bpifrance

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

LES PÔLES DE COMPÉTITIVITÉ
MOTEURS DE CROISSANCE ET D'EMPLOI

Projet WAVES : Des flux de données brutes et hétérogènes à l'information qualifiée

N° du contrat F1411006 QDate de début 2 juin 2014Durée 36 mois

Livrable D3.1
Semantization, Storage and Reasoning over RDF Streams

Atos

suez environnement

Ondeo Systems

GASPARD MONGE INSTITUT

isep

DATA PUBLICA

Statut

Niveau dissémination	Publique
Date d'échéance	Mois 8, ??/??/2017
Date de soumission	Mois 8, ??/??/????
Work Package	3
Tâche T?	titre
statut d'approbation	Draft
Version	0.1
Nombre de Pages	1
Nom du fichier	D?

2 Historique

Version	Date	Revu par
0.0	-	-
0.1	-	-

3 Auteurs

Organisation	Nom	Contact
UPEM - LIGM	Jérémy Lhez	jeremy.lhez@u-pem.fr
UPEM-LIGM	Olivier Curé	olivier.cure@u-pem.fr

4 Résumé

Ce livrable correspond à un état de l'art sur les thèmatiques de la tâche 3 du projet WAVES, c'est-à-dire la sémantisation, le stockage et le raisonnement de flux RDF. L'objectif du document est donc d'identifier, de décrire et d'évaluer des travaux et systèmes dans ces trois domaines. Après une introduction situant les opérations de la tâche 3 dans le contexte de WAVES, nous dédions une section à chacune de ces trois sous-tâches.

N.B.: le reste du document est rédigé en anglais.

Contenu

5 Introduction

This deliverable is part of WAVES's Task 3 which is dedicated to semantizing and reasoning over heterogeneous data streams as well as storing RDF triples, quads (standard triple with a graph name or a temporal dimension) and 5-tuples (standard triple with both a graph name and a temporal dimension). Figure 1 highlights the impact each of those operations have in the overall WAVES processing workflow. The purpose of this document is to provide a state of the art on these different operations. This approach should help in designing an efficient system by learning from conducted research, by using some existing methods and open source systems.

In this deliverable, we consider that the data collection process is supported by a data provider, e.g.,Ondeo-Suez. These data are obtained from some sensors and are formalized using a certain format, e.g., CSV, XLS, XML, JSON. In order to be integrated within the system, they need to be transformed into RDF triples (quads or 5-tuples). During this transformation step, some filters or sampling operations can be performed. We consider that the characteristics of the input sensors are known in advance and that a set of rules are established to support these transformations. These rules may use some static ontologies which are stored in a RDF store. Using these ontologies, a set of reasoning tasks are executed to detect anomalies and to support decision taking.

In the remaining of this document, the semantization, storage and reasoning aspects are each considered in different sections. Note that due to the innovant character and high activity of these research fields, this document is expected to evolve during the three years span of the project.

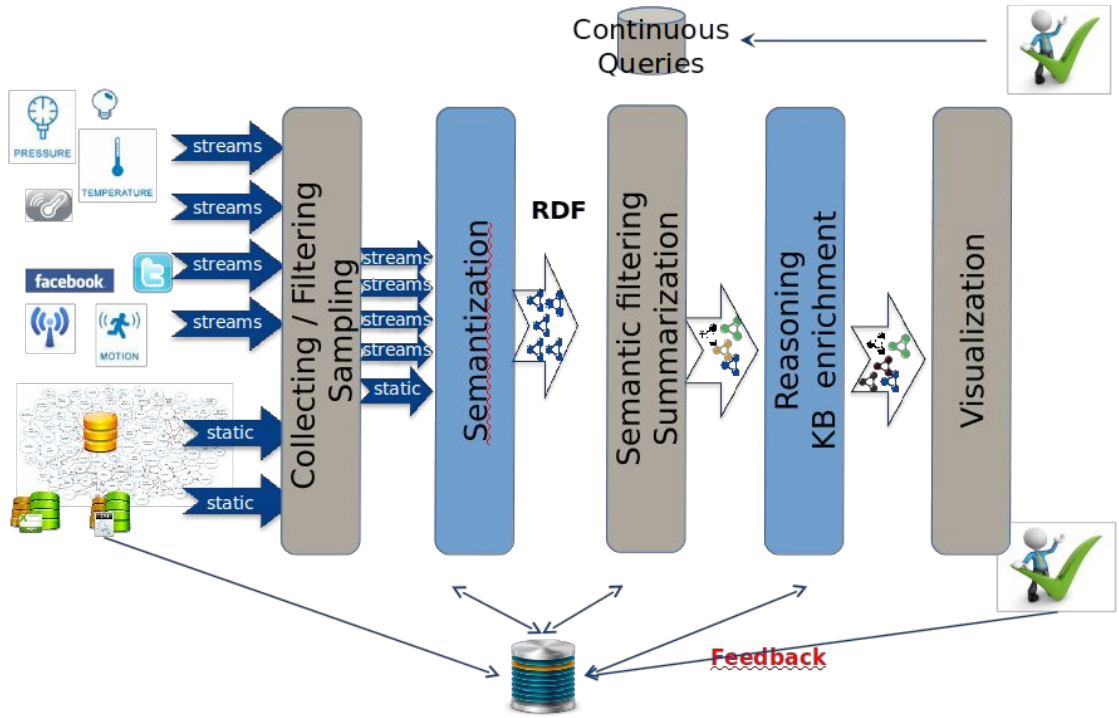

Figure 1: Waves processing steps

6 Semantization

6.1 Introduction

Semantization aims to provide some semantics to the different data items collected from sensors and transformed during a first stage, namely the “collecting, filtering and sampling” box of Figure 1. Hence its objective is to relate data sources to some ontologies. These ontologies are stored in an RDF repository corresponding to the database icon at the bottom of Figure 1 (and which is the subject of the next section). This semantization task is related to the work on schema and ontology matching and mapping [5], [13]. Nevertheless, certain aspects of WAVES ensure that we do not need the complete machinery of standard matching/mapping systems. In particular, in WAVES, we can assume the following aspects:

• the input source is represented as n-tuples (with n=3, 4 or 5). In contrast, in schema and ontology matching, different data models are considered as the input, e.g., relational databases, XML documents, CSV, XSL.

• the kind of information stored in our n-tuples can be quite numerical and specific to a scientific domain, , e.g., pH or pressure measures. This aspect makes the data linking to general purpose, Open data ontologies difficult or impossible. That is, we will need to access

ontologies specified in the contest of the project. For instance, in our potable water use case, we will develop ontologies on the different kind of sensors Ondeo Suez is using. It will contain information such as the brand, the model, precision, autonomy, etc. of the sensors.

A common characteristic of schema/ontology matching and mapping is the semi-automatic nature of this running process. That is human interactions are required to produce high quality results. This applies to our semantization solution where a domain expert needs to provide some mapping assertions between the schema of an input document and a set of ontologies. In order to perform these operations semi-automatically, we are interested in tools that enable to map ontology concepts and properties to attributes of a given schema. In the next section, we present some existing tools.

6.2 Systems

6.2.1 RDF Refine

Open Refine (formerly Google Refine) is defined by its creators team as a ”tool for working with messy data, cleaning it up, transforming it from one format into another”[1] . This system runs as a Web application, i.e., in a Web browser. This enables for a user-friendly experience with the system. Intuitively, the end-user can correct and transform data by directly interacting with a spreadsheet-like presentation. Some simple statistics on presented data guide the user through the cleaning process. It is also possible to link some data columns to a database like Freebase. Google Refine can handle several data formats that can be represented in a tabular form but is not adapted to link RDF data to external ontologies such as those of the LOD.

[1] https://code.google.com/p/google-refine/

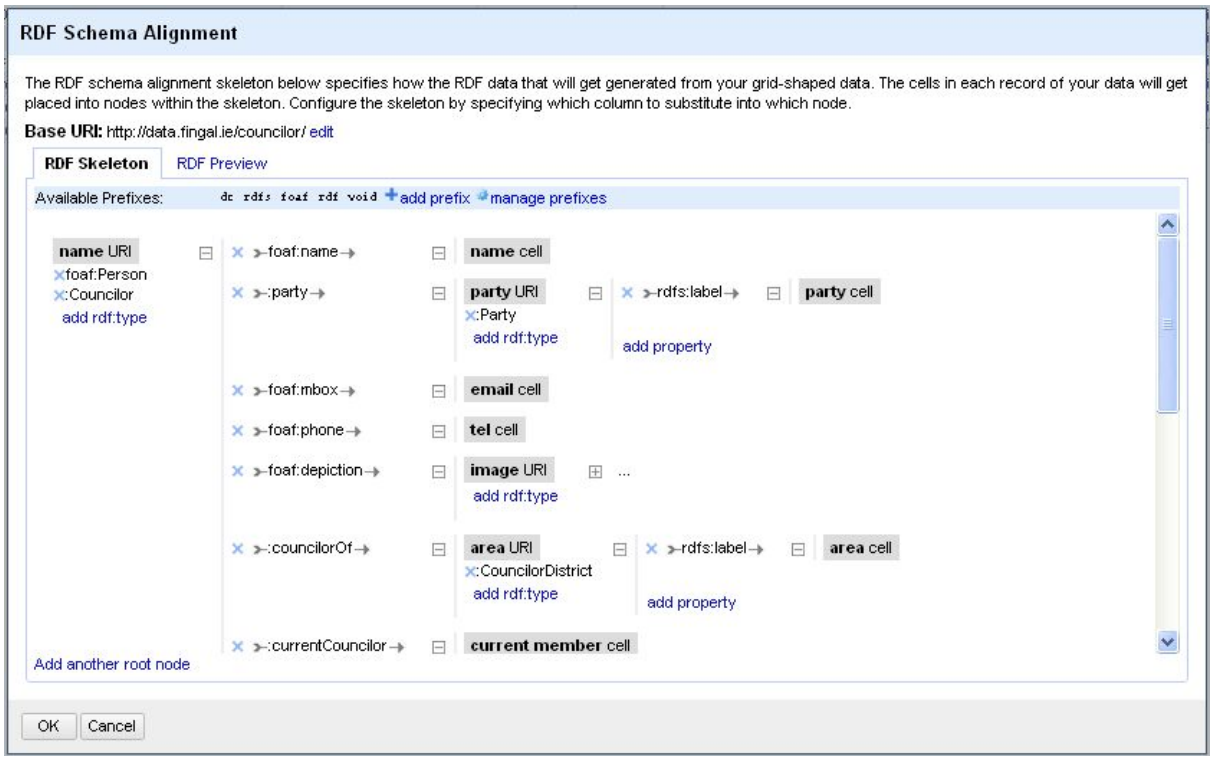

Figure 2: RDF refine schema alignment screenshot

The RDF Refine [27] tool is an open-source, free extension of Google Refine. The tool is developed at DERI and also runs as a Web application (Figure 2). It proposes to produce RDF data from different data files (TSV, CSV, JSON, excel, XML, RDF as XML). Several interesting features are present to support this transformation. One can point to or import an ontology or use one of the pre-loaded ontologies, e.g., RDFS, OWL, FOAF, to map attributes of the spreadsheet (generally the first row) to ontology properties. It is then possible with just on click to generate an RDF document (in Turtle or RDF/XML).

The possibilities of RDF Refine go beyond this simple mapping solution. Reconciliation enables to point to a knowledge source, e.g.,, SPARQL endpoint, and select a property of that source for an input attribute. The system then searches for correspondences of the input data with entries in the selected knowledge base. This approach enables to enrich the original dataset with links to external knowledge base,e.g., DBPedia. In case, of an incomplete reconciliation, the end-user can use the cleansing facility which based on some Open Refine recommendations helps in correcting the data input.

RDF Refine is a user-friendly Web application that benefits from many of the features of Open Refine. With a fast learning curve, it enables one to be rapidly productive in transforming some tabular data into RDF triples. The reconciliation functionality enables to enrich the a given data source with links to external knowledge bases. The system addresses a one-shot transformation and enrichment approach. This does not correspond to our

stream processing concern where one may define a map and use those mapping assertions to transform thousands of instance files as they arrive in the system.

6.2.2 Karma

Karma [24] is developed as an open source project at the ISI lab of the University of South California. It maps structured data to RDF according to an ontology an end-user has selected. In [24], this system is used to map 41,000 objects of the Smithsonian American Art Museum to DBPedia and Getty vocabularies. Compared to RDF Refine, Karma automates several steps of the mapping process using a statistical modeling method such as Conditional Random Field (CRF)[26]. For instance, given an ontology and an input data set, the system proposes some mapping assertions which can be validated or adjusted through interactions in the visual interface (Figure 3. The system involves two semiautomatic steps: semantic type assignment and specification of the relationships between semantic types. A Semantic type can be an OWL class or the range of a data property. Once the model of a data input is completed, it can be published as RDF triples or stored in a relational database. In the context of the WAVES project, Karma has more or less the same limitations as RDF Refine. That is, it corresponds to a one-shot data mapping and transformation solution and is not adapted to the processing of streams. Moreover, its CRF approach, although efficient on textual information, is not adapted to numerical values that we may encounter when collecting measures from sensors

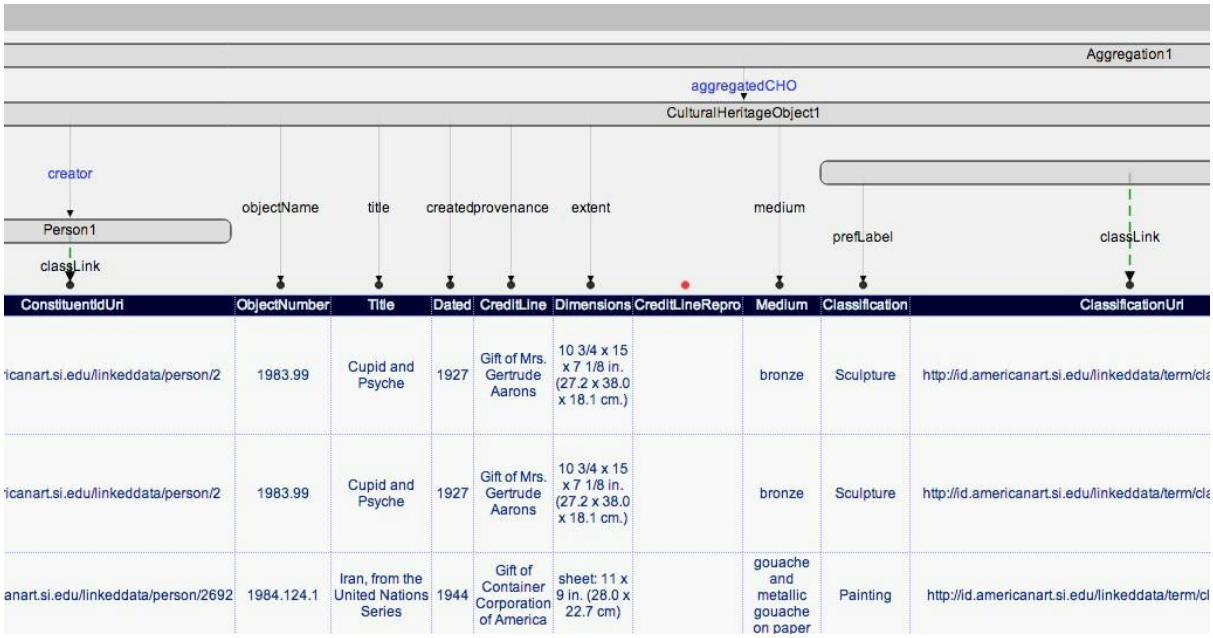

Figure 3: Karma alignment screenshot

6.2.3 Data lift

Datalift is defined as a catalyser for the Web of data. It has been designed and implemented in the context of a french ANR (Agence National de la Recherche) by a consortium of eight

partners (including academic, industry, institutional and innovation partners). Its principal goal is to support semantic lifting of raw data on the Web. One of its tasks consists in converting raw data into RDF conformant to a set of preloaded ontologies. This lift aims to take you from the ground floor, where the raw data reside, to fourth floor where interconnected data is accessible. The building of Datalift is constituted as follows. At the first floor, a set of a catalog of ontolgoies, possibly LOD ones, is stored. At the second floor, the transformation from raw data (thing CSV, XML, JSON, etc) to RDF is computed by using the vocabularies of the first floor. The third floor consists of the different repositories that are storing all the produced RDF data. At the last floor, generated datasets are analyzed to establish interlinking. The semantization task we are interested in this deliverable is performed at the second floor of the Datalift architecture. This aspect corresponds to the OntoMapper module in Datalift and maps RDF data to ontologies.

6.2.4 Summary

by a consortium of eight partners (including academic, industry, institutional and innovation partners). Its principal goal is to support semantic lifting of raw data on the Web. One of its tasks consists in converting raw data into RDF conformant to a set of preloaded ontologies. This lift aims to take you from the ground floor, where the raw data reside, to fourth floor where interconnected data is accessible. The building of Datalift is constituted as follows. At the first floor, a set of a catalog of ontolgoies, possibly LOD ones, is stored. At the second floor, the transformation from raw data (thing CSV, XML, JSON, etc) to RDF is computed by using the vocabularies of the first floor. The third floor consists of the different repositories that are storing all the produced RDF data. At the last floor, generated datasets are analyzed to establish interlinking. The semantization task we are interested in this deliverable is performed at the second floor of the Datalift architecture. This aspect corresponds to the OntoMapper module in Datalift and maps RDF data to ontologies.

7 RDF storage

7.1 Introduction

At each stage of the data streaming process, the system needs to access and possibly store RDF triplets (or quads). Due to the current architecture of the system, we can consider that several kinds of RDF stores are needed. At least two system categories are suspected to evolve in the WAVES ecosystem: one solution characterized by a low latency, in-memory, distributed storage and another one with a more classical disk-based persistence. The latter may be used in the storage of static, i.e. with a relatively low update rate, ontologies, such as Semantic Sensor Network (SSN), and knowledge bases, such as Geonames, descriptions of sensors and associated topologies. The former may be used during the reasoning process to store temporary inference results. There exists more than fifty RDF Stores(Figure 4, [9]) with different characteristics. Note that among these systems, several solutions are adapted to store either RDF triples (standard subject, predicate and object) or quads (subject, predicate,

object plus graph name) but, to the best of our knowledge,we do not know of any system able to store n-tuples with n > 4. In the following subsections we survey the main ones which will enable us to select the most adapted and efficient stores.

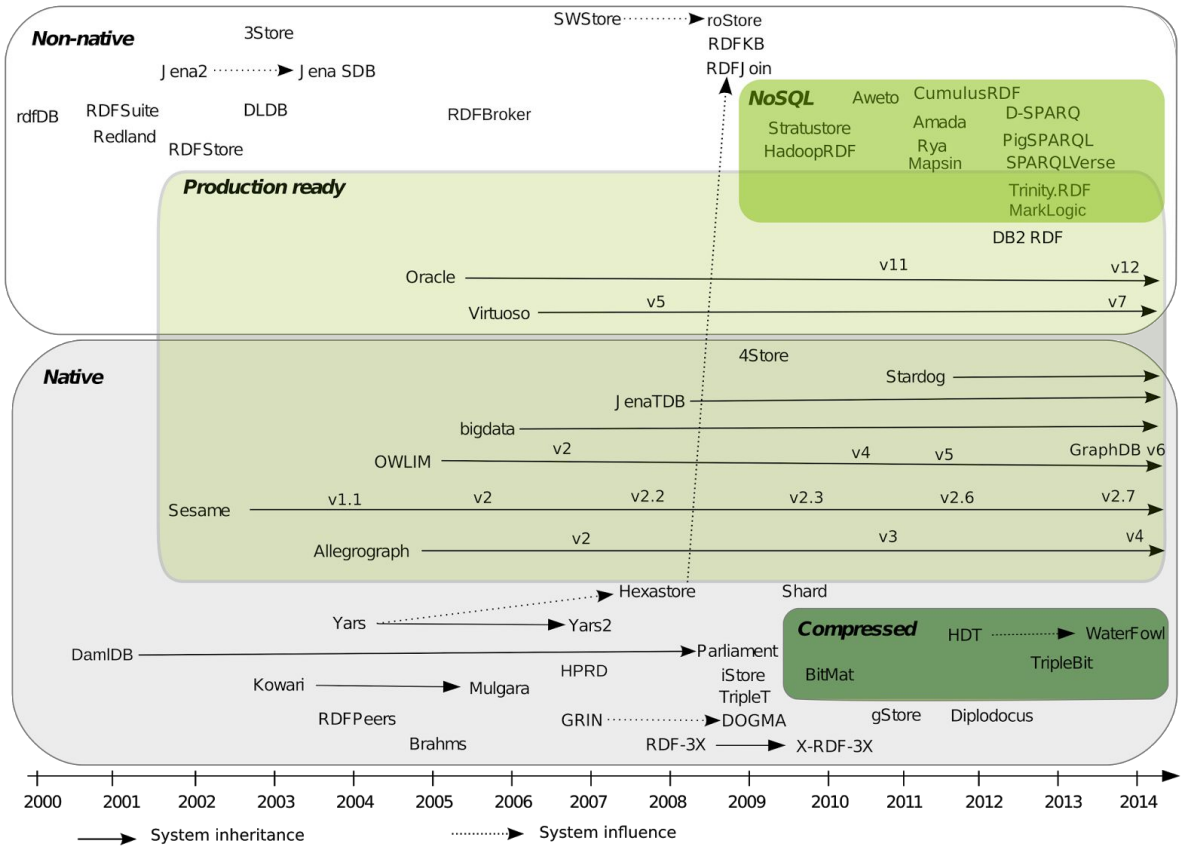

Figure 4: RDF stores ecosystem

7.2 Native vs non-native RDFstores

Among the plethora of system categories presented in Figure 5, we can distinguish between solutions that are implementing their own storage back-end, denoted as native, and those that are using an existing database management system, denoted as non-native. Systems following the native approach generally have to start their development from scratch and hence heavy design and implementation efforts are needed if one wants to release a ready for production system. It is not a surprise to encounter the most efficient, in terms of query and inference performances, solutions in this native category. Compared to the native storage solution, non-native are less demanding in terms of design and implementation efforts. Nevertheless, to obtain satisfying query performances the handling of the mismatch between the two data models, e.g., graph to relational, as well as query translation, for instance from SPARQL to SQL or some NoSQL query language like CQL, have to be taken into consideration.

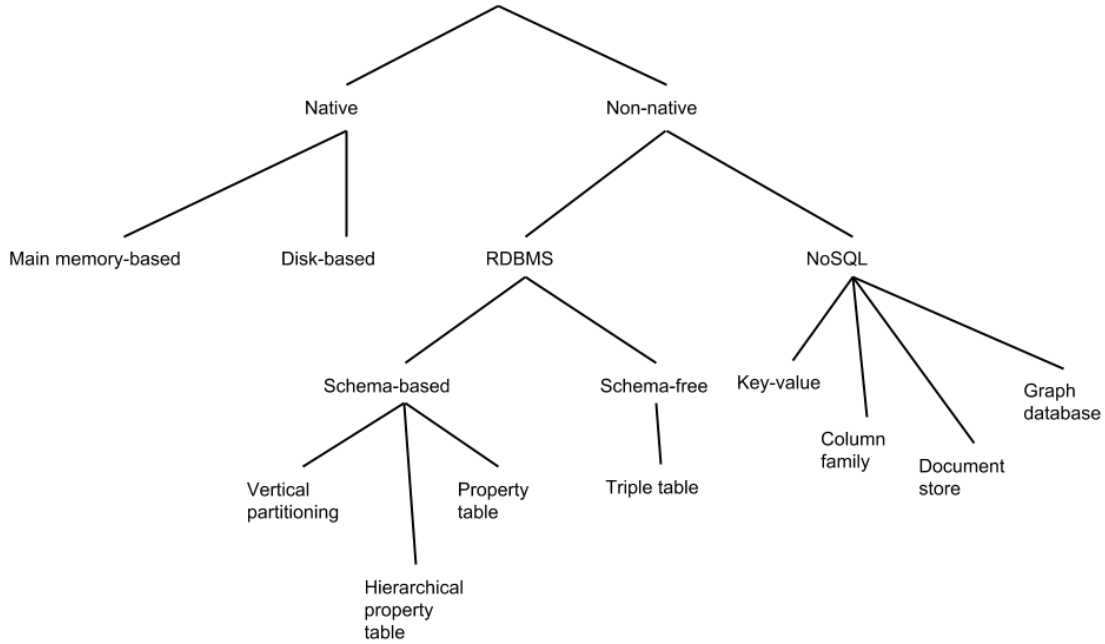

Figure 5: Taxonomy of RDF storage approaches

7.2.1 Native approach

The native approach provides a way to store RDF data closer to its data model, eschewing the mapping to entities of a DataBAse Management System (DBMS), such as relations. It uses the triples (or quads) nature of RDF data as an asset and enables to tackle the peculiarities of its graph approach, such as the ability to handle data sparsity and the dynamic aspect of its schema. These systems can be broadly classified as persistent, i.e., disk–based, and main memory–based systems. The persistent disk-based storage is the prevalent solution to store RDF data permanently on a file system. These implementations may use well known index structures, such as B-Trees which are extensively used in Relational DBMS (RDBMS). One may consider that reading from and writing to disks induces an important performance bottleneck. This consideration has motivated inmemory solutions which aim to store as much of the data, e.g., RDF triples, dictionaries or ontologies, in main memory. We then distinguish the standalone and the embedded representations. On one hand, standalone RDF native representations can be stored, transmitted and processed by their own means without referring to a specific host. On the other hand, embedded RDF native representations are part of a specific application/framework and only usable within a special context. Their representation is only defined on the context of this framework. Here, the triples are produced by applying a set of transformations [22]. The in-memory storage of

RDF data allocates a certain amount of the available main memory to store the whole RDF graph structure. Like the persistent disk based storage, this approach relies on research results in the database domain (e.g., indexes or efficient processing) and multiple indexing based techniques. When working on RDF data stored in main memory, some of the most time-consuming operations are the loading and parsing of the RDF file, but also the creation of suitable indexes. Therefore, an RDF store must have a memory efficient data representation that leaves enough space for the operation of search algorithms. We next present some systems belonging to the native approach. An important academic system that has influenced the RDF store community is RDF-3X [29]. It is characterized by a high number of indexes. RDF-3X stores its triples in the classical triple table approach, i.e., a single triple table with three columns. In RDF-3X, this huge table does not rely on an RDBMS but rather depends on its own storage system which has been especially designed for the purpose of RDF storage. One identified problem with the triple table storage approach is the proliferation of self joins in SPARQL queries. That is, since all triples are contained in a single table, joins are performed using 'copies' of this table. This can be very costly or even saturate the main memory of a powerful server since these tables can potentially contained millions of triples and SPARQL queries can possibly imply many joins.

The following two systems can be considered as production ready. Several storage solutions have been implemented in the context of the Apache Jena Semantic Web framework. The Apache Jena TDB system is stated as being faster, more scalable and better supported than the Jena SDB which is non-native system relying on an RDBMS. It is for instance the system supporting persistence in the Fuseki SPARQL server. The architecture is built around three concepts, namely node table, triple/quad indexes and prefixes table. The node table serves to store the dictionary and follows the standard to string-toid (aka locate) and id-to-string (aka extract) mapping approaches. Practically, the string-to-id and id-to-string operations are respectively implemented using B+ trees and a sequential file. A large cache is dedicated to ensure fast data retrieval during query processing. Triple and quad indexes are stored in specialized structures and respectively store three and four identifiers from the node table. B+ trees are used to persist these indexes. The system supports SPARQL Update operations which are handled using ACID transactions. Finally, TDB supports a bulk load solution which does not support transaction. The different features contained in Jena TDB, e.g., some security aspects as well as some APIs (Application Programming Interface), make it a solution to consider in a production setting. OWLIM (now GraphDB) corresponds to a family of RDF database systems that is implemented by the Ontotext company. Three implementations are available: OWLIM-Lite, OWLIM-SE (Standard Edition) and OWLIM-Enterprise. The OWLIM systems are taking advantage of the Sesame libraries which provide APIs for storing, querying as well as reasoning purposes. Using such libraries provides support for different RDF syntaxes, e.g., RDF/XML, N3, Turtle, as well as query languages (including SPARQL). Hence, it frees RDF store designers in developing their own parsers. The set of APIs provided by Sesame is named SAIL, standing for Storage And Inference Layer, and is used by several

other systems, e.g., GraphDB, BlazeGraph. The first system can be used free of charge while the last two are commercial products. OWLIM-Lite thrives on small data sets that can be stored in main memory (around 100 million statements for a server with 16Gb of RAM). Since all data related operations, i.e., querying and inferencing, are performed in memory, a basic (N-triples) file-based persistence is deemed sufficient. This aims to support data preservation and consistency between server starts up. The main memory approach enables high throughput: it is considered to be three times faster than OWLIM-SE. Nevertheless, OWLIM-Lite does not propose any query optimization nor advanced features, e.g., full-text search. OWLIM-SE provides a more advanced persistence layer which is based on binary data files and several indices. OWLIM-SE aims at data sets ranging around billion triples even on a desktop machine. Just like in OWLIM-Lite, queries can be expressed in SPARQL or SeRQL (due to the use of SAIL) but OWLIM-SE provides some forms of query optimizations. Advanced features such as RDF rank, full-text search (integrated into SPARQL with a standalone or Lucene approach) and geospatial extension are provided. The OWLIM Lite and SE share many features. For instance, being implemented as a Sesame SAIL providers, repositories can be created, loaded and queried through the user-friendly Sesame Workbench. Triple repositories can also be accessed with external editors such as TopBraid Composer. Similarly, applications can be developed using either the Sesame or Jena APIs. The cornerstone of the OWLIM systems is the support for reasoning. This is handle by the TRREE (Triple Reasoning and Rule Entailment Engine) which is being developed by Ontotext. Inferences are based on forward-chaining of entailment rules. OWLIM-Lite performs all reasoning in-memory while OWLIM-SE uses its set of data structures backed by the filesystem.

7.2.2 Non-native approach

The approach of the latter RDF Store category, i.e., non native, makes use of a database management system to store RDF data permanently. In terms of provided features, the most interesting non-native implementations are using a RDBMS to store their data. Such an approach benefits from years of research and development on these DBMS, this is especially relevant for systems based on an RDBMS. For instance, most RDBMS are known to have industrial strength transaction support and security considerations that most native approaches are lacking. In the RDBMS category, we can distinguish between schema-based and schemafree approaches. With schema-free, we mean that a single table, denoted tripletable, is responsible for the storage of all triples. In schema-based, unlike the previous representation that is quite straightforward, the schema characteristics are used to split the triple table into different tables based on the RDFS or OWL schema properties or classes. We can distinguish two major schemas, namely property table and the vertical partitioning approach. Recently some systems came to light with some NoSQL stores taking care of the storage back end. The main motivation behind these approaches, at least those based on a key-value, document or column-family store, is to address the distribution of very large data sets over a cluster of commodity hardware. Considering the graph database category, they

present the qualities of not providing an important mismatch with the RDF data model and to generally support ACID transactions. But supported partitioning strategies are considered to be less efficient that for the other NoSQL systems. The Virtuoso system [12] is produced by the OpenLink company. The database engine was first developed as an RDBMS and progressively evolved toward the XML and RDF data models. Given the maturity of the OpenLink RDBMS and efforts provided on the RDF, the system definitely has to be considered as a production ready solution. Originally following a row-store approach, the system was recently extended to become an hybrid approach by supporting column-store features. Later in this section, we will present the swStore system which was in 2007, a first RDF engine to emphasize the adequacy and efficiency of a column-store type of data storage for RDF triples. Virtuoso stores quads combining a graph to each triple (s,p,o). It, thus, conceptually stores the quads in a triples table expanded by one column. The columns are g for graph, p for predicate, s for subject and o for object. While technically rooted in an RDBMS, it closely follows the model of YARS [20] but with fewer indices. Several additional optimizations are added, including bitmap indexing. In this approach, the use of fewer indices tips the balance slightly towards insertion performance from query performance, but still favors a query one. Being a popular production ready system implies the existence of several features. The system also supports stored procedure and built-in function definition that can be used from SPARQL queries. Finally, SPARQL extensions such as its own full text search engine, geo spatial queries (using a special type of index which is denoted R-tree), business analytics and intelligence features and sub queries are supported. Virtuoso proposes several kinds of licenses. For a single machine deployment, a GPL open source solution is available.

As a NoSQL system, we can reference CumulusRDF [25] which is an RDF engine that uses the Apache Cassandra, a column family NoSQL store, as storage back-end. Hence, this RDF store benefits from the whole Cassandra machinery providing a decentralized, highly available, scalable storage solution with failure tolerance through replication and failover mechanisms using a no single point of failure approach. The storage of RDF triples supports the notion of named graph, i.e., it stores quads, and aims to cover all 6 possible RDF triple patterns with indexes and to exploit prefix lookups to reduce the number of materialized indexes. In [25], the authors describe two storage layouts which are denoted as hierarchical and flat. In the hierarchical layout, an important use of Cassandra's supercolumn supports the overall storage organization. Intuitively, a supercolumn is an additional layer of key-value pairs that occurs in column family. With that solution, the subject, predicate and object of a triple are respectively stored in the key, supercolumn and column and the value is left unspecified. Hence, there is a multiple supercolumns, i.e., predicate, for each row key, i.e., subject, and for each predicate, there is multiple possible columns, each of them storing a single object value. This provides an efficient SPO index. Two similar indexes are constructed for POS and OSP, i.e., motivating the distribution over row keys, supercolumns and columns. These 3 indexes are considered sufficient to support the 6 possible triple patterns. Note that, at the

time of writing this deliverable, in the latest versions of Cassandra, this option is not supported anymore and corresponds to the approaches used in HBase and BigTable, i.e., providing only the notions of keyspaces, column families and columns. Hence, this hierarchical storage layout is not adapted to recent Apache Cassandra releases. The flat layout makes an intensive use of Cassandra's secondary index solution. In that solution, a standard key/value model is adopted where the row key is a triple element and the column key and value correspond to the remaining triple pair. For instance, considering the SPO pattern, the subject is stored as the row key, the predicate is a column's key and the object pair is the column's value. Since column are stored in a given sorted order, this approach enables to perform both range scans and prefix lookups on column keys in an efficient manner. Just like in the hierarchical layout, 2 other indexes are needed, namely POS and OSP, to cover all possible triple patterns. The POS requires a special attention due to the distributed approach of Cassandra. The goal is to prevent data skew caused by the inequitable distribution of triples over predicates, e.g., rdf:type usually represent an important fraction of the entire dataset. Hence, in the case of POS, setting P as the unique row key is not an efficient approach, i.e., some very large rows will emerge and be stored on some given nodes, i.e., possibly exceeding node capacity for some data sets and preventing efficient load balancing. The proposed solution is based on Cassandra's secondary indexes. That is PO is used as the row key for the POS pattern and the remaining S is stored as the column key. In each row, a special column stores the predicate with a 'p' key. This, through the use of a secondary index on 'p', enables to retrieve all values for a given predicate. Note that this flat layout does not use supercolumns. Recently, CumulusRDF has been extended with a fourth index to support RDF named graph, i.e., storing quads instead of triples, with an CSPO index. Curiously, CumulusRDF does not make any use of dictionaries but prefers to store URIs, literals and blank nodes as column keys and values.

7.3 Distributed RDF stores

Most of the systems mentioned in the previous sections are characterized as centralized, i.e., storage and processing are handled by a single machine. While being adapted to certain situations, this type of system architecture suffers from many limitations. For instance, it is the case for big data workloads that generally address very large data volumes through distribution and replication. That is data and associated processing, e.g., query ones, are distributed over a set of machines and a lot of benefits are obtained by replicating those data fragments over some of these nodes. Hence, for applications facing big data constraints, a Distributed DataBase Management System (henceforth DDBMS) is required. DDBMS can be defined as a system where data management is distributed over several computers in a computer network. This field has long research and development histories in the relational data model with systems such as Ingres and System R being designed as early as the beginning 1980s. Mainly due to advances in network computing, e.g., emergence of the Internet, and computer clusters, these system's features and capacities have evolved since then. This has lead to the development of novel distribution approaches and new

DDBMS categories, e.g., Peer to Peer (P2P), federated databases. Being an integral part of the big data ecosystems, RDF is totally concerned with the distribution phenomena. Distributed RDF stores have benefited from the experience and results obtained in the relational DDBMS context. Hence, it is not surprising to more or less observe the same system categories even if the schema-less characteristic of RDF imposes some peculiarities. Figure 6 presents a taxonomy of distributed RDF stores.

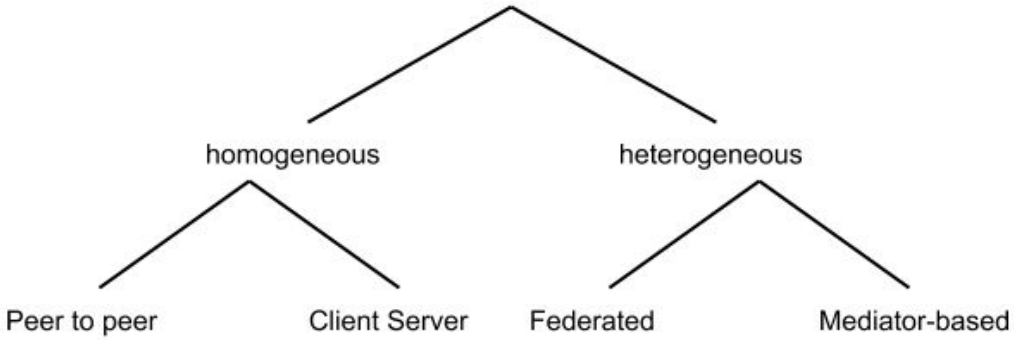

Figure 6: Taxonomy of distributed RDF stores

4Store has been developed by Garlik (now Experian) and is one of the most established distributed triple stores. It stores quads in so-called storage nodes which handle a segment of the quad data set. Each such node consists of two indexes. A first one corresponds to a hash on predicates where the value points to two distinct structures. A new version of 4store, named 5store, is being developed but has been only used within Experian and is not yet published. This distributed system is based on shared nothing master slave architecture. It is constituted of processing and storage nodes which persist data fragments in a non overlapping manner (called segments in the paper but which correspond to our notion of a fragment). Each machine of the cluster can store one or more fragments. For SPARQL triple patterns where the subject is not known, the fragmentation approach forces to send the query to all storage nodes. The allocation method is also based on a simple modulo operation. The system supports replication of fragments other several storage nodes. In terms of communication, 4Store permits exchange between Processing and Storage nodes via TCP/IP with a single connection established per segments between processing and storage nodes. This enables to process some queries in parallel since several requests can be sent to different segment. Query processing is coordinated by a single processing node. The joins of a SPARQL Basic GRaph Pattern (BGP) are executed on this node exploiting access paths reading data on storage nodes. The query processor is inspired by the standard relational algebra with specific optimizations related to the selectivity of the bind operations. Note that 4Store natively supports reasoning services but as we will see in the next section, they are not compatible with the inferences we would like to conduct in our streaming environment.

Yars2 [21] introduces distributed indexing method and parallel query evaluation methods to the Yars [20] system, a native system. The cluster is made up of commodity hardware and a shared nothing architecture. Based on the fact that it stores quads, the system proposes three different forms of indexes: (i) a keyword index which uses Apache Lucene as an inverted index to map terms occurring in RDF objects to subjects, (ii) 6 quad indexes (motivated by the use of prefix lookups), based on sparse indexes are used (index are in-memory and 6 sorted files are stored on disk and Huffman encoded), (iii) join indexes to speed up queries containing combinations of values or paths in the graph. The Partitioning method is hash based. The distributed query processor uses lookup requests performed in parallel.

In the previous section, we presented the open source single machine version of Virtuoso, but OpenLink also produces a commercial version which permits the distribution over a machine cluster. Two partitioning strategies are proposed. The first one consists of a data replication and hence enables high availability. The second strategy is based on partitioning that is specified at the index level using a hash function on key parts.

The clustering approach of the MarkLogic system distinguishes between two kinds of nodes: data managers (denoted as D-nodes) and evaluators (denoted as E-nodes). The D-nodes are responsible for the management of a data subset while the E-nodes handle the access to data and the query processing. The same physical machine can act as both kind of nodes, in fact this corresponds to a single-host environment. In the situation where an E-node fails, the load balancer just needs to send the query to another running E-node. In case of a D-node failure, the data is still needed to answer the query. This is can be resolved via a replication approach (the data fragment stored by the failing system is also stored on some other known D-nodes) or with a clustered filesystem. Finally, note that MarkLogic uses MapReduce to ingest, transform and export large volumes of data in a bulk processing manner.

SHARD (Scalable High-Performance, Robust and Distributed) [33] is a triple store designed on top of Hadoop. It relies on this framework for both the data persistence and query processing aspects. Considering data storage, RDF triples are stored in flat files in the HDFS file system. Each line of these flat files are organized as single key value where the key is the subject of any triple and the value is a set of predicate/object pairs associated to that subject. This approach is similar to the way HBase stores its data on disk. This flat file organization is adapted to the Map Reduce approach which expects key values as entries. Nevertheless, it is inefficient in terms of data redundancy (since some subjects frequently appear as the object of other triples) and it provides a single index on the subject. That second aspect is supposed to taken care of by parallel query processing which uses the Hadoop framework. Intuitively, each triple pattern of a SPARQL query is handled by an iteration of a MapReduce operation. The variable binding obtained at one iteration is assigned to the SPARQL triple patterns that have not yet been executed. A final MapReduce

set is performed to filter the distinguished variables (those present in the SELECT clause) and to remove duplicates.

7.4 Summary

Some of the systems we have described in this section address the issues we expect to face in the Waves. Concerning the persistent store, adopting the Virtuoso open source edition seems to be a good option for a first version of the project for the following reasons: the system is known to be robust and efficient, Atos already has some experience in configuring, tuning and implementing over this system.

8 Reasoning over RDF streams

8.1 Reasoning

Supporting reasoning services is an important feature in the context of RDF stream processing. This is rather obvious if one considers that inference services are one of the main added feature of RDF stores compared to traditional relational database management systems or NoSQL stores, graph-based ones included [9]. This ability to operate upon explicit as well as implicit data is a peculiarity of the knowledge base context that we do not find in the requirements of relational database based stream processing [34].

Reasoning in the Semantic Web has a long history that takes its roots in Artificial Intelligence and logic. Inferences drawn from ontologies support operations such as concept classification, qualification of the consistency of a knowledge base, retrieval of the instances of a given concept, finding the most specific concept an individual is an instance of and instance checking to name the main ones. They are generally implemented using automata theory, a translation to predicate logic, resolution-based methods or the semantic tableau approach. The latter two have been the most widely adopted among existing systems, e.g., Fact++, Pellet, HermiT, RacerPro for expressive ontologies, i.e., OWL2DL, and ELK, CEL, Quonto, Jena for lightweight ontologies, i.e., OWL2 profiles, namely EL, RL, QL, and RDFS.

These systems can not be used out of the box in a stream context due to the near real-time nature that forces to obtain reasoning results within a given duration. In this situation, a particular attention is given to the ontology expressiveness/computational complexity trade-off which should ensure that the relevant inferences can be performed in the defined time-window. This is the main reason for the adoption of RDFS reasoning, the least expressive ontology language of the W3C Semantic Web stack, in available systems. Other ontology languages of the W3C stack, namely OWL2DL and its profiles, are considered too computationally expensive in the stream context.

Nevertheless, some systems have considered an extension of RDFS, e.g., by introducing property transitivity, in RDF stream processing. Different types of RDF stream reasoning have been identified: data-driven, query-driven and hybrid solutions.

In the data-driven appraoch, the information contained in RDF streams is extended at load-time by materializing, a.k.a., saturating, possible inferences. The main limitations of this approach is the latency implied at data loading time due to the computation of all deductions and the difficulty to maintain a valid data set in the face of data updates, e.g., removing a single information can induce the deletion of an important number facts some of which can be derived by some other valid data. The main advantage of saturation is the performance of query processing due to the absence of any overhead. The Virtuoso RDF database management system is adopting this reasoning approach but does not support the notion of streams.

This approach can be opposed to query-driven which rewrites, a.k.a., reformulates, the (continuous) queries in order to retrieve all valid answers. With this solution, the advantages consist of fast loading times and easier handling of data updates. The main drawback correspond to slow query answering due to the overhead of reasoning over the data and the knowledge which is required for all queries. Note that these poor performances are due to the query evaluation process and not on the query reformulation which is being computed only once. Moreover, the rewriting can possibly generate a large number of queries, some of which can be semantically equivalent and syntactically different, resulting in a high rate of duplicate answers. GraphDB (formerly OWLIM) is an RDF store using this query-driven solution in non-streaming context.

The latter solution is a combination of the previous two approaches, i.e., a method where only a portion of the data would be materialized and a part of the query would be reformulated. Blazegraph (formerly bigdata) is an example of a hybrid system that deals with traditional, i.e., non streaming, database management.

In these inference services, the order in which the data is processed is not important. This is not the case in streaming context where the order of stream tuples may imply a certain interpretation and thus different deductions. We can distinguish between several order notions. Natural order is the most frequent and generally corresponds to the order in which stream tuples arrive in the system. In the case of RDF data, a timestamp value to the subject, predicate, object triple of RDF to form a quad. Some other forms of order, requiring a more or less involved computation, are also possible, e.g., popularity, frequency. In [38], the authors present a taxonomy of implemented systems and open research problems in RDF stream reasoning. They argue that most implemented systems belong to the data-driven category with a natural order. IMaRS [4] and EP-SPARQL [2] are such systems which respectively belong to the DSMS and CEP categories. It is recognized that the query-driven approach is promising due to the static nature of queries in a streaming context. That is, the reformulation of the query would be performed once and its optimized version could potentially be executed an infinite number of times. The hybrid, mixing data and query driven approaches, presents the most important research challenges but also the highest potential in terms of task adequacy and efficiency. A third dimension for RDF stream

reasoning corresponds to the parallel computation of the inference services. In a non-streaming context, systems such as WebPie [37] and SAOR [23] have been implemented using the MapReduce [10]. The batch processing nature of MapReduce is not adapted to handle streams and hence, these systems can not be used in a streaming context. To the best of our knowledge, parallel systems dedicated to stream processing, e.g., Storm, Spark Streaming, Samza or S4, have not been used by any RDF streaming processing system equipped with inference services.

8.2 Reasoning and query processing systems

Simplified table from the paper Streaming the Web : Reasoning over dynamic Data. [28]

8.2.1 C-SPARQL

Continuous SPARQL [3] is one of the main reference in stream processing. It supports time-stamped RDF triples, continuous queries over streams, and data aggregation. The streaming data are annotated with expiration time, that are handled in a different structure which deals with inference (based on validity). Then the reasoner can preserve the entailment of transitory knowledge.

8.2.2 IMaRS

Incremental Materialization for RDF Streams [4] proposes an algorithm for maintainting the materialization of ontologial entailments in the presence of streaming information. The approach depends on the expiration time of the triples. One of themain concepts consists in annotating each statement in the query window. Intuitively, it indicates when a statement has to be removed from the knowledge base. The maintenance algorithm is used inside the time window, and is executed each time the window slides: for each new triple, if there is some entailment, the algorithm stores the new knowledge. If the triple reaches its expiration time, it is removed, and the algorithm tries to rederive the entailment; if a derivation is found, the expiration time of the entailment is updated. This mechanism relies on the ability of pre-computing the expiration time of each entering stream triple Thus, it is not adapted to count-based windows approaches.

8.2.3 TrOWL

TrOWL [36] maintains a very expressive ontology, covering OWL2-DL, which makes it better than IMaRS on the complexity. It uses a DRed approach (Delete and Re-derive) for the entailments, just like IMaRS, making the maintenance of the materialization of the knowledge base an easy operation (no need to recompute everything). A comparison between both systems is not possible, as the tests on each one have not been conducted the same way. The DRed mechanism of TrOWL relies on a truth maintenance system that stores a justification for each entailment; with this approach, the program maintains a directed graph to be used for additions and deletions of entailments. However, computing all the justifications is costly, and the truth maintenance graph is expensive too.

8.2.4 EP-SPARQL

Event processing regroups techniques and programs that allow to control event-driven and real-time systems, and thus can be useful to process data streams. Event Processing-SPARQL [2] is an extension of SPARQL that uses the processing of CEP systems; the query language now contains some binary operators to combine graph patterns (seq, equals, optionalseq, and equals optional). With the accompanying ETALIS (Event TrAnsaction Logic Inference System) component, complex events are derived from simpler events by means of deductive rules. ETALIS is based on a declarative semantics, grounded in Logic Programming and is implemented in Prolog. Due to its root in logic, ETALIS also supports inferences over events, context, and real-time complex situations (i.e., Knowledge-based Event Processing).

8.2.5 Sparkwave

Sparkwave processes RDF data streams with additional RDF Schema (extended with inverse and symmetric propertic) entailments. The key contributions are the usage of the RETE network approach which corresponds to a pattern matching algorithm that has been developed in the context of production rule systems. Intuitively, the reasoning method determines which of the system's rules should be fired. The standard set of RDFS rules is extended to support the owl:inverseOf and owl:symmetricProperty constructors (onmy 3 additional rules are needed to support both of them). Sparkwave operates on fixed schema and requires a pre-processing. The system is also limited by the fact that the knowledge base is retained in main-memory.Finally, the system does not seem to be maintained.

8.3 Summary

Given the presentation of the main RDF stream processing systems, we can consider that defining an RDF stream processing system which support to reason over expressive ontologies in a scalable, parallelized manner is an open problem.

In order to identify expected reasoning features of the WAVES system, we need to clarify the the kind of inferences one expects from the potable water user case of the project.

9 Glossaire

CEP	Complex Event Processing
CSV	Comma Separated Values
DSMS	Data Streams Management Systems
JSON	JavaScript Object Notation
OWL	Web Ontology Language
RDF	Resource Description Framework

SPARQL	SPARQL Protocol and RDF Query Language
XML	eXtended Markup Language

10 Bibliographie

[1] D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In VLDB, pages 411–422, 2007.

[2] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011, pages 635–644, 2011.

[3] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. CSPARQL: a continuous query language for RDF data streams. Int. J. Semantic Computing, 4(1):3–25, 2010.

[4] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Incremental reasoning on streams and rich background knowledge. In The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I, pages 1–15, 2010.

[5] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. Schema Matching and Mapping. Springer, Heidelberg, 2011.

[6] A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL - extending SPARQL to process data streams. In The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings, pages 448–462, 2008.

[7] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an efficient rdf store over a relational database. In SIGMOD Conference, pages 121–132, 2013.

[8] A. Chatzistergiou and S. D. Viglas. Fast heuristics for near-optimal task allocation in data stream processing over clusters. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014, pages 1579–1588, 2014.

[9] O. Curé and B. Guillaume, editors. RDF Database Systems: Triples Storage and SPARQL Query Processing. Morgan Kaufmann, Boston, MA, USA, 1st edition, 2015.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, pages 137–150, 2004.

[11] D. Dell'Aglio, J. Calbimonte, M. Balduini, O. Corcho, and E. D. Valle. ´ On correctness in RDF stream processor benchmarking. In The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II, pages 326–342, 2013.

[12] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In S. Auer, C. Bizer, C. Müller, and A. V. Zhdanova, editors, Conference on Social Semantic Web, volume 113 of LNI, pages 59–68. GI, 2007.

[13] J. Euzenat and P. Shvaiko. Ontology Matching, Second Edition. Springer, 2013.

[14] J. D. Fernandez, A. Llaves, and O. Corcho. Efficient RDF interchange (ERI) format for RDF data streams. In The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II, pages 244–259, 2014.

[15] S. Gao, T. Scharrenbach, and A. Bernstein. The CLOCK data-aware eviction approach: Towards processing linked data streams with limited resources. In The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings, pages 6–20, 2014.

[16] N. F. Garcia, J. Arias-Fisteus, L. S´anchez, D. Fuentes-Lorenzo, and O. Corcho. RDSZ: an approach for lossless RDF stream compression. In The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings, pages 52–67, 2014.

[17] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub. Answer set programming for stream reasoning. CoRR, abs/1301.1392, 2013.

[18] D. Gerber, S. Hellmann, L. Bühmann, T. Soru, R. Usbeck, and A. N. Ngomo. Real-time RDF extraction from unstructured data streams. In The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I, pages 135–150, 2013.

[19] S. Harris, , N. Lamb, and N. Shadbol. 4store: The design and implementation of a clustered rdf store. In In SSWS2009: Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems, 2009.

[20] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In LA-WEB '05: Proceedings of the Third Latin American Web Congress, pages 71–80, Washington, DC, USA, 2005. IEEE Computer Society.

[21] A. Harth, J. Umbrich, A. Hogan, and S. Decker. Yars2: A federated repository for querying graph structured data from the web. In Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South

Korea, volume 4825 of LNCS, pages 211–224, Berlin, Heidelberg, November 2007. Springer Verlag.

[22] M. Hausenblas and B. Adida. Rdfa in html overview, 2007.

[23] A. Hogan, J. Z. Pan, A. Polleres, and S. Decker. Saor: Template rule optimisations for distributed reasoning over 1 billion linked data triples. In International Semantic Web Conference (1), pages 337–353, 2010.

[24] C. A. Knoblock, P. A. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick. Semi-automatically mapping structured sources into the semantic web. In The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings, pages 375–390, 2012.

[25] G. Ladwig and A. Harth. Cumulusrdf: Linked data management on nested key-value stores. In Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10th International Semantic Web Conference (ISWC2011), October 2011.

[26] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the Eighteenth International Conference on Machine Learning, ICML ’01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[27] F. Maali, R. Cyganiak, and V. Peristeras. Re-using cool uris: Entity reconciliation against lod hubs. In In LDOW, 2011.

[28] A. Margara, J. Urbani, F. van Harmelen, and H. E. Bal. Streaming the web: Reasoning over dynamic data. J. Web Sem., 25:24–44, 2014.

[29] T. Neumann and G. Weikum. Rdf-3x: a risc-style engine for rdf. Proc. VLDB Endow., 1(1):647–659, 2008.

[30] M. Nickles and A. Mileo. Web stream reasoning using probabilistic answer set programming. In Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings, pages 197–205, 2014.

[31] D. L. Phuoc, H. N. M. Quoc, C. L. Van, and M. Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I, pages 280–297, 2013.

[32] Y. Ren and J. Z. Pan. Optimising ontology stream reasoning with truth maintenance system. In Proceedings of the 20th ACM Conference on Information and Knowledge

Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011, pages 831–836, 2011.

[33] K. Rohloff and R. E. Schantz. High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triplestore. In PSI EtA, page 4, 2010.

[34] M. Stonebraker, U. C¸ etintemel, and S. B. Zdonik. The 8 requirements of real-time stream processing. SIGMOD Record, 34(4):42–47, 2005.

[35] K. Teymourian and A. Paschke. Plan-based semantic enrichment of event streams. In The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings, pages 21–35, 2014.

[36] E. Thomas, J. Z. Pan, and Y. Ren. Trowl: Tractable OWL 2 reasoning infrastructure. In The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II, pages 431–435, 2010.

[37] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal. Owl reasoning with webpie: Calculating the closure of 100 billion triples. In ESWC (1), pages 213–227, 2010.

[38] E. D. Valle, S. Schlobach, M. Kr¨otzsch, A. Bozzon, S. Ceri, and I. Horrocks. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web, 4(2):219–231, 2013.

[39] C. Weiss and A. Bernstein. On-disk storage techniques for semantic web data - are b-trees always the optimal solution? In Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009), pages 49–64, Washington DC, USA, October 26 2009.

[40] Z. Yang, J. Tang, and Y. Zhang. Active learning for streaming networked data. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014, pages 1129–1138, 2014.

[41] S. Komazec, D. Cerri, D. Fensel. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. Proceedings of the Sixth {ACM} International Conference on Distributed Event-Based Systems, (DEBS) 2012, Berlin, Germany, July 16-20, 2012, pages 58-68