The BigDataEurope Platform – Supporting the Variety Dimension of Big Data

Sören Auer, Simon Scerri, Aad Versteden, Erika Pauwels, Angelos Charalambidis, Stasinos Konstantopoulos, Jens Lehmann, Hajira Jabeen, Ivan Ermilov, Gezim Sejdiu, Andreas Ikonomopoulos, Spyros Andronopoulos, Mandy Vlachogiannis, Charalambos Pappas, Athanasios Davettas, Iraklis A. Klampanos, Efstathios Grigoropoulos, Vangelis Karkaletsis, Victor de Boer, Ronald Siebes, Mohamed Nadjib Mami, Sergio Albani, Michele Lazzarini, Paulo Nunes, Emanuele Angiuli, Nikiforos Pittaras, George Giannakopoulos, Giorgos Argyriou, George Stamoulis, George Papadakis, Manolis Koubarakis, Pythagoras Karampiperis, Axel-Cyrille Ngonga Ngomo, Maria-Esther Vidal

The H2020 BigDataEurope Project Consortium
c/o Fraunhofer IAIS, Sankt Augustin, Germany
Corresponding authors: {auer,scerri,jabeen,mami}@cs.uni-bonn.de

Abstract. The management and analysis of large-scale datasets – described with the term Big Data – involves the three classic dimensions volume, velocity and variety. While the former two are well supported by a plethora of software components, the variety dimension is still rather neglected. We present the BDE platform – an easy-to-deploy, easy-to-use and adaptable (cluster-based and standalone) platform for the execution of big data components and tools like Hadoop, Spark, Flink, Flume and Cassandra. The BDE platform was designed based upon the requirements gathered from seven of the societal challenges put forward by the European Commission in the Horizon 2020 programme and targeted by the BigDataEurope pilots. As a result, the BDE platform allows to perform a variety of Big Data flow tasks like message passing, storage, analysis or publishing. To facilitate the processing of heterogeneous data, a particular innovation of the platform is the Semantic Layer, which allows to directly process RDF data and to map and transform arbitrary data into RDF. The advantages of the BDE platform are demonstrated through seven pilots, each focusing on a major societal challenge.

1 Introduction

The management and analysis of large-scale datasets – described with the term Big Data – involves the three classic dimensions *volume*, *velocity* and *variety*. While the former two are well supported by a plethora of software components, the variety dimension is still rather neglected. We present the *BigDataEurope*[1]

[1] BigDataEurope (https://www.big-data-europe.eu/) is a Coordination and Support Action funded by the Horizon 2020 programme of the European Commission.

(BDE) platform – an easy-to-deploy (cluster-based and standalone), easy-to-use and adaptable platform where the variety dimension of Big Data is taken into account right from its inception. The BDE platform is currently being applied to the seven societal challenges put forward by the European Commission in its Horizon 2020 research programme (Health, Food and Agriculture, Energy, Transport, Climate, Social Sciences and Security)[2].

While working with stakeholder communities of the seven societal challenges over the last two years, we have identified the following requirements to be crucial for the success of Big Data technologies:

R1 Simplifying use. The use of Big Data components and the development of analytical algorithms and applications is still cumbersome. Many analytical applications are 'hard-wired', requiring lavish data 'massaging' and complex development in various languages and data models.

R2 Easing deployment. Various deployment schemes are required for different Big Data applications or even during the lifecycle of one particular Big Data application. Prototyping and development, for example, should be possible on a single machine or small cluster, testing and staging possibly on a public or private cloud infrastructure, while production systems might have to be deployed on a dedicated cluster.

R3 Managing heterogeneity. Our survey of societal challenge stakeholders [3] clearly showed that in most cases, Big Data applications originate with a large number of heterogeneous, distributed and often relatively small datasets. Only after their aggregation and integration true Big Data emerges. Hence, a major challenge is managing the heterogeneity of data in terms including data models, schemas, formats, governance schemes and modalities.

R4 Improving scalability. Finally, especially if a number of different storage and processing tools are employed, scalability is still an issue.

The BDE platform addresses these requirements and facilitates the execution of Big Data frameworks and tools like Hadoop, Spark, Flink and many others. We have selected and integrated these components based upon the requirements gathered from the seven different societal challenges. Thus, the platform allows to perform a variety of Big data flow tasks such as *message passing* (Kafka, Flume), *storage* (Hive, Cassandra), *analysis* (Spark, Flink) or *publishing* (GeoTriples).

The remainder of the article is structured as follows: section 2 presents an overview of the BDE platform, while section 3 introduces its Semantic Layer. We present two out of the overall seven pilots demonstrating the advantages of the BDE platform in detail in section 4. We discuss related work and conclude in section 5 along with directions for future work.

2 Platform Overview

The requirements gathered from the seven societal challenge stakeholders revealed that the BDE platform should be generic and must be able to support a

[2] https://ec.europa.eu/programmes/horizon2020/en/h2020-section/societal-challenges

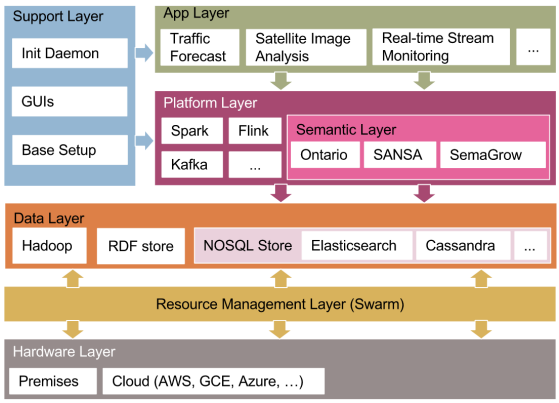

Fig. 1. High-level BDE platform architecture

variety of Big Data tools and frameworks running together. Installing and managing such a system on the native environments without running into resource or software conflicts is rather hard to achieve. Figure 1 gives a high-level overview on the BDE platform architecture, which is described in the sequel.

Hardware layer We decided to use *Docker*[3] as packaging and deployment methodology so as to manage the variety of underlying hardware resources efficiently alongside the varying software requirements for different stakeholders. Docker allows to package an application into a standardized unit for development. Docker containers wrap a software in a complete file system that contains everything needed to run, making sure that the software always runs as intended, regardless of the server environment. Therewith, dockers offers a lightweight virtual environment by sharing the same operating system kernel. The images of containers are constructed from a layered file system, sharing common files, thus making the disk usage and image downloads efficient. It is based on open standards and able to run on major operating systems. We have dockerized a large number of Big Data components[4]. None of them posed major problems to be run in a Docker container. Based on our experience and the popularity of Docker, we are confident that all components can be dockerized in reasonable time.

Resource management *Docker Swarm*[5] is an orchestration tool that allows to deploy Docker containers on a cluster in a transparent way. With its built-in scheduler, it offers most of the features required by the platform, i.e., scalability, interlinking the containers, networking between different containers, resource management between containers, load balancing, fault tolerance, failure recovery, log-based monitoring etc. Docker Swarm operates as a resource manager directly on top of the hardware layer. This hardware layer can vary from small set of

[3] https://www.docker.com
[4] https://github.com/big-data-europe/README/wiki/Components
[5] https://www.docker.com/products/docker-swarm

machines in the premises of an organization to the infrastructure of some big cloud provider. On top of Docker Swarm, applications can be deployed easily as a single Docker container, or through *Docker Compose*[6] as a collection (cluster) of communicating containers that can be scaled and scheduled dynamically.

Support layer. The technical aim of the BDE Platform is to reuse components wherever possible and build tools which are necessary to fit the societal challenge needs. In this regard, we discovered that when starting a Docker Compose application, the defined services will be started all at once. This is not always the intended behaviour, since some applications may depend on each other, or on a human intervention. For example, a Flink worker requires the Flink master to be available before it can register with the master. Another example is a Flink MapReduce algorithm that requires a file to be available on HDFS before starting the computation. At the moment, these dependencies cannot be expressed in Docker Compose. Awaiting the general solution of the Docker community, we have developed a semantic alternative. We provide an *init daemon* service that, given an application-specific workflow, orchestrates the initialization process of the components. The init daemon service is a microservice built on the `mu.semte.ch` platform and provides requests through which the components can report their initialization progress. The init daemon is aware of the startup workflow and, thus, it can validate whether a specific component can start based on the initialization status reported by the other components. The workflow needs to be described per application. It specifies the dependencies between services and indicates where human interaction is required. The UIs described in the next section together with the base Docker images and the init daemon service provide additional support to the user. It facilitates the tasks of building, deploying and monitoring Big Data pipelines on the BDE platform. This support is illustrated in Figure 1 as an additional layer in the platform architecture.

User interfaces. In order to lower the usage barrier of Big Data technologies, we have implemented several UIs, including a pipeline builder, a pipeline monitor, an integrator UI and a Swarm UI (see figure 2). The UIs serve different purposes, but in general make it easier for the user to build, deploy and monitor applications on the BDE platform. Most of the UI applications are built on `mu.semte.ch`, a microservices framework backed by Linked Data. Each of the applications is described in more detail in the next sections.

Workflow Builder. This interface allows users to create and edit workflows. The workflow steps reflect the dependencies between the container images or manual actions in the application. Example dependencies are:

- The Spark master needs to be started before the Spark worker such that the worker can register itself at the master.
- The input data needs to be loaded in HDFS before the MapReduce algorithm starts computing.

The order of the steps can be rearranged by dragging-and-dropping the step panels. Once finished the workflow can be exported and fed into the init daemon.

[6] `https://docs.docker.com/compose`

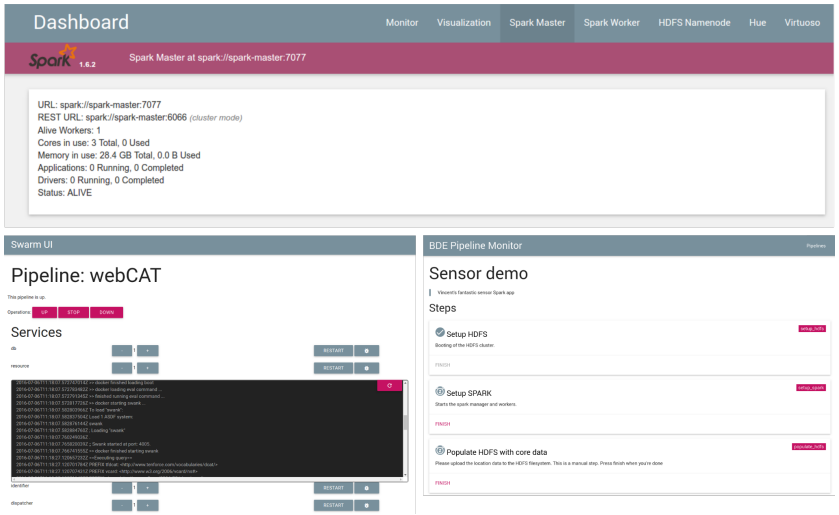

Fig. 2. BDE Platform UI Screenshots. Top: Integrator UI, focusing on Spark Master Dashboard. Bottom left: Swarm UI showing the services and the status of a pipeline named *webCAT*. Bottom right: Workflow Monitor showing a case of a pipeline named *Sensor demo* consisting of a set of stages – installing HDFS, installing Spark, populating HDFS with data, etc.

Workflow monitor. Once an application is running on the BDE platform, this interface allows a user to follow-up the initialization process. It displays the workflow as defined in the pipeline builder application. For each step in the workflow, the corresponding status (not started, running or finished) is shown as retrieved from the init daemon service. The interface automatically updates when a status changes, due to an update through the init daemon service by one of the pipeline components. The interface also offers the option to the user to manually abort a step in the pipeline if necessary.
Swarm UI. The Swarm UI allows to clone a Git repository containing a pipeline (i.e., containing a `docker-compose.yml`) and to deploy this pipeline on a Swarm cluster. Once the pipeline is running, the user can inspect the status and logs of the several services in the pipeline. Users can also scale up/down one or more services or start/stop/restart them.
Integrator UI. Most components (e.g. Spark, Flink, HDFS) provide dashboards to monitor their status. Each of the components runs in a separate Docker container with its own IP address, ports and varying access paths. The Integrator UI displays all component dashboards in a unified interface.

3 Semantic Layer

The ability to cross-link large-scale data with each other and with structured Semantic Web data, and the ability to uniformly process Semantic Web and

other data adds value both to the Semantic Web and the Big Data communities; it extends the scope of the former to include a vast data domain and increases the opportunities for the latter to process data in novel ways and combinations.

3.1 Semantic Data Lake

The term Data Lake, in the context of Big Data, appeared in recent years[7] to describe the repository of datasets that are provided for processing and analysis in their very original formats. It is often regarded as the opposing concept of Data Warehouse in the sense that, in the later, data is ready for analysis only *a posteriori* of a mandatory data reorganization phase. This difference is captured by two opposing data access strategies: data *on-read* and data *on-write*, respectively. Data on-read corresponds to the Data Lake where the schema of the data is looked at only when the data is actually used. Data on-write, on the other hand, corresponds to the Data Warehouse where data is organized according to a rigid schema before processing. Another substantial difference between Data Lakes and Data Warehouses is the type of processing the data is undergoing in each case. A Data Lake, as per its definition, contains data that is open to any kind of processing, be it natural language processing, machine learning, ad-hoc (semi-)structured querying, etc. In a Data Warehouse, on the other side, data is accessible only using a specific suite of tools, namely ad-hoc OLAP queries and BI standards. In the BDE platform, we put emphasis on easing and broadening and, whenever possible, harmonizing the use of Big Data technologies. The adoption of the Data Lake concept is, therefore, less of a choice, it is rather a logical consequence. This brings us back to our main goal: addressing the problem of variety and heterogeneity of the data.

While a Data Lake allows heterogeneous data to be stored and accessed, dealing with this data is cumbersome, time-consuming and inefficient, due to the different data models, instance structures and file formats. To address this, the idea of semantifying Data Lakes was recently proposed. The idea is to equip datasets in the lake with mappings to vocabularies, ontologies or a knowledge graph, which can then be used as a semantic access layer to the underlying data.

Ontario. The BDE platform comprises a Semantic Data Lake implementation, named *Ontario*. Ontario builds a Semantic Layer on top of the Data Lake, which is responsible for mapping data into existing Semantic vocabularies/ontologies. A successful mapping process, termed Semantic Lifting, provides a 'uniform' view over the whole data. As a result, the user can deal with the heterogeneous data in the lake as if it was in one unique format. Data can then be extracted, queried or analysed using a unique high-level declarative query language. The primary challenge is then to trigger a process with three main steps:

1. *Query analysis and decomposition:* The query is broken down into subqueries. An execution plan is generated.

[7] https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes

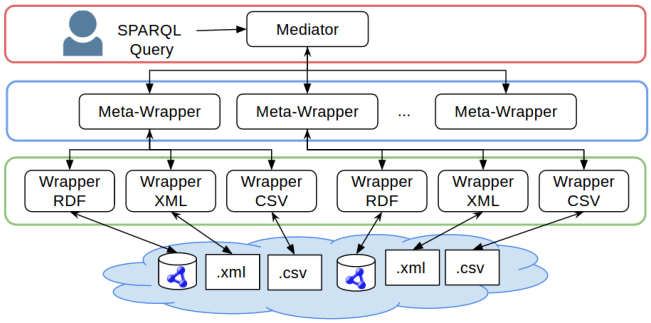

Fig. 3. Ontario multi-layer architecture for ontology-based data access to the Data Lake.

2. *Selection of relevant data sources:* Relevant datasets are selected starting from the generated sub-queries and using the mappings we have predefined.
3. *Extraction of the results:* The sub-queries are translated into the syntax of the selected datasets, executed and their results are put together in a certain way (following the plan generated in 1: merge, join, etc.) that the original query is accurately answered.

The main characteristics of Ontario are that the query execution process does not require any data materialization or shipping, and is fully transparent to the user. Data extraction rather happens fully on-the-fly upon a reception of a query.
Data Model. Data in Ontario is conceptually modelled around RDF classes. Each class can be seen as a star-shaped graph centred on the RDF subjects of the same type. The predicates of a class C consist of all the predicates found connected to the class C, even if they were dispersed across different data sources. Thus, the data in the Semantic Data Lake is conceptually represented as a set of class instances. Thanks to the Semantic Mappings associated to each dataset in the lake, every data instance has a class, even if it is not RDF.
Architecture. Ontario adopts a Wrapper-Mediator architecture (cf. Figure 3) with one extra middle-layer:

- *Mediator:* Decomposes the SPARQL query into a set of star-shaped groups. Star-shaped groups are planned into a bushy tree execution plan.
- *Meta-Wrapper:* Each star-shaped group is submitted to a Meta-Wrapper.
- *Wrapper:* Translates the star-shaped group into a query of the syntax of the final data source.

Wrappers are selected by the Meta-Wrappers to obtain a sub-set of the final data. Those result sub-sets returned by the wrappers are joined together according to the bushy tree execution plan forming the final query answer.
Example. We are interested in getting the *number of distinct publications and number of distinct deaths due to the disease Tuberculosis in India.* To obtain the answer, three datasets of different formats need to be queried: PubMed in XML, GHO in CSV and LinkedCT in RDF.

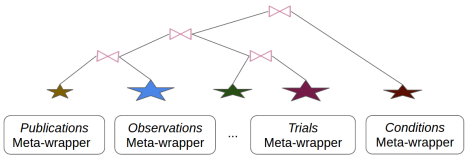

Fig. 4. Bushy execution plan and Meta-Wrapper invocation.

Fig. 5. An example of a SPARQL query on the left, and the corresponding star-shaped groups the Mediator generates on the right.

Listing 1.1. Example of RML Mapping Rules

```
<#ObservationMappings>
  rml:logicalSource [
    rml:source                "hdfs://.../GHO/observations.csv" ;
    rml:referenceFormulation  ql:CSV  ];
  rr:subjectMap   [ rr:template "http://ex.com/{@OID}" ];
  rr:predicateObjectMap [
    rr:predicate  gho:Country;
    rr:objectMap  [ rml:reference "country"; ];
  rr:predicateObjectMap [
    rr:predicate  gho:Disease;
    rr:objectMap  [ rml:reference "disease"; ];
...
```

The respective query used is shown in Figure 5 left. The Mediator decomposes the SPARQL query into star-shaped groups and generates a corresponding bushy tree execution plan, as shown in Figure 4. Each star-shaped group is submitted to a Meta-Wrapper. Each Meta-Wrapper checks the mapping rules, which are expressed in RML[8] in our implementation, and selects the relevant Wrappers, as depicted in Figure 6. A snippet from the mapping rules used is shown in listing 1.1. We can read that observations exist in the CSV file located in HDFS. The CSV columns are mapped to Ontology terms, e.g. country to `gho:Country`, disease to `gho:disease`, etc. Using those Mappings, the Wrapper is able to convert the star-shaped group, sent from the Meta-Wrapper, into an executable

[8] http://rml.io

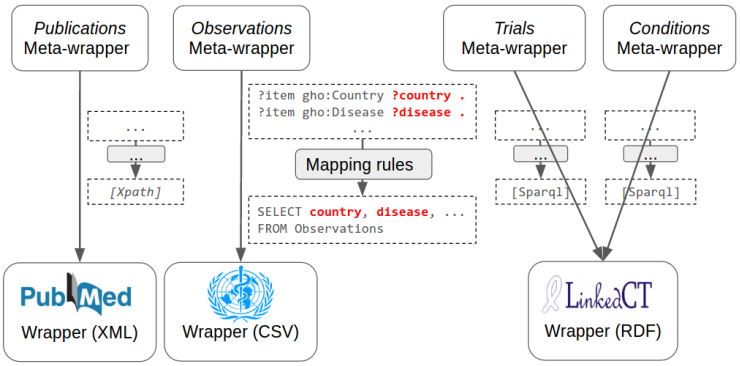

Fig. 6. Examples of Data Lake wrappers for PubMed, LinkedCT and GHO datasets.

query. For example, in Figure 6, the Wrapper converts the star-shaped group into an SQL query, as the CSV file is queried using *Apache Spark SQL*[9]. Wrappers' individual results are joined according to the execution plan generated earlier.

3.2 Big Data Analytics for RDF

One of the key features of Big Data is its complexity and heterogeneity. While most of the big data applications can deal with concurrent computations of different kinds of data, there is still the need to combine these data from different simultaneous resources in a meaningful manner. RDF provides a model for encoding semantic relationships between items of data so that these relationships can be interpreted computationally. This section presents the *Semantic ANalytics StAck* (SANSA)[10] which is a Big Data platform that provides tools for implementing machine learning algorithms directly on RDF data. SANSA is divided into different layers described below.

Read/write Layer. This layer provides the facility to read and write RDF data from HDFS or local drive and represent it in the distributed data structures of the frameworks.

Listing 1.2. Example of reading an RDF file in SANSA

```
import
net.sansa_stack.rdf.spark.io.NtripleReader

val input = "hdfs://.../file.nt"

val triplesRDD =
NTripleReader.load(spark, new File(input))
triplesRDD.take(5).foreach(println(_))
```

Listing 1.3. Example to load an OWL file

```
// In RDD
FunctionalSyntaxOWLAxiomsRDDBuilder.build(sc,
      "path/to/functional/syntax/file.owl")
// Using Dataset
FunctionalSyntaxOWLAxiomsDatasetBuilder.build(
      spark, "path/to/functional/syntax/file.owl")
// Manchester syntax RDD
ManchesterSyntaxOWLAxiomsRDDBuilder.build(sc,
      "path/to/manchester/syntax/file.owl")
// Manchester syntax Dataset
ManchesterSyntaxOWLAxiomsDatasetBuilder.build(sc,
      "path/to/manchester/syntax/file.owl")
```

[9] http://spark.apache.org/sql
[10] http://sansa-stack.net/

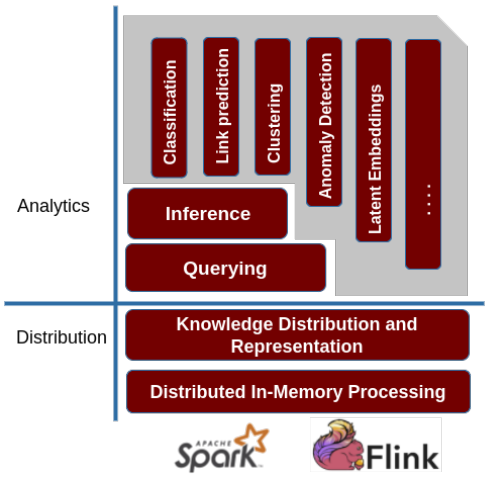

Fig. 7. Conceptual view of the BDE Scalable Semantic Analytics Stack (SANSA).

Querying Layer Querying an RDF graph is a major source of information extraction and searching facts from the underlying linked data. In order to efficiently answer runtime SPARQL queries for large RDF data, we are exploring different representation formats, namely graphs, tables and tensors.

Listing 1.4. Example to query an RDF file using SPARQL

```
val graphRdd = NTripleReader.load(spark, new File(input))
val partitions = RdfPartitionUtilsSpark.partitionGraph(graphRdd)
val rewriter = SparqlifyUtils3.createSparqlSqlRewriter(spark, partitions)
val qef = new QueryExecutionFactorySparqlifySpark(spark, rewriter)
```

Our aim is to have cross representational transformations for efficient query answering. Spark's GraphX is not very efficient, due to complex querying related to graph structure. On the other hand, an RDD based representation is efficient for queries like filters or applying user defined functions on specific resources, data frames have been found efficient for calculating the support of rules.
Inference Layer The core of the inference process is to continuously apply schema related rules on the input data to infer new facts. This process can derive new facts from the knowledge base, detect inconsistencies from the KBs, and extract new rules to help in reasoning. Rules describing general regularity can help to understand the data better.

Listing 1.5. Example of inferencing an RDF Graph in SANSA

```
val graph = RDFGraphLoader.loadFromFile(new File(input).getAbsolutePath, spark, 4)
val reasoner = profile match {                    // create reasoner
   case TRANSITIVE => new TransitiveReasoner(spark, properties, parallelism)
   case RDFS => new ForwardRuleReasonerRDFS(spark, parallelism)
   case RDFS_SIMPLE =>
     val r = new ForwardRuleReasonerRDFS(spark, parallelism)
        r.level = RDFSLevel.SIMPLE
```

```
        r
    case OWL_HORST => new ForwardRuleReasonerOWLHorst(spark)
}
val inferredGraph = reasoner.apply(graph)        // compute inferred graph
```

We use an adaptive rule engine that will be able to optimize itself based on the rules available in the KB. This helps in developing an execution plan from a set of inference rules enabling applications to fine tune the rules for scalability.
Machine Learning Layer In addition to above mentioned tasks, one of the very important tasks is to perform machine learning or analytics to gain insights of the data for relevant trends, predictions or detection of anomalies. There exists a wide range of machine learning (supervised and unsupervised) algorithms for the structured data. However, the challenging task is to distribute the data and to devise distributed versions of these algorithms to fully exploit the underlying frameworks. This distribution effort can be further divided into two separate categories; One is parallelizing the algorithms, and the other is ensemble learning or parallel modeling techniques. We are exploring different algorithms namely, tensor factorization, association rule mining, decision trees and clustering. The aim is to provide a set of out-of-the-box algorithms to work with the structured data. As an example consider the implementation of a partitioning algorithm for RDF graphs given as NTriples. The algorithm uses the structure of the underlying undirected graph to partition the nodes into different clusters.

Listing 1.6. Example for Clustering RDF

```
import net.sansa_stack.ml.spark.clustering.RDFByModularityClustering
val numIterations = 100
val input ="path_to_your_RDFgraph"
val output ="path_name_for_clusters"
RDFByModularityClustering(spark, numIterations, input, output)
```

SANSA's clustering procedure follows a standard algorithm for partitioning undirected graphs aimed to maximize a modularity function. Usage examples and further information can be found at http://sansa-stack.net/faq/.

3.3 Semagrow query federation

Another component of the BDE Platform relevant to the semantics of the data being processed, is the *Semagrow* federation engine. Semagrow is a SPARQL query processing system that federates multiple remote endpoints. Semagrow hides schema heterogeneity by applying the appropriate vocabulary transformations and also uses metadata about the contents of the remote data sources to optimize querying plans [2]. Client applications are presented with a single SPARQL endpoint, and Semagrow transparently optimizes queries, executes sub-queries to the remote endpoints, dynamically integrating results in heterogeneous data models, and joins the partial results into the response to the original query and into the original query's schema.

In the context of integrating Semagrow in the BDE Platform, we have redesigned both the query planner and the execution engine so that it can be

Listing 1.7. SPARQL query that retrieves uniform event summaries.

```
PREFIX geo:<http://www.opengis.net/ont/geosparql#>
PREFIX ev:<http://bde.eu/man-made-changes/ont#>
PREFIX nev:<http://cassandra.semagrow.eu/events#>
SELECT ?id ?title ?loc ?name ?desc {
  ?e rdf:type          ev:NewsEvent ;
     ev:hasId          ?id ;
     ev:hasTitle       ?title ;
     ev:hasArea/geo:hasGeometry/geo:asWKT ?loc .
  ?s nev:event_id      ?id ;
     nev:description   ?desc ;
     nev:event_date    ?date .
  OPTIONAL{?s nev:tweet_post_ids ?tweet_post_ids}
  FILTER regex(?title, 'zaatari','i') .
}
```

Listing 1.8. Federated query retrieving events from Strabon & Cassandra.

```
SELECT ?id ?title ?loc ?name ?desc
{ ?e rdf:type      ev:NewsEvent ;
     ev:hasId      ?id ;
     ev:hasTitle   ?title ;
     ev:hasArea/geo:hasGeometry/geo:asWKT
          ?loc .
   FILTER regex(?title, 'zaatari','i') .
} @ Strabon
{ ?s nev:event_id      ?id ;
     nev:description   ?desc ;
     nev:event_date    ?date .
   OPTIONAL { ?s nev:tweet_post_ids
        ?tweet_post_ids }
} @ Cassandra
```

extended to support different querying languages. In cases where the target language is less expressive than SPARQL, Semagrow itself undertakes the required additional computations. For instance, when Apache Cassandra endpoints are included in the federation, the Semagrow query planner is aware of the fact that CQL, the Apache Cassandra query language, does not support joining across tables. The resulting plan is built in such a way that the tuples to be joined are fetched by different CQL queries to the same endpoint and joined at the Semagrow side. But this does not mean that only individual query patterns are fetched: "star" queries that fetch multiple properties of the same entity are expressible in CQL and the Semagrow planner takes into account that such patterns may be bundled together in one query [6]. Consider, for example, the SPARQL query in Listing 1.7 that involves geo-location from events stored in Strabon, a geospatial triple store that supports stSPARQL, and event summaries harvested from Twitter stored in Apache Cassandra.

Semagrow provides a transparent way to access and cross-join data from both sources. This is achieved by decomposing the initial SPARQL query into valid subqueries taking the capabilities and expressivity of each store into account, deciding the order of execution by estimating the cost of each alternative execution, and lastly translating each subquery into an appropriate query that the underlying system can understand. In our example the query in Listing 1.7 will be split into two subqueries, depicted in Listing 1.8 as blocks annotated with the name of the store to be executed. Note also that the subquery to be executed in Apache Cassandra is further translated into a valid CQL query. In order to scale out the query execution across different blades of a distributed infrastructure, we are re-implementing the execution engine over Apache Flink. (possibly on different machines) rather than different threads on the same machine.

4 BDE Platform Showcases

The main goal of the BigDataEurope project is to produce an easy-to-develop, -use and -adapt platform for wildly varying Big Data challenges. To validate this,

we develop pilot implementations in seven different domains, corresponding to the seven Societal Challenges (SC) of Horizon2020. These pilots are defined and developed by user partners in each of these challenges in collaboration with the technical team. A particular pilot comprises key data assets and domain-specific enabling technology in this domain and a BDE pilot implementation supports domain-specific workflows, exploration and visualization technologies. This has resulted in a versatile, but coherent set of demonstrators, which illustrate how relevant large-scale datasets or data-streams for the respective seven SC communities can be processed by the BDE infrastructure and provide novel insights that are promised by the Big Data community. We present two of these pilots, the BDE components used as well as an evaluation of the specific pilot and refer the reader to BDE deliverables[11] for details on all pilots.

Note that for the functional and non-functional requirements of the generic infrastructure part the *FURPS model* [5] is followed, classifying the software quality attributes with respect to Functionality, Usability, Reliability, Performance and Scalability. The details of each of these requirements are different for each challenge and were evaluated separately. At the same time, the generic BDE infrastructure was evaluated independently of these challenges according to the FURPS model. For each pilot, challenge-specific key evaluation questions were answered by the challenge partners. These were specified from generic evaluation questions corresponding to FURPS items. An example of such a question for SC1 is *'Are there currently vulnerabilities in the BDE infrastructure that might reveal any sort of communication to a 3rd party (e.g. queries and results, or IP addresses)?'*, relating to the functionality item.

4.1 SC1 Health, demographic change and wellbeing

The first pilot[12] in SC1 "Health, demographic change and wellbeing" implements the Open PHACTS Discovery Platform [10] for drug discovery on the BDE infrastructure.

The Open PHACTS Discovery Platform has been developed to reduce barriers to drug discovery in industry, academia and for small businesses. Researchers in drug discovery use multiple different data sources; Open PHACTS integrates and links these together so that researchers can easily see the relationships between compounds, targets, pathways, diseases and tissues. The Open PHACTS platform is a good example of a Big Data solution for efficient querying over a wide variety of large data sources that are integrated via an elaborate and mostly human curated process. The platform is founded on semantic web and linked data principles and uses industrial strength tools such as *Virtuoso*[13] to provide fast and robust access to the integrated chemistry and biological data sources. This integration effort resulted in a set RDF link sets, which map the large numbers of identifiers from these various sources and are stored in Virtuoso to answer queries from users. To simplify and scale access, an abstraction

[11] https://www.big-data-europe.eu/results/

[12] https://www.big-data-europe.eu/pilot-health/

[13] https://virtuoso.openlinksw.com/

layer using the *Puelia*[14] implementation of the *Linked Data API Specification*[15] is added to translate REST-full requests to instantiated SPARQL queries. The REST API is documented via the *OpenAPI specification*[16] (formerly Swagger) and available on the Open PHACTS Github repository[17].

The Open PHACTS pilot. The advantage of making the Open PHACTS functionality available as an instance on the BDE infrastructure is threefold:

- To improve security, an organisation might prefer to have the Open PHACTS functionality available on their own secure local cluster. The BDE infrastructure allows an open, almost 'one-click install' which makes this an affordable option for smaller companies and other organisations with a limited budget.
- The BDE approach, and in particular the Docker stack, provides a modular architecture where components can easily be replaced. For example, the current Open PHACTS platform uses the commercial cluster version of Virtuoso. The BDE stack makes it easy to replace this with another RDF store, for example, the open source 4Store or the Ontario and SANSA stack.
- The modularisation also makes it relatively simple to adapt for a different domain. One only has to have data available as RDF, create the link sets between the sources and describe the SPARQL templates which, via Puelia, provide the REST interface for the desired functionality.

In order to realise the Open PHACTS platform on the BDE infrastructure, all third party software and the functionally independent components from the Open PHACTS platform are 'dockerized' (cf. section 2). Two third party tools used by Open PHACTS are already available as Docker containers, namely *MemCached* and *MySQL*. For data storage and management, as an alternative to the commercial version of Virtuoso, it is possible to use the open source version (which is also available as a Docker container[18]) or 4Store integrated with the SANSA stack (cf. subsection 3.2). The remaining internally developed components[19] are:

- *OPS LinkedDataApi*, a Docker component containing the Puelia related code and the Swagger documentation generator.
- *Open PHACTS Explorer*, an HTML5 & CSS3 application for chemical information discovery and browsing. It is used to search for chemical compound and target information using a web search interface.
- *IdentityMappingService*, a Docker container of the native Open PHACTS service *queryExpander*, which includes the *IdentityMappingService* (IMS) and the *VoID Validator*.

The current pilot can be deployed on Linux and Windows, and the instructions can be found on the BDE Github repository[20]. BDE project Deliverable 6.3[4] provides an evaluation of all the pilots and shows that this pilot adheres to the

[14] https://code.google.com/archive/p/puelia-php/
[15] https://code.google.com/archive/p/linked-data-api/wikis/Specification.wiki
[16] https://www.openapis.org/
[17] https://github.com/openphacts/OPS_LinkedDataApi/blob/develop/api-config-files/swagger-2.0.json
[18] https://hub.docker.com/r/stain/virtuoso/
[19] Available from: https://hub.docker.com/r/openphacts/
[20] https://github.com/big-data-europe/pilot-sc1-cycle1

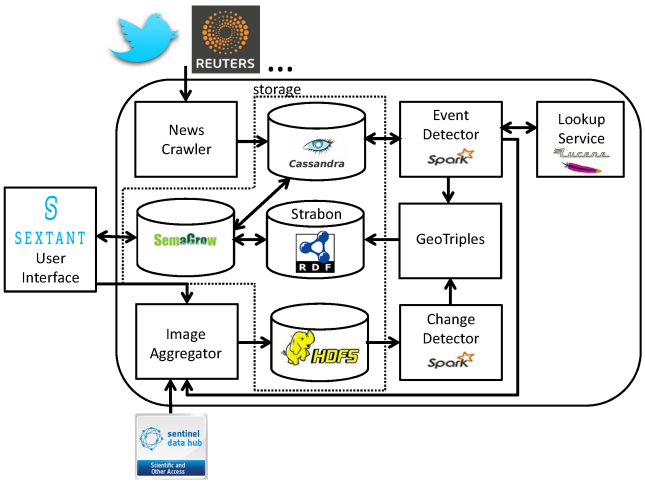

Fig. 8. High-level architecture of the SC7 pilot.

specified requirements, that the software can be easily deployed, and that the code is well documented. Community feedback has guided the next pilot cycle to a) broaden the community beyond drug research to include other data and b) add functionality to support extended domain requirements. In the next pilot cycle the existing datasets will be updated with new data from their sources, and functionality and data-sources to address the domain of food safety will be investigated.

4.2 SC7: Secure Societies

The SC7 pilot[21] combines the process of detecting changes in land cover or land use in satellite images (e.g., monitoring of critical infrastructures) with the display of geo-located events in news sites and social media. Integrating *remote sensing* with *social sensing* sources is crucial in the Space and Security domain, where useful information can be derived not only from Earth Observation products, but also from the combination of news articles with the user-generated messages of social media. The high-level architecture of the pilot is presented in Figure 8. In total, it comprises 11 BDE platform components, which can be grouped into the following three workflows:

– The *change detection workflow* consists of the three components at the bottom of Figure 8. The Image Aggregator receives the area and the time interval of interest from the UI and retrieves corresponding satellite images from ESA's *Sentinel Scientific Data Hub* (SciHub)[22]. The images are ingested into HDFS to be processed by the Change Detector, which relies on Spark

[21] https://www.big-data-europe.eu/pilot-secure-societies
[22] https://scihub.copernicus.eu

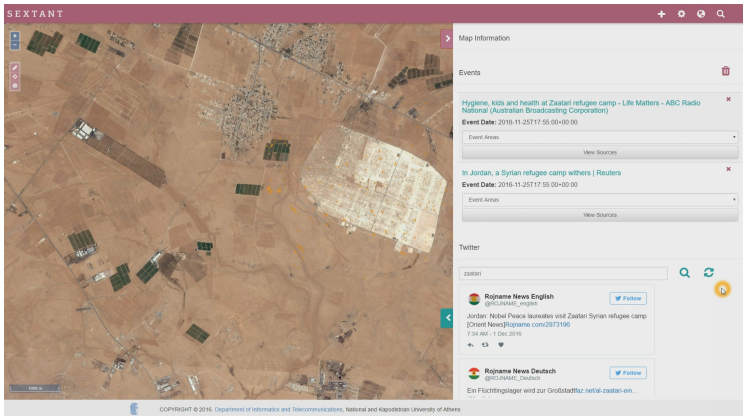

Fig. 9. Visualization of the change detection (displayed in orange on the map) and event detection results (clustered events and tweets in the right panel).

in order to apply a set of established operators efficiently and in parallel for comparing satellite images.

- The *event detection workflow* comprises the four components at the top of Figure 8. The News Crawler periodically checks various public news streams (Twitter, specific Twitter accounts, RSS feeds of Reuters[23]. These are stored in Cassandra in a way that abides by the corresponding privacy regulations. The Event Detector is periodically executed in order to cluster the news items into events using Spark for efficiency. In this process, special care is taken to associate every event with one or more geo-locations. The location names that are extracted from the text are associated with their geo-coordinates through a query in the Lookup Service, which indexes 180,000 location names from the *GADM dataset*[24] using Lucene.
- The *activation workflow* consists of the four components in the middle of Figure 8. *GeoTriples* [8] receives the detected areas with changes in land cover or use and summaries of detected events in order to convert their descriptions into RDF. These are stored in the spatio-temporal triplestore *Strabon* [7], which efficiently executes *GeoSPARQL* and *stSPARQL* queries. Semagrow federates Cassandra and Strabon, offering a unified access interface to *Sextant* [9], the user interface of the pilot. To meet all user requirements, Sextant has been significantly extended, allowing users to call both the change and the event detection workflows and to visualize their outcomes.

Figure 9 shows an example of refugee camps located in Zaatari, Jordan. The results from the change detection and the event detection workflows are displayed on Sextant using SemaGrow to retrieve information from Cassandra and Strabon in a uniform way (i.e., through the queries in Listings 1.7 and 1.8).

[23] http://www.reuters.com/tools/rss
[24] http://www.gadm.org

	Hortonworks	Cloudera	MapR	Bigtop	BDE Platform
File System	HDFS	HDFS	NFS	HDFS	HDFS
Installation	Native	Native	Native	Native	Lightweight Virtualization
Plug & play components	□	□	□	□	■
High Availability	SFR (yarn)	SFR (yarn)	MFR, SF	SFR (yarn)	MFR
Cost	Commercial	Commercial	Commercial	Open-source	Open-source
Scaling	Freemium	Freemium	Freemium	Free	Free
Extensibility	Difficult	□	□	□	■
Integration testing	■	■	■	■	□
Operating systems	Linux	Linux	Linux	Linux	All
Management tool	Ambari	Cloudera manager	MapR Control system	□	Docker swarm UI+ Custom

Table 1. Comparison of the BDE Stack with other Big Data distributions (SFR = Single failure recovery; MFR = Multiple failure recovery, SF = Self healing).

In general, variety is manifested in this pilot in the form of the different types of satellite images (e.g., SAR and optical ones) as well as the textual content from news agencies and social media. To address variety, useful information is extracted from these types of data by the change and the event detection workflows and is subsequently converted into searchable RDF triples by the activation workflow. Additionally, SemaGrow federates efficiently and effectively the access to the information that is stored in Cassandra (part of the original news text) and Strabon (RDF data).

5 Discussion and Conclusions

Table 2 outlines how the BDE Stack fulfills the initially derived requirements and indicates what measures support each requirement. In certain ways the BDE platform was inspired by the LOD2 Stack [1], which however used Debian packaging as deployment technology, since Docker was not yet available. Table 1 gives an overview of how the BDE Stack compares to major Big Data distributions Hortonworks, Cloudera, MapR and BigTop. The Plug-and-play components row describes how customized workflows can be realized. Extensibility means whether its possible to add custom components. The scaling refers to the requirement to pay additional licensing fees for cluster deployments. The comparison shows, that the BDE platform is complementary in many ways and especially with its extensibility, adaptability, consequent Docker-containerization and deployment as well its semantic layer goes far beyond the state-of-the-art. However, the interplay and integration with other Big Data platforms (especially Apache BigTop) is not only possible but increasingly advancing with related standardization efforts such as ODPi (`https://odpi.org`).

The advantages of semantics in distributed architectures appeal to both the Semantic Web community and to the Big Data processing community. The contribution the Semantic Web is SPARQL query processing that can scale over more voluminous query responses than what is currently possible. This makes Semantic Web approaches and representations a viable solution for new domains and applications, such as Earth observation (cf. subsection 4.2), where not only

Requirement	BDE Platform Measure
R1 Simplifying use	M1.1 Integration of Web UIs for the overall platform and individual components
	M1.2 Visual Big Data workflow authoring and monitoring
	M1.3 Unified and integrated semantic representation of data
R2 Easing Deployment	M2.1 Dockerization of components
	M2.2 Deployment orchestration using Docker Swarm
	M2.3 Support for various deployment schemes (individual machine, cloud, cluster)
R3 Managing heterogeneity	M3.1 Ingestion of heterogeneous data through RDF mapping and transformation
	M3.2 Direct operation of analytical algorithms for Big Data components on top of RDF data representations
	M3.3 Extensibility of the platform with custom, domain specific components
R4 Improving scalability	M4.1 Federated, parallelized query execution with Ontario and Semagrow
	M4.2 Pushing down of analytical queries on semantic representations to optimized Big Data analytics components with SANSA

Table 2. BDE Platform measures for addressing the requirements.

the underlying data but also the volume of the response is often orders of magnitude larger than the implicit limits of current SPARQL endpoints. For the Big Data processing community, the ability to join results from heterogeneous data stores allows integrating data and metadata. As an example, in the BigDataEurope SC5 climate modelling pilot numerical data in Hive and its provenance metadata in a triple store is joined in order to provide filters that refer not only to the data itself but also to provenance metadata regarding this data's origin.

References

1. Sören Auer, Volha Bryl, and Sebastian Tramp, editors. *Linked Open Data - Creating Knowledge Out of Interlinked Data - Results of the LOD2 Project*, volume 8661 of *Lecture Notes in Computer Science*. Springer, 2014.
2. Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. Semagrow: Optimizing federated SPARQL queries. In *SEMANTiCS*, 2015.
3. Big Data Europe. WP2 deliverable: Report on interest groups workshops III, 2016.
4. Big Data Europe. WP6 deliverable: Pilot evaluation and community specific assessment, 2016.
5. Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992.
6. Stasinos Konstantopoulos, Angelos Charalambidis, Giannis Mouchakis, Antonis Troumpoukis, Jürgen Jakobitch, and Vangelis Karkaletsis. Semantic Web technologies and Big Data infrastructures: SPARQL federated querying of heterogeneous Big Data stores. In *ISWC Demos and Posters Track*, 2016.
7. Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A semantic geospatial DBMS. In *ISWC*, pages 295–311, 2012.
8. Kostis Kyzirakos, Ioannis Vlachopoulos, Dimitrianos Savva, Stefan Manegold, and Manolis Koubarakis. Geotriples: a tool for publishing geospatial data as RDF graphs using R2RML mappings. In *ISWC Posters & Demonstrations Track*, 2014.
9. Charalampos Nikolaou, Kallirroi Dogani, Konstantina Bereta, George Garbis, Manos Karpathiotakis, Kostis Kyzirakos, and Manolis Koubarakis. Sextant: Visualizing time-evolving linked geospatial data. *J. Web Sem.*, 35:35–52, 2015.
10. Antony J. Williams, Lee Harland, Paul Groth, Stephen Pettifer, Christine Chichester, Egon L. Willighagen, Chris T. Evelo, Niklas Blomberg, Gerhard Ecker, Carole Goble, and Barend Mons. Open PHACTS: Semantic interoperability for drug discovery. *Drug Discovery Today*, 17(21–22):1188 – 1198, 2012.