, compatibility and logical decidability and reach the goals of scalable inference, other dialects of OWL have been created which lay between RDF(S) and OWL Lite. 0 presents a simplified map of the expressivity or complexity of a number of these OWL-related languages together with their bias towards description logic and logical programming based semantics. The diagram provides a very rough idea about the expressivity of the languages, based on the complexity of entailment algorithms for them. A direct comparison between the different languages is impossible in many of the cases. For instance, Datalog is not simpler than OWL DL, it just allows for a different type of complexity.

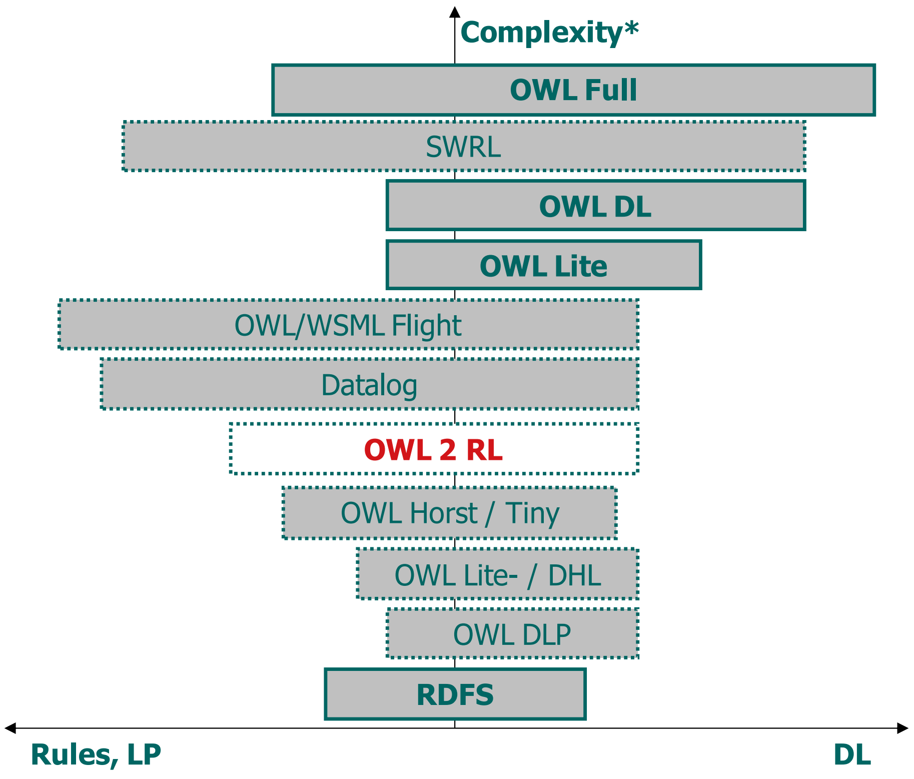

Figure 2: Diagram of expressivity of OWL dialects

OWL DLP is a non-standard dialect, offering a promising compromise between expressive power, efficient reasoning, and compatibility. It is defined in "Description Logic Programs: Combining Logic Programs with Description Logic" (Grosof et al, 2003) as the intersection of the expressivity of OWL DL and logical programming . In fact, OWL DLP is defined as the most expressive sub-language of OWL DL, which can be mapped to Datalog. OWL DLP is simpler than OWL Lite. The alignment of its semantics to the one of RDFS is easier, as compared to the Lite and DL dialects. Still, this can only be achieved through the enforcement of some

[2] The issues related to the interoperability and layering of the Semantic Web languages is also discussed in the introductory Chapter 1.

additional modeling constraints and transformations. A broad collection of information related to OWL DLP can be found in "Ontology Logic and Reasoning at Semantic Karlsruhe"[3]. DLP has certain advantages:

- There is freedom to use either DL or LP (and associated tools and methodologies) for modeling purposes, depending on the modeler's experience and preferences.
- From an implementation perspective, either DL reasoners or deductive rule systems can be used. Thus, it is possible to model using one paradigm, e.g. a DL-biased ontology editor, and to use a reasoning engine based on the other paradigm, e.g. a semantic repository based on rules.

These features of DLP provide extra flexibility and ensure interoperability with a variety of tools. Experience with using OWL has shown that existing ontologies frequently use only very few constructs outside the DLP language.

Ter Horst (2005) defines RDFS extensions towards rule support and describes a fragment of OWL, more expressive than OWL DLP. He introduces the notion of R-entailment of one (target) RDF graph from another (source) RDF graph on the basis of a set of entailment rules R. R-entailment is more general than the D-entailment used by Hayes (2004) in defining the standard RDFS semantics. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are "extended" RDF statements, where variables can take any of the three positions. The head of the rule comprises one or more consequences, each of which is, again, an extended RDF statement. The consequences may not contain free variables, i.e. which are not used in the body of the rule. The consequences may contain blank nodes.

The extension of the R-entailment (as compared to the D-entailment) is that it "operates" on top of the so-called generalized RDF graphs, where blank nodes can appear as predicates. R-entailment rules without premises are used to declare axiomatic statements. Rules without consequences are used to imply inconsistency.

This extension of RDFS became popular as "OWL Horst". As outlined in "Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity" (ter Horst, 2005) this language has a number of important characteristics:

- It is a proper (backward-compatible) extension of RDFS. In contrast to OWL DLP, it puts no constraints on the RDFS semantics. The widely discussed meta-classes (classes as instances of other classes) are not disallowed in OWL Horst.
- Unlike the DL-based rule languages, like SWRL (Horrocks et al, 2005), R-entailment provides a formalism for rule extensions without DL-related constraints;
- Its complexity is lower than the one of SWRL and other approaches combining DL ontologies with rules of "Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity" (ter Horst, 2005).

OWL Horst is supported by GraphDB and ORACLE, which makes it the OWL dialect that has the largest industry support. An official OWL dialect with the same properties emerged recently under the name OWL 2 RL. The latter is one of the tractable profiles (dialects) defined in the specification of OWL 2 (Motik et al, 2009) – the next version of the OWL language that is currently in process of standardization. OWL 2 RL is designed with the objective to be the most expressive OWL dialect which allows for efficient reasoning with large volumes of data in rule-based systems. OWL 2 RL was inspired by OWL Horst – its semantics is defined with rule language

[3] https://ieeexplore.ieee.org/document/7072815/

equivalent to R-entailment. However, OWL 2 RL is considerably more expressive than OWL Horst. Support for OWL 2 RL is provided by several reasoning engines, including GraphDB and ORACLE.

Recent research reported in "UniProt in RDF: Tackling Data Integration and Distributed Annotation with the Semantic Web" (Redaschi and the UniProt Consortium, 2009) evaluates the level of completeness of the inference supported by few inference engines (namely HAWK) and semantic repositories: IBM's Minerva, Sesame and GraphDB by ONTOTEXT. It demonstrates that although GraphDB supports sufficient reasoning to answer the LUBM[4] queries correctly, it is still not complete with respect to the semantics of the data and the queries.

3.2 Advantages of the Different Distribution Approaches

The general approaches for distribution of database management systems are:

- Data partitioning, where the information stored and accessed by the system is spread across multiple machines, so that none of them contains the entire dataset;
- Data replication, where the entire dataset resides on each of the machines.

Data replication is a traditional approach for boosting the read performance of a DBMS at the cost of redundancy and write propagation complexity. In a classic scenario, several slave nodes are assigned incoming read requests by a central master node that performs any sort of load balancing (e.g. round robin) to distribute the load evenly across the slaves. Writes are executed on the master node and updates are propagated to the slaves in the background. Such a setup is very appropriate in situations when a lot of read requests occur while write requests are rare or clustered together in large batches (for example if a large dataset is initially loaded in the repository). In such situations the resource-intensive replication procedure will not be necessary most of the time, while a theoretically linear scalability will take place on the reading side.

While data partitioning looks as the more promising schema, it is also the one which is most problematic to implement. In general, it enables the management of larger volumes of data and provides more space for in-memory data-structures. Each node can apply more efficient caching and optimization with respect to the fraction of the data that it deals with. Data partitioning with redundancy also allows for failover support. Still, the major issue is that query evaluation against distributed data requires intensive communication between the nodes for exchanging of intermediate results; the most common variety of such communication is known as "remote join". Query optimization schemata, which consider the communication costs, are far harder to implement, which triggers less-optimal query evaluation plans and larger overall numbers of index lookups. In large number of scenarios these effects neutralize the gains from the additional computing power gained from several machines. The same concerns are application for rule-based reasoning in repositories using data partitioning.

Two approaches to data partitioning appear in database systems from the established distributed DBMS research: horizontal and vertical partitioning. The horizontal data partitioning approach partitions a dataset across several repository nodes where no schema limitations apply to any of the nodes. A vertical partitioning approach would assign different parts of the data schema to different repository nodes, so, that later on requests for any type of data would be redirected to the respective repository node. This approach can be further extended and types of data that usually appear "close" together can be placed within a single node (when possible). In principle, such clustering would allow for whole sub-queries to be

[4] LUBM benchmark is introduced in D7.1

executed within a single node. It would therefore avoid the transfer of intermediate results between the repository nodes and the central query processing node only to complete the query.

As an overview of the two major distribution approaches we can summarize that:

- Data partitioning improves data scalability, however in most of the cases hampers the query evaluation performance due to high communication overheads. It can improve loading performance if there is no materialization involved;
- Data replication allows for better handling of concurrent query loads and failover. It is neutral with respect to loading and inference performance.

None of these approaches provides a principal advantage for evaluation of complex queries. Under data replication one of the nodes can be off-loaded from concurrent queries, which would allow faster execution of a complex query. An approach known as "federated join" can in theory improve the performance of such queries in very specific data partitioning scenarios, where the communication costs can be minimized.

3.3 Related work

3.3.1 WebPIE by VU Amsterdam

"WebPIE (Web-scale Parallel Inference Engine)[5] is a MapReduce distributed RDFS/OWL inference engine written using the Hadoop framework. This engine applies the RDFS and OWL ter Horst rules and it materializes all the derived statements."

It's a stream of work starting from MSc thesis of Urbani (2009) supervised by Frank van Harmelen. Their notable achievement is performing materialization of RDFS on 100B statements using notable cluster of machines with high-speed connectivity funded as part of LaRCK[6] project.

The major concern about their approach is regarding the manually optimization of Map Reduce rules in a very special manner in order to avoid the pitfalls of remote joining, therefore implementing full OWL 2 RL this way is unfeasible.

The more-recent re-implementation of WebPIE (Kim and Parkis, 2015) based on SPARK with its Resilient Distributed Datasets (RDDs). They achieve lower scale (less than 1B) and proud of the fact that they doubled the speed 5 years later.

What evidence this provides for our deliverable:

- Distributed reasoning is only possible for logical languages with very limited expressivity
- Even for such languages, it requires tailor made inference flows crafted by human experts
- Although the results demonstrated in experimental setting were very impressive in terms of scalability, such systems appeared impractical to exploit in enterprise setting.
- This is way the project was abandoned and none of the commercial RDF engines with reasoning support adopted it.

[5] http://few.vu.nl/~jui200/webpie.html
[6] https://cordis.europa.eu/project/rcn/85416_it.html

3.3.2 RDFox by Oxford / Boris Motik

"RDFox[7] is a highly scalable in-memory RDF triple store that supports shared memory parallel datalog reasoning behind the founded in 2017 Oxford Semantic Technology[8] as university spin-off which aims to commercially exploit the technology. The most recent general article about the system (Kim and Park , 2015) is co-author by Zhe Wu – the architect of ORACLE's RDF support; ORACLE also have parallelized, but not distributed inference.

RDFox is not a fully-fledged triplestore as discussed in "RDFox: A Highly-Scalable RDF Store" (Motik et al, 2015), it still does not have full SPARQL support but uses owl:sameAs optimization and incremental delete of inferred statements.

Note: this is parallel inference, not distributed inference system.

What evidence this provides for our deliverable:

- Parallel inference is feasible and can speed up reasoning substantially, subject to very specific and careful implementation of lock-free data structures with low-level programming. This is why it is written in C++, not in Java
- Parallel inference is way easier to implement than distributed reasoning. The core reason of this is that there is no data sharding and no "remote join" problem. In 2015 the RDFox team declared future work plans for implementation of distributed share-nothing reasoning system. 3 years later they haven't published any results in this direction

MULTI SENSOR write up on reasoning

Multisensor deliverable 5.4 (Simeonov et al, 2016) reports relevant results on parallel inference and provides evidence that a general-purpose commercial data store parallelization can speed up inference at least twice, both for small but tangled dataset like Wordnet and for 1B triples knowledge graph. Datasets used in Multisensor combine DBPedia, Bablenet, statistics and other business information and the implementation of parallel inference has become part of GraphDB since version 7.2. However, the practical results of GraphDB 8.x[9] show that on dataset like the one of LDBC SPB, the speed up is minimal, because in SPB's knowledge graph provides owl:sameAs mappings the reasoning implications of which limit the applicability of some of the parallelization techniques.

[7] https://cs.ox.ac.uk/isg/tools/RDFox/
[8] https://www.oxfordsemantic.tech/
[9] https://ontotext.com/products/graphdb/benchmark-results/

4 IMPLEMENTATION OF DISTRIBUTED INFERENCE

In BigDataGrapes project we envision using GraphDB by ONTOTEXT as main semantic graph database. GraphDB is a highly-efficient and robust graph database with RDF and SPARQL support. GraphDB uses RDF4J[10] as a library, taking advantage of its APIs for storage and querying, as well as the support for a wide variety of query languages (e.g., SPARQL and SeRQL) and RDF syntaxes (e.g., RDF/XML, N3, Turtle).

The development of GraphDB is partly supported by SEKT[11], TAO, TripCom[12], LarKC[13], and other FP6[14] and FP7[15] European research projects[16].

Distributed inference engine for BigDataGrapes will be implemented as a set of external to GraphDB engines which use the APIs of the database to access the data in real time and synchronize inference indexes, power inference algorithms and provide provenance of the newly inferred fact.

4.1 GraphDB Plugin API

The most powerful access mechanism to GraphDB and its data layer is via GraphDB Plugin API. It is a framework and a set of public classes and interfaces that allow developers to extend GraphDB and build custom inference mechanism over distributed data space. These extensions are bundled into plugins, which GraphDB discovers during its initialisation phase and then uses to delegate parts of its query processing tasks. The plugins are given low-level access to the GraphDB repository data, which enables them to do their job efficiently. They are discovered via the Java service discovery mechanism, which enables dynamic addition/removal of plugins from the system without having to recompile GraphDB or change any configuration files.

Plugin API can be effectively used to modify, filter or enrich the final results of a request which is particularly suitable to query-based distributed inference. For each binding set that is to be returned to the GrapHDB client the implemented plugin modifies the binding set and return it. After a binding set is processed by a plugin, it is passed to the next plugin that has enabled post-processing. Finally, after all results are processed and returned, each plugin serves modified result set to the client.

Alternatively, Plugin API can be used for custom inference as it allows for processing of data update statement (forward-chaining inference) as it looks for patterns containing specific predicates. It works as during initialization the plugin register the predicates it is interested in and then GraphDB filters updates for statements using these predicates and notifies the plugin. Filtered updates are not processed further by GraphDB, but the particular implementation of the Plugin API must handle insert or delete operation.

Further documentation of how to implement Plugin API and provide plugin configuration can be found in GraphDB documentation[17].

[10] http://rdf4j.org/about/
[11] http://www.sekt-project.com/
[12] http://www.tripcom.org/
[13] http://www.larkc.org/
[14] http://cordis.europa.eu/fp6/
[15] http://cordis.europa.eu/fp7/
[16] http://ontotext.com/knowledge-hub/
[17] http://graphdb.ontotext.com/documentation/standard/plug-in-api.html#making-a-plugin-configurable

4.2 RDF4J API EXTENSION OF GRAPHDB

Programmatically, GraphDB can be used via the RDF4J[18] Java framework of classes and interfaces. Documentation for these interfaces (including Javadoc[19]). Code snippets in the following sections are taken from (or are variations of) the developer-getting-started examples, which come with the GraphDB distribution.

RDF4J comprises a large collection of libraries, utilities and APIs. The important components for this section are:

- the RDF4J classes and interfaces (API), which provide a uniform access to the SAIL components from multiple vendors/publishers;
- the RDF4J server application.

With RDF4J 2, local repository configurations are represented as RDF graphs. To access remote repositories RDF4J server provides a Web application that allows interaction with repositories using the HTTP protocol. It runs in a JEE compliant servlet container, e.g., Tomcat, and allows client applications to interact with repositories located on remote machines. In order to connect to and use a remote repository, local repository manager must be replaced with a remote one.

The RDF4J HTTP server is a fully fledged SPARQL endpoint - the RDF4J HTTP protocol is a superset of the SPARQL 1.1[20] protocol. It provides an interface for transmitting SPARQL queries and updates to a SPARQL processing service and returning the results via HTTP to the entity that requested them.

4.3 GRAPHDB REMOTE NOTIFICATIONS

Remote notifications are powerful mechanism for maintaining distributed inference, where changes in the remote instances notify the inference engine for changes on data level which affect the inferred facts and indexes. GraphDB's remote notification mechanism provides filtered statement add/remove and transaction notifications for both local or remote GraphDB repositories. Subscribers for this mechanism use patterns of subject, predicate and object (with wildcards) to filter the statement notifications to only actionable ones in order to increase inference performance.

Apart from the native GraphDB notifications[21], RDF4J API provides such a mechanism implementing RepositoryConnectionListener which can be notified of changes in a NotifiyingRepositoryConnection. However, the GraphDB notifications API works at a lower level and uses the internal raw entity IDs for subject, predicate and object instead of Java objects. The benefit of this is that a much higher performance is possible. The downside is that the client must do a separate lookup to get the actual entity values and because of this, the notification mechanism works only when the client is running inside the same JVM as the repository instance.

[18] http://rdf4j.org/about/
[19] http://docs.rdf4j.org/javadoc/latest/
[20] http://www.w3.org/TR/sparql11-protocol/
[21] http://graphdb.ontotext.com/documentation/standard/notifications.html

5 SUMMARY

Inference distribution is a topic that has been studied previously, however the reported results vary in terms of applicability to wide range of use cases. Reviewed literature, general methodology and individual tools for both distribution and parallelization provide valuable insights of possible options for applying distributed inference to BigDataGrapes project use-cases.

Next steps will include pragmatic evaluation of the use case scenario, the level of complexity and distribution of related datasets and need for particular type of inference. This will allow us to experiment with state-of-the-art tools and draw the roadmap for future implementations using one of the mechanisms for extending GraphDB build-in inference capabilities.

6 REFERENCES

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Scheider, P. (2003). The Description Logic Handbook. Theory, Implementation, Applications, Cambridge University Press.

Brickley, D. and Guha, R. (2004). Resource Description Framework (RDF) Schemas. W3C Recommendation.

Dean M. and G. Schreiber, "OWL Web Ontology Language Reference," W3C Recommendation, 2004.

Grosof, B., Horrocks, I., Volz, R. and Decker, S. (2003).Description Logic Programs: Combining Logic Programs with Description Logic. In WWW2003, Budapest, Hungary.

Hayes, P. (2004). RDF Semantics

Horrocks, I., Patel-Schneider, P., Bechhofer, S., and Tsarkov, D. (2005). OWL Rules: A Proposal and Prototype Implementation. Journal of Web Semantics, pp. 23-40.

Kim J. M. and Park, Y.T. (2015). Scalable OWL-Horst ontology reasoning using SPARK. [Online]. Available: https://ieeexplore.ieee.org/document/7072815/.

Motik, B., Nenov, Y., Piro, R., Horrocks, I., Wu, Z. and Banerjee, J. (2015). RDFox: A Highly-Scalable RDF Store. In ISWC 2005.

Motik, B., Nenov, Y., Piro, R., Horrocks, I. and Olteanu, D. (2014).Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In 28th AAAI Conf. on Artificial Intelligence (AAAI 2014).

Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A. and Lutz, C. (2009). OWL 2 Web Ontology Language Profiles. W3C Candidate Recommendation.

Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., Teije, A. and van Harmelen, F. (2009). MARVIN: A platform for large scale analysis of Semantic Web data. In Proceedings of the WebSci'09: Society On-Line.

Priya, S., Guo, Y., Spear, M. and Heflin J. (2014). Partitioning OWL Knowledge Bases for Parallel Reasoning, IEEE, p. 108–115.

Redaschi N. and the UniProt Consortium (2009). UniProt in RDF: Tackling Data Integration and Distributed Annotation with the Semantic Web. S. I. o. Bioinformatics. [Online]. Available: http://precedings.nature.com/documents/3193/version/1/files/npre20093193-1.pdf.

Shironoshita, P., Zhang, D., Kabuka, M. and Xu, J. (2017).Parallelization of Query Processing over Expressive Ontologies, in CEUR Workshop Proceedings.

Simeonov, B., Alexiev, V., Simov, K. and Kotsev, V. (2016). Final semantic infrastructure and final decision support system. Deliverable D5.4. MULTISENSOR.

Ter Horst, H. (2005). Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In ISWC 2005, Galway, Ireland.

Urbani, J., Margara, A., Jacobs, C., van Harmelen, F. and Bal, H. (2013). DynamiTE: Parallel Materialization of Dynamic RDF Data. In: Alani H. et al. (eds) The Semantic Web, Berlin.

Urbani, J. (2009). RDFS/OWL Reasoning Using MapReduce Framework.

Urbani, J., Piro, R., van Harmelen, H. and Bal, H. (2013). Hybrid reasoning on OWL RL, Semantic Web Journal, vol. 5, no. 6, pp. 423-447.