Received: 9 January 2023 | Accepted: 8 August 2023

DOI: 10.1002/spe.3258

RESEARCH ARTICLE

WILEY

NORA: Scalable OWL reasoner based on NoSQL databases and Apache Spark

Antonio Benítez-Hidalgo | **Ismael Navas-Delgado** | **María del Mar Roldán-García**

KHAOS Research, ITIS Software, Universidad de Málaga, Málaga, Spain

Correspondence
Antonio Benítez-Hidalgo, KHAOS Research, ITIS Software, Universidad de Málaga, Málaga, Spain.
Email: antonio.b@uma.es

Funding information
AETHER-UMA, Grant/Award Number: PID2020-112540RB-C41; Spanish Ministry of Science, Innovation and Universities, Grant/Award Number: PRE2018-084280; Universidad de Málaga / CBUA

Abstract

Reasoning is the process of inferring new knowledge and identifying inconsistencies within ontologies. Traditional techniques often prove inadequate when reasoning over large Knowledge Bases containing millions or billions of facts. This article introduces NORA, a persistent and scalable OWL reasoner built on top of Apache Spark, designed to address the challenges of reasoning over extensive and complex ontologies. NORA exploits the scalability of NoSQL databases to effectively apply inference rules to Big Data ontologies with large ABoxes. To facilitate scalable reasoning, OWL data, including class and property hierarchies and instances, are materialized in the Apache Cassandra database. Spark programs are then evaluated iteratively, uncovering new implicit knowledge from the dataset and leading to enhanced performance and more efficient reasoning over large-scale ontologies. NORA has undergone a thorough evaluation with different benchmarking ontologies of varying sizes to assess the scalability of the developed solution.

KEYWORDS

Apache Spark, knowledge base, NoSQL, OWL, reasoner

1 | INTRODUCTION

The Semantic Web is an extension of the World Wide Web proposed by the World Wide Web Consortium (W3C). The main goal of the Semantic Web is to enable computers to understand the meaning of the data available on the Internet. This vision has been evolving into the concept of the Web of data. This evolution has been done through a technology stack to support this goal. This vision is being materialised through a set of standards for ontologies such as OWL 2 (mainly for TBox, i.e., terminologies) and RDF (mainly for ABox, i.e., assertions). Thus, Semantic Web users have a wide range of possibilities to create data stores on the Web, build vocabularies, and write rules for handling data.

However, as the Web is not a set of isolated Web pages, the goal is to enable ways to connect different data repositories instead of having isolated datasets. In this sense, the Linked Data approach proposes connecting related datasets in the same domain. In line with this concept, FAIR Data principles (Findability, Accessibility, Interoperability, and Reuse) propose a set of requirements to improve how data is published. These recommendations include using standards for

Abbreviations: DL, Description Logic; HDFS, Hadoop Distributed File System; KB, Knowledge Base; NoSQL, Not Only SQL; OWL, Web Ontology Language; RDD, Resilient Distributed Dataset; RDF, Resource Description Framework; SPARQL, SPARQL Protocol and RDF Query Language; SWRL, Semantic Web Rule Language; W3C, World Wide Web Consortium.

metadata such as OWL and RDF and public registries for the data and metadata to enable the findability of relevant data. In this regard, the Linked Open Data Cloud includes 1255 datasets with 16,174 links (as of May 2020[*]), which is the largest registry of Linked Data in history. The Linked Open Data Cloud references repositories counting billions of RDF triples that open the possibility of obtaining new knowledge through inference mechanisms, thus improving Linked Data applications. Therefore, the Linked Open Data Cloud and the requirement to publish data as FAIR Data in public funding projects produce many RDF datasets. This public data has become essential for today's global economy, which is fundamentally driven by data.

The semantic Web standard for ontology description is OWL 2, which is based on Description Logic. Using a formal specification of an ontology enables the use of reasoning mechanisms. These mechanisms can derive facts that are not explicitly expressed in the ontology. Thus, automatic processing tools, reasoners, can apply a set of formalisms to discover new knowledge or inconsistencies in ontologies. Due to properties such as decidability or computational complexity, computing limitations can limit reasoners in some scenarios. Thus, it is known that reasoning over large TBoxes with complex ontologies is not feasible. Even simple reasoning phases on large ABoxes cannot be feasible with limited computing resources. Therefore, reasoning techniques are not usually scalable, so we cannot use them with the size of the Linked Data or single large RDF datasets. Therefore, the problem is that we cannot find solutions that provide scalable reasoning on Linked Data in practice. On the one hand, due to the volume of Linked Data and its continuous evolution, this reasoning should be distributed and incremental. On the other hand, if we talk about distribution and scalability, we immediately think of Big data technologies.

One approach to deal with the problem of reasoning over large datasets is to materialise the reasoning on databases and use this as a data view to access the data, including the facts discovered by the reasoner. However, Relational Database Management Systems can be insufficient to deal with massive datasets or the need to reason over changing data that can impose modifications in the data structure. Big Data technologies allow us to store and process large amounts of information. Specifically, NoSQL databases are designed to scale horizontally to provide good performance in data retrieval and flexible mechanisms to adapt to changing data schemas. Meanwhile, other approaches like Apache Spark enable the processing of such large datasets, taking advantage of parallelisation mechanisms. In this context, Big Data technologies are a promising approach to deal with the reasoning and management of large RDF datasets of Linked Data, including several related RDF datasets. To the best of our knowledge, there are no existing proposals implementing highly scalable Linked Data/OWL reasoning based on NoSQL databases and Apache Spark.

With this challenge in mind, we designed an architecture aimed at high scalability in the context of Semantic Web technologies. In this article, we present NORA (***No****SQL-based* ***R****easoner in* ***A****pache Spark*), a persistent and scalable OWL reasoner, aimed at providing efficient reasoning support for (Big Data) ontologies. NORA adopts a materialised inference architecture that relies on a highly available and scalable NoSQL database to persist OWL data. Apache Spark is then used to perform OWL reasoning over large ontologies. As a result, this approach can validate semantically enabled datasets and discover new (not explicit) knowledge in large RDF datasets of Linked Data. NORA is implemented in Java under the MIT license, and its source code is publicly available in GitHub.[†]

This article is organised as follows. Section 2 delves into related background concepts about Semantic Web Technologies and relevant tooling. In Section 3, a review of related work is present, with Section 4 providing a more detailed description of NORA architecture and its main components. Use cases are described in Section 5, with Section 6 devoted to conclusions and future work.

2 | BACKGROUND

2.1 | Semantic web technologies

Ontologies, as defined in the W3C standard stack of the Semantic Web,[‡] represent the concepts and relationships used to describe and represent a given domain. The ontology specification is provided through a formal language that allows information processing not only by humans but also by software systems in an automated way. The knowledge included in an ontology must result from an agreement between specialists or users within a workgroup restricted to a particular application domain, an element of common interest or a given field.

[*] https://lod-cloud.net/.

[†] https://github.com/benhid/nora.

[‡] https://www.w3.org/standards/semanticweb/.

An RDF[1] data model is used to publish and link structured data on the Web using Semantic Web technologies. This data model is based on a directed graph formed by triplets of type subject-predicate-object, where the graph nodes are the subject and object, and the edges are the predicates. SPARQL is a query language for RDF graphs.[2] SPARQL queries use RDF patterns to retrieve the set of triples in the RDF repository that match such patterns.

The Web Ontology Language (OWL)[3] is the more well-known ontology language for defining and instantiating ontologies. From the perspective of first-order logic, Gruber[4] identified five types of ontology components: classes, relations, functions, formal axioms, and instances (also referred to as *individuals*). OWL is built on top of RDF data; in other words, the OWL specification defines what you can write with RDF to make it possible to have valid ontologies.

SWRL (Semantic Web Rule Language) provides the OWL ontologies with procedural knowledge compensating for some limitations of ontology inference, particularly in identifying semantic relationships between individuals. The SWRL is used to construct rule expressions in the form of "Antecedent ⇒ Consequent" to represent those semantic relationships. Antecedent and consequent are formulated as conjunctions of elements associated with one or more attributes. They are denoted as a question mark and a variable (e.g., ?*x*) in the rule. These variables will be instantiated to concrete instances (ABox) in the rule evaluation process. Antecedent and consequent could be empty, but the typical case will define both. In this case, where a set of individuals instantiating the variables make the antecedent true, these individuals will also be instantiated in the consequent. These instances will be classified in classes where they were not initially classified. In 2012, Google introduced the term "knowledge graph",[§] whose core idea is to use graphs to represent data, possibly enhanced by the use of ontologies to describe the nodes and edges. Enterprise knowledge graphs have been actively developed in a variety of industries, such as e-commerce,[5] social networking,[6] and finances.[7] Open and proprietary knowledge graphs are often built upon several data sources and thus can grow massively large (with billions of nodes and edges). However, using traditional techniques, deductive reasoning using inference rules can be challenging to apply to large-scale graphs.

2.2 | Apache Cassandra

There is a wide range of database solutions to deal with large amounts of data. In choosing between relational or NoSQL, many differences must be considered. Regarding scalability, relational databases are vertically scalable, meaning they can handle increasing load by scaling up a single server. In contrast, NoSQL databases are horizontally scalable (also known as *sharding*), so they can handle increasing computing needs by adding more machines (nodes of a cluster) to the pool of available resources. This horizontal scalability is of particular interest in Big Data applications where the vertical scalability is not enough to maintain the response time under increasing computing requirements.

Apache Cassandra[8] is a NoSQL, lightning-fast distributed column-oriented database management system. Figure 1 depicts the core Cassandra data model components. At a high level, a keyspace defines data replication on nodes. A cluster in Cassandra contains one keyspace per application. The data is organised into tables horizontally or vertically (as rows or columns, respectively). Following this model, the primary key of a table in Cassandra can be a compound key of several columns. When using compound keys, the first key is called the partition key and is responsible for uniquely identifying a record in the table and for data distribution across the nodes. The rest are called the clustering key and are responsible for data sorting within the partition.

2.3 | Apache Spark

Apache Spark[9] is the world's leading distributed analytics platform. By keeping data in memory for fast processing, Spark can improve performance by an order of magnitude in a broad class of applications compared to traditional systems. Spark implements the master-slave architecture, where one central coordinator (or driver) communicates with executors, the processes that run computations and store data for that application.

Spark provides a high-level abstraction for working with distributed data across a cluster: RDDs (*Resilient Distributed Dataset*)[10] (see Figure 2). RDDs provide a range of coarse-grained transformations, such as *map* and *join*, that produce new RDDs from the existing RDDs. By design, all transformations in Spark are lazy. Thus no computation is executed right away. RDDs can express existing programming models such as MapReduce and iterative MapReduce.

[§] https://www.blog.google/products/search/introducing-knowledge-graph-things-not/.

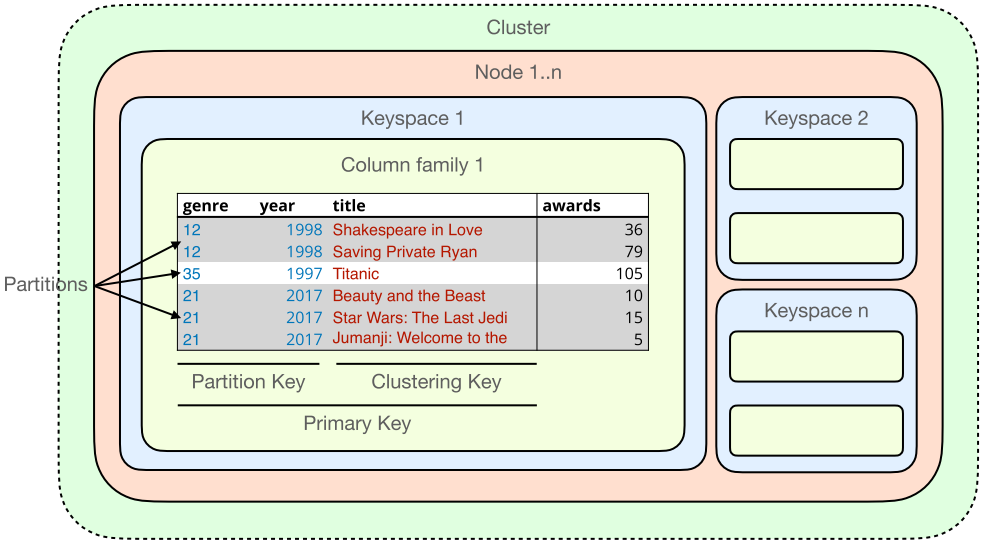

FIGURE 1 Apache Cassandra data model components include keyspaces, tables (also referred to as *column families*), and columns.

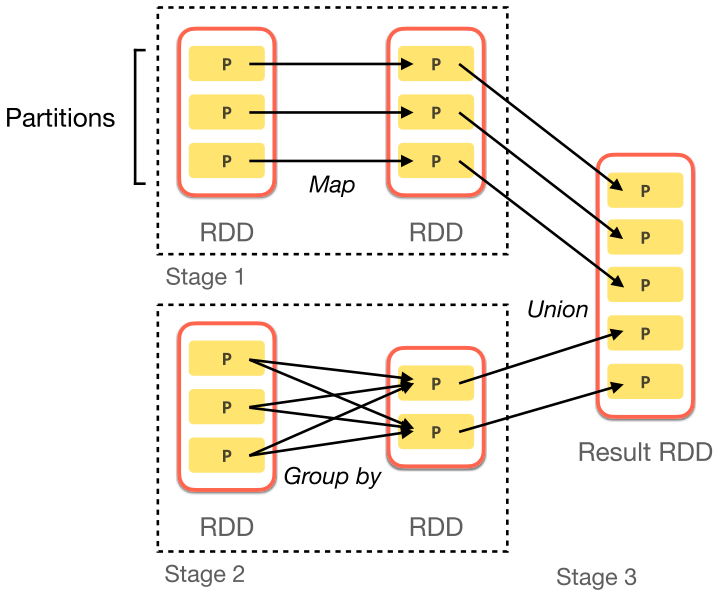

FIGURE 2 Example of Apache Spark's transformations over RDDs. RDDs are a collection of partitions (i.e., atomic chunks of data) distributed across different nodes in a cluster. Partitions are basic units of parallelism in Apache Spark.

3 | RELATED WORK

Using ontologies as knowledge representation formalism and exploiting its reasoning capabilities for inferring knowledge is not new. Over the past few decades, researchers in artificial intelligence have developed many techniques for representing knowledge, including using ontologies as knowledge representation formalisms. Furthermore, this community has defined complete and sound reasoning algorithms for inferring new knowledge from the knowledge explicitly represented by the ontologies. The complexity and efficiency of these algorithms have been studied at length. These traditional reasoners were main-memory oriented and more focused on completeness and soundness than on scalability. Pellet,[11] Hermit,[12] or Racer,[13] among others, are the most representative systems in this category.[14,15]

With the explosion of the Semantic Web, many disk-oriented proposals using relational databases for implementing OWL reasoning appeared. These systems provided ontology, data persistence, and some scalability while sacrificing

completeness.[16] gives a review of this type of reasoners. DBOWL[17] and Stardog[¶] were reasonable attempts to develop an OWL-DL complete reasoner based on relational databases technology.

Centralised systems cannot deal with the Web of data, the Big Data, and the exponential growth of published Linked Data. However, we cannot find much research on using different database paradigms, such as NoSQL databases, to build scalable RDF triple stores or OWL reasoners. There exist several systems to process RDF triples with these technologies, such as Jena-Hbase,[18] H2RDF[19] and CumulusRDF.[20] Jena-Hbase and H2RDF, for example, use Apache HBase, a NoSQL column family store, as the underlying store layer for RDF data. CumulusRDF follows a cloud-based architecture, with Apache Cassandra being used to store RDF triples. In Reference 17, the authors propose a tool for materialising OWL ontologies in a NoSQL database. Ontologies are stored in a Cassandra database along with its pre-computed class and property hierarchies.

Furthermore, the technological advance towards artificial intelligence suggests exploring distribution and parallelism to provide scalable OWL reasoning. Proposed systems are usually limited to some OWL fragment or a specific description logic.[17] uses a materialisation schema and a reasoning model that were the first step toward an OWL reasoner based on the Hadoop MapReduce paradigm by exploiting the inherent scalability of NoSQL databases.[21] studies different approaches to implement parallel and distributed reasoning algorithms for the OWL EL profile.[22] describes Deslog, a parallel reasoner for the ALC description logic. More recently,[23] presents a survey of large-scale reasoning on the Web of data. The evolution of the OWL reasoners could not prevent the proliferation of the Big Data technologies, such as the Apache Hadoop Framework[#][24,25] and Apache Spark.[26,27]

WebPIE applies *map and reduce* operations over RDF data to support reasoning by exploiting Hadoop Distributed File System (HDFS). However, its approach does not consider data partitioning. WebPie treats data as a whole without partitions, spending so much time reading and processing the entire data set. It also implies some limitations in scaling up when dealing with large datasets. NORA, on the other hand, uses Spark to leverage data partitioning and access, making it more efficient with large datasets.

Cichlid[26] is a reasoning system for RDFS and OWL, based on Spark. Its architecture shares several similarities with our proposal (such as using Spark to leverage data partitioning and access). It differs, among other things, on the storage layer. Cichlid claims to be around ten times faster than other state-of-the-art distributed reasoning solutions by storing data in memory, which has severe limitations when dealing with large datasets that could not fit into memory.

Likewise, SPOWL[27] is a Spark-based OWL 2 reasoner. SPOWL uses HDFS to persist data. While HDFS, together with Spark, is well suited for this task, it lacks record-level indexing, which is often bound to give lower query response times. Our proposal, by using Apache Cassandra, which provides tunable consistency and allows for record-level indexing, provides a higher query response time.

More recently, Mohamed et al. introduced the SANSA framework.[28] Their approach shares many similarities with ours, combining multiple techniques to achieve scalable reasoning over large-scale OWL datasets using Spark. However, they use data stored in memory. Memory capacity becomes a bottleneck, and the whole system's performance could degrade significantly if the size of the data exceeds the available memory. On the other hand, NORA uses Apache Cassandra for data storage, a NoSQL database renowned for its capability to handle large datasets. Apache Cassandra distributes data across many nodes, thereby mitigating the memory limitation. Furthermore, data stored in memory is transient and can be lost in the event of a failure. This necessitates additional steps for data recovery. NORA's use of Cassandra offers data persistence, ensuring data is safely stored and easily recoverable, which provides an extra layer of security and robustness.

The previous works evidence the popularity of the Spark framework and its application in distributed reasoning systems. However, to the best of our knowledge, the approach presented in this article is the first to successfully combine a NoSQL database with Spark in order to implement a scalable OWL reasoner (Table 1). The benefits of using a NoSQL database, such as Apache Cassandra, include improved performance, horizontal scalability, and flexibility in handling large datasets. These advantages, coupled with the power of the Spark framework, allow our system to efficiently process and reason over massive knowledge bases, addressing some of the limitations faced by existing solutions.

4 | SYSTEM ARCHITECTURE

The core NORA's architecture consists of two steps: an OWL parser (including a bulk loader for ontology instances) and an OWL reasoning engine. These services work together effectively to achieve scalable reasoning. The process is

[¶]https://www.stardog.com/.

[#]https://hadoop.apache.org/.

TABLE 1 Popular reasoning systems with Big Data support.

Name	Version	Language	Fragment	Storage layer	Processing	Parallel	Distributed
WebPIE	1.1.1	Java	RDFS/OWL	HDFS	MapReduce	✓	✓
Cichlid	-	Scala	RDFS/OWL	In Memory	Apache Spark	✓	-
SPOWL	-	Java	OWL 2 RL	HDFS	Apache Spark	-	-
SANSA framework	0.7.1	Scala	RDF/OWL	HDFS	Apache Spark	✓	-
DBOWL	0.0.1	Java	OWL-DL	Oracle	SQL views	✓	✓
NORA	1.0.0	Java	OWL 2	Cassandra	Apache Spark	✓	✓

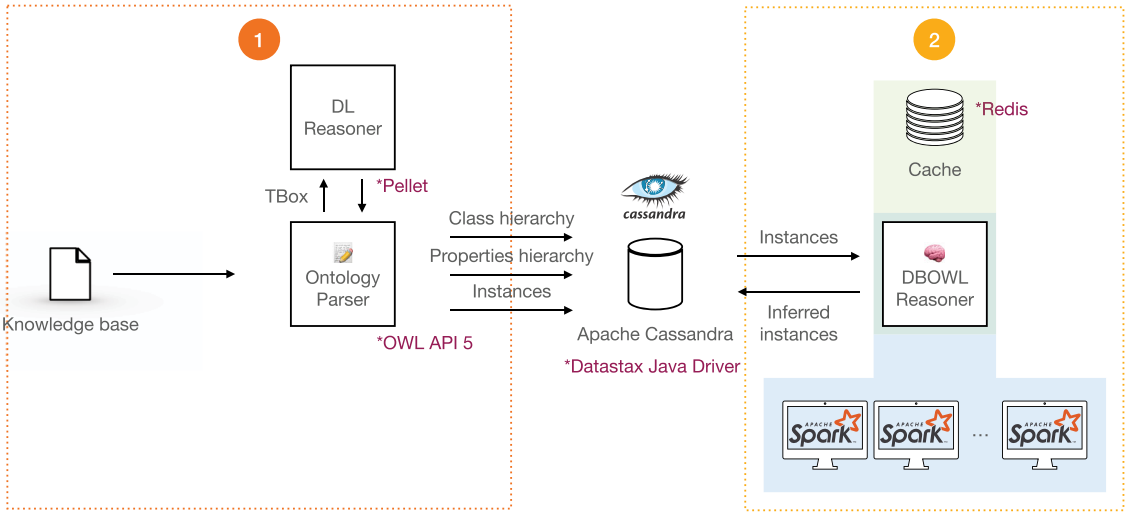

FIGURE 3 The high-level architecture of NORA.

summarised in Figure 3. In the first step, data obtained from any Knowledge Base (KB), represented as an OWL ontology, is materialised in a Cassandra database. A description logic (DL) reasoner is used to pre-compute complete subsumption relations regarding the T-Box from an OWL file (the starting point). We use the OWL API, a Java interface and reference implementation for the OWL language, to read the input OWL ontologies. Pellet, a well-known OWL 2 DL reasoner, is used to classify on the TBox. By combining these tools, we obtain several taxonomies, such as the class/subclass and property/subproperty hierarchies, but no newly derived knowledge is inferred from the ABox. Subsequently, a database schema is created to materialise all this information.[17] As a result, the reasoned TBox is stored in Cassandra together with the explicit instances described in the Knowledge Base. Section 4.1 provides a more detailed description of this strategy.

In the second step, the reasoning in the ABox occurs. We create an RDD from a specific table to read data from the database. This abstraction allows us to manipulate the data spread across many machines in a distributed fashion. The NORA reasoner engine powered by Apache Spark evaluates the inference rules, expressed as Spark jobs, by accessing the database with materialised data. The inference rules are applied repeatedly until no further inferred statements are produced. Under the hood, an external cache, Redis[‖] in our case, is used to speed up this operation by loading information from the database into the cache for fast lookup operations. Section 4.2 provides a more detailed description of this strategy.

4.1 | Data materialisation

NORA persists OWL data in a NoSQL database, Apache Cassandra. The idea of working with NoSQL databases for storing large volumes of RDF-like data has been explored in the past.[17,20,29-31] We have chosen Apache Cassandra over other

[‖]https://redis.io.

software implementations of NoSQL databases for several reasons. None of these reasons is unique to Cassandra, but it is the only system that incorporates many of them. For example, like most NoSQL databases, Cassandra is massively scalable by simply increasing the number of nodes. Cassandra goes beyond and has a masterless architecture, where all nodes are equally important. This fact directly contributes to the cluster's resilience, which is of particular interest in Big Data applications, and implies that multiple nodes can fail without impacting the global availability of the system. Furthermore, Cassandra is also designed for heavy writes, which are unaffected by large volumes of data. Performance-wise, the Cassandra Query Language (CQL) also defines prepared statements, that is, statements that are repeated multiple times can be pre-parsed, validated once and executed with a minimal impact on performance.

All of these benefits are not without concessions. NoSQL databases, in general, and Cassandra, in particular, lack relational features such as joins and other join-like operators. Dropping these features allows Cassandra to scale across multiple nodes easier. Some solutions exist to overcome this barrier by enabling joins efficiently over such databases, which will be highlighted later.

Many concepts in Cassandra are closely related to its relational counterpart. In Cassandra, a database is known as a keyspace. Keyspaces contain one or more column families (tables), which organise data into rows. Our proposed optimised schema is based on our previous work as described in,[17] but with some refinements to ease data management. We use a compound primary key consisting of a partition key (responsible for determining which node stores the row) and a clustering key (accountable for data sorting within the partition) to allow multiple rows with the same identifier. The combination of partition and clustering key(s) enables performant reads/writes by distributing data efficiently across the cluster. In our implementation, we have addressed a comprehensive OWL fragment, including many essential OWL constructs, for example, intersectionOf, unionOf, and complementOf, support for cardinality restrictions (such as minCardinality and maxCardinality), etc. These constructs are stored in their corresponding column families within Cassandra. We defined 29 column families in Cassandra to store all the OWL data. The column families are:

ClassIndividuals	***IsComplementOf***	***IsDisjointWith***
IsEquivalentToAll	*IsEquivalentToClass*	*IsEquivalentToIntersection*
IsEquivalentToMaxCardinality	*IsEquivalentToMinCardinality*	*IsEquivalentToSome*
IsEquivalentToUnion	*IsFunctionalProperty*	*IsInverseFunctionalProperty*
IsObjectPropertyDomain	*IsObjectPropertyEquivalentTo*	*IsObjectPropertyRange*
IsSameAs	*IsSubclassOfAll*	*IsSubclassOfClass*
IsSubclassOfComplementTo	*IsSubclassOfIntersection*	*IsSubclassOfMaxCardinality*
IsSubclassOfMinCardinality	*IsSubclassOfSome*	*IsSubclassOfUnion*
IsSubPropertyOf	*IsSuperClassOf*	*PropIndividuals*

For example, the *ClassIndividuals* column family materialises the individuals w.r.t. its class in the ontology (see Table 2). In this case, the partition key is *cls* (the class to which the instance belongs). Thus, all the instances of the previous class are stored in the cluster node, making the operation of retrieving instances for each class more efficient. The clustering key is *num* (the order in which this instance has been found in storing it). This order is critical in the process of discovering new instances in the reasoning process. For nested class descriptions defined as the intersection or union of two or more descriptions, we generate a random internal identifier and use it as a partition key in the column family. To illustrate the whole process, let us describe the following set of axioms:

$$\text{Tbox} = \text{WomanCollege} \equiv \text{College} \sqcap \forall \text{hasStudent}.\neg\text{Man}.$$

$$\text{Abox} = \text{WommanCollege(college1)}, \text{College(college2)}, \text{hasStudent(college2,professor1)}.$$

Figure 4 shows how the former Tbox and Abox axioms are materialised in the database. We have two instances of class WomanCollege and College, namely college1 and college2. The property hasStudent is also stored in the proper column family. To keep the class description in the Tbox, we generate random identifiers (_uid1 and _uid2) and use them as partition keys in their corresponding column family.

TABLE 2 Proposed materialisation schema for storing Tbox and Abox axioms. For the sake of simplicity, only some axioms are shown.

		Schema		
Axiom	**Column family name**	**Column name**	**Type**	**Datatype**
Class Axioms				
$A \sqsubseteq B$	IsSubclassOfClass	*cls*	Partition key	*text*
		num	Clustering key	*int*
		supclass	Regular	*text*
$A \equiv B$	IsEquivalentToClass	*cls*	Partition key	*text*
		num	Clustering key	*int*
		equiv	Regular	*text*
$A \sqsubseteq \neg B$	IsDisjointWith	*cls*	Partition key	*text*
		num	Clustering key	*int*
		ind1	Regular	*text*
$A \equiv \neg B$	IsComplementOf	*cls*	Partition key	*text*
		num	Clustering key	*int*
		complement	Regular	*text*
Abox Axioms				
$A(a)$	ClassIndividuals	*cls*	Partition key	*text*
		num	Clustering key	*int*
		individual	Regular	*text*
$P(a, b)$	PropIndividuals	*prop*	Partition key	*text*
		num	Clustering key	*int*
		domain	Regular	*text*
		range	Regular	*text*
$a_1 \equiv a_2$	IsSameAs	*ind*	Partition key	*text*
		num	Clustering key	*int*
		same	Regular	*text*

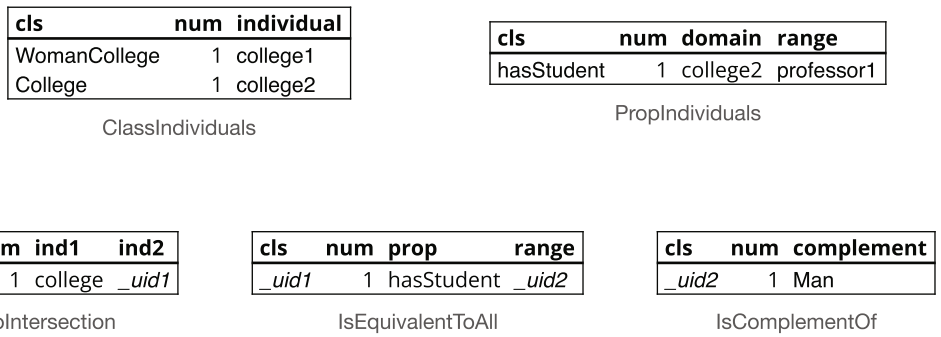

FIGURE 4 Materialised axioms from Example 1. All column families are represented.

4.2 | Reasoning strategy

Reference 17 proposes the storage of KBs in Cassandra, but the reasoning is not provided. Thus, there is a limitation as the NoSQL database offers efficient access to the KB, but only for explicit knowledge. NORA's reasoning engine implements ABox reasoning by translating each inference rule as an Apache Spark expression. The reasoning engine evaluates these Spark programmes iteratively until no new inferred knowledge is derived. The Spark connector for Cassandra can read data (columns and rows) from a keyspace and run standard Resilient Distributed Dataset (referred to as an RDD) methods to interact with Cassandra directly. RDD transformation includes *map*, *sort* and *join* operations which are not available in Cassandra natively. Listing 1 shows how a view of a column family can be loaded by using Spark and the DataStax Spark Cassandra Connector. Each row is converted to a serializable object in Java, and a representation of key-row pairs is returned.

Listing 1 Read Cassandra column family from Java with Apache Spark.

```
JavaPairRDD<String, ClassIndividuals.Row> classIndividualsRDD = javaFunctions(spark)
    .cassandraTable(connection.getDatabaseName(), "classindividuals",
            CassandraJavaUtil.mapRowTo(ClassIndividuals.Row.class))
    .keyBy((Function<ClassIndividuals.Row, String>) ClassIndividuals.Row::getCls);

JavaPairRDD<String, IsSubclassOfClass.Row> isSubclassOfClassRDD = javaFunctions
(spark)
    .cassandraTable(connection.getDatabaseName(), "issubclassofclass",
            CassandraJavaUtil.mapRowTo(IsSubclassOfClass.Row.class))
    .keyBy((Function<IsSubclassOfClass.Row, String>) IsSubclassOfClass.Row::getCls);
```

Listing 2 Reasoning algorithm for inferring sub-classes of classes

```
JavaPairRDD<String, String> firstJoinRDD = isSubclassOfClassRDD
    .join(classIndividualsRDD)
    .mapToPair(tuple -> {
        Tuple2<IsSubclassOfClass.Row, ClassIndividuals.Row> secondOperand =
tuple._2();

        IsSubclassOfClass.Row isSubclassOfClassRow = secondOperand._1();
        ClassIndividuals.Row classIndividualsRow = secondOperand._2();

        return new Tuple2<>(isSubclassOfClassRow.getSupclass(), classIndividualsRow
.getIndividual());
    });
```

TBox axioms can be rewritten in terms of join expressions which Spark can handle. We can join multiple column families using Spark's RDD API by selecting rows matching keys in both relations. We can apply multiple joins over the same column family and return only the final result to the driver program (lazy evaluation). NORA includes over 29 Spark programmes to perform the reasoning, available at the GitHub repository.[**] In order to illustrate how they operate, we will describe and show an example of two of them.

In Listing 2, we use a join operation to combine *ClassIndividuals* and *IsSubclassOfClass* column families to infer new sub-classes for the individuals in the keyspace. Figure 5 shows an example of this process. In this example, our Knowledge Base declares that Airport is a sub-class of Infrastructure. We also know that Madrid Barajas, Paris City Airport and New York Airport are instances of Airport and that New York Airport is an Infrastructure. If we join both column families, our

[**] https://github.com/benhid/nora.

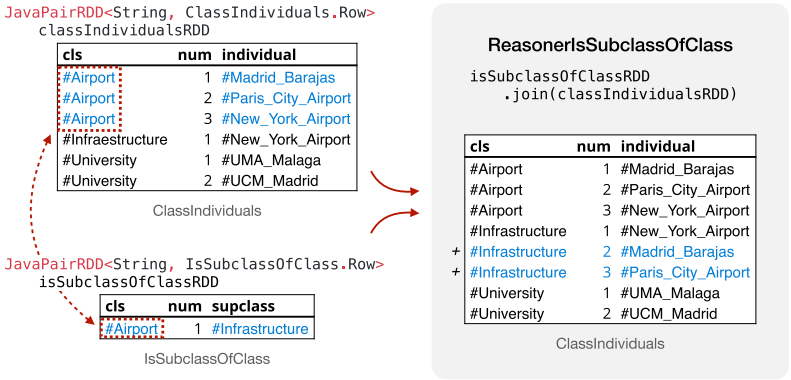

FIGURE 5 Join operation between *ClassIndividuals* and *IsSubclassOfClass* column families. Matching rows are classified as new individuals in the Knowledge Base and inserted in the *ClassIndividuals* column family if they do not exist yet.

Listing 3 Reasoning algorithm for inferring subclasses

```
JavaPairRDD<String, Tuple3<IsSubclassOfAll.Row, String, String>> firstJoinRDD =
isSubclassOfAllRDD
    .join(classIndividualsRDD)
    .mapToPair(tuple -> {
        String cls = tuple._1();

        Tuple2<IsSubclassOfAll.Row, String> secondOperand = tuple._2();
        String individual = secondOperand._2();
        IsSubclassOfAll.Row isSubclassOfAll = secondOperand._1();

        return new Tuple2<>(isSubclassOfAll.getProp(), new Tuple3<>(isSubclassOfAll
, individual, cls));
    });

JavaPairRDD<String, String> secondJoinRDD = firstJoinRDD
    .join(propIndividualsRDD)
    .mapToPair(Tuple2::_2)
    // Filters out rows where individual == domain
    .filter(tuple -> {
        Tuple3<IsSubclassOfAll.Row, String, String> firstOperand = tuple._1();
        String individual = firstOperand._2();

        PropIndividuals.Row propIndividualsRow = tuple._2();

        return individual.equals(propIndividualsRow.getDomain());
    })
    // Transforms to <range, range>
    .mapToPair(tuple -> {
        Tuple3<IsSubclassOfAll.Row, String, String> firstOperand = tuple._1();
        IsSubclassOfAll.Row isSubclassOfAllRow = firstOperand._1();

        PropIndividuals.Row propIndividualsRow = tuple._2();

        return new Tuple2<>(isSubclassOfAllRow.getRange(), propIndividualsRow.
getRange());
    });
```

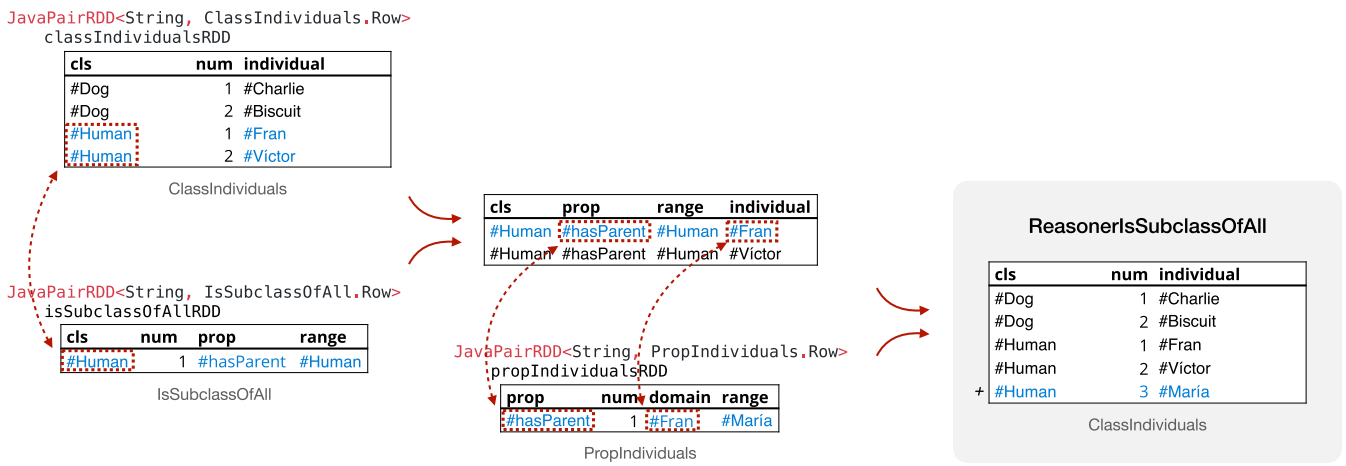

FIGURE 6 Join operation between *ClassIndividuals*, *IsSubclassOfAll*, and *PropIndividuals* column families to infer new individuals belonging to the Human class.

reasoning algorithm will infer that Madrid Barajas and Paris City Airport are sub-classes of Infrastructure as well. New York Airport being sub-class of Infrastructure is also inferred, but it is discharged as no new knowledge is produced in this step.

In many cases, reasoning is not as straightforward as simply loading and joining various column families. For example, the reasoning of *owl:subClassOf* property involves several column families, namely *IsSubclassOfAll*, *ClassIndividuals* and *PropIndividuals*. This process is shown in Listing 3. First, RDDs are created for each of these column families. Then, *IsSubclassOfAll* and *ClassIndividuals* are joined (first join), and an RDD is returned. This RDD, together with the *PropIndividuals* RDD, is joined (second join) using the property (from *IsSubclassOfAll* and *PropIndividuals* column families) as matching key. After that, the RDD is filtered, as we are only interested in those rows in which the individual (from *ClassIndividuals* column family) is the same as the domain (from *PropIndividuals* column family). Finally, the RDD is transformed, and a tuple of ranges (from *IsSubclassOfAll* and *PropIndividuals* column families) is returned. Figure 6 shows an example of this process. In this case, the Knowledge Base describes Humans, where a Human's parent is also a Human. For example, Fran, a Human, has a parent, María. Using this explicit knowledge, we can infer that María is a Human by joining the three column families above and filtering out the results.

To avoid redundancy in the database (i.e., duplicate entries), when a new inference is found, we need to determine whether the individual is already stored in the column family. The lookup is performed by querying a fast in-memory key-value store, which is pre-populated with the individuals from the KB at the beginning of the reasoning process. This lookup helps to reduce the number of database read operations, potentially avoiding network timeouts or other latency-related issues. If the individual is found in the cache, the inference is already present in the database, and we can skip the insertion: no further actions are required. Otherwise, it is added to the cache and a counter of the number of individuals belonging to the class is incremented. This value is used to perform the insertion.

5 | USE CASES

In this section, we present several use cases to illustrate the use of our proposal and show examples of the ABox reasoning involved. For evaluating the reasoner, we focus on assessing its performance and ability to provide correct inferences by employing two benchmarks. The first one is the well-known Lehigh University Benchmark for benchmark tests.[32] The second one entails a more expressive scenario using the University Ontology Benchmark.[33] These benchmarks help us to better understand the system's efficiency and its potential to generate accurate results under varying conditions. We performed our experiments 25 times and reported the mean value. To this end, we used a computing Spark cluster of 7 nodes (a master node and 6 slave nodes), each equipped with 100 GB RAM and 8 cores, running Ubuntu 16 and Apache Spark 3.3.0. Apache Cassandra database has been deployed in a cluster composed of 6 nodes, using a virtualized environment with Docker. Each node has 32 GB of RAM and runs Cassandra 4.0.4.

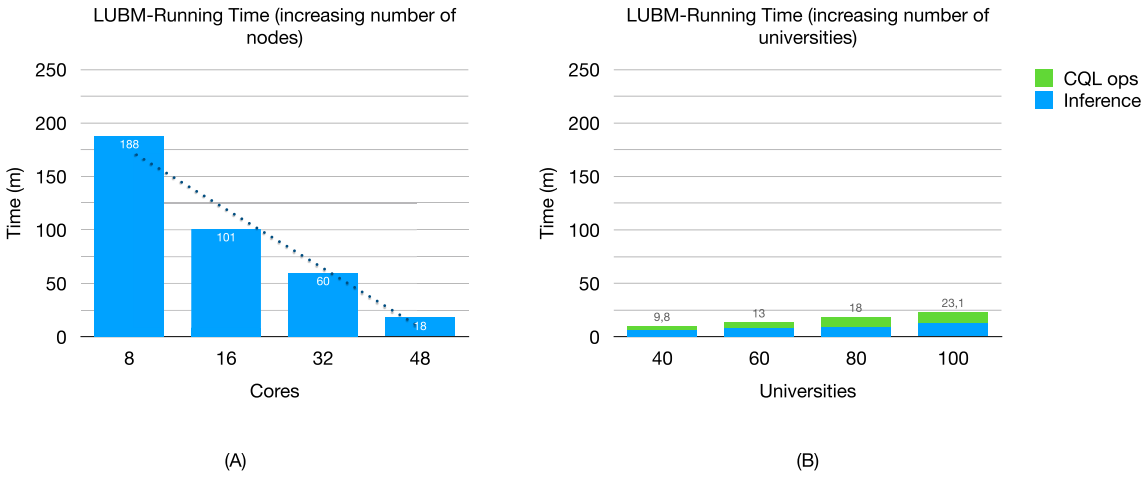

FIGURE 7 Running time (in minutes) spent in reasoning with different configurations.

5.1 | The univ-bench benchmark

This first use case has been developed to show the scalability of the reasoning process using NORA. The Lehigh University Benchmark (LUBM) is a benchmark ontology describing an academic domain (universities, departments, and activities) that facilitates the evaluation of reasoning algorithms. A companion tool generates synthetic OWL data with different ABox sizes. Using this tool, we generated 80 universities, yielding approximately 8 million triples, to evaluate NORA's performance and efficiency in reasoning, using a range of different cluster configurations. NORA was able to generate over 4 million new *ClassIndividuals* and more than 1.5 million *PropIndividuals* as a result of the reasoning process.

Figure 7A depicts the time used by NORA for reasoning materialisation with caching within the context of the LUBM ontology. Our initial findings suggest that the running time obtained decreases along with the number of cores used. We find this a compelling indication of NORA's capacity to handle increasing workloads efficiently. It is important to note that our efforts in optimising the system's code are ongoing, and hence, we anticipate achieving even more impressive results in subsequent evaluations.

In Figure 7B, we have taken a detailed look at the time taken for inference and the time NORA requires to carry out CQL operations (database interactions) with an increasing number of universities, while maintaining a fixed cluster configuration of 48 cores. As the volume of data, in this case represented by an increasing number of universities, expands, NORA demonstrates a robust ability to effectively manage and process this data without a significant increase in execution time. To maintain system stability and prevent potential issues arising from overload, we deliberately limit the number of concurrent write operations directed towards the database. A careful tuning of the database can streamline these interactions, thereby reducing the execution time. Furthermore, we anticipate that the removal of the limit on concurrent write operations would facilitate more rapid data processing, albeit potentially necessitating higher hardware specifications.

5.2 | The university ontology benchmark

The second use case has been performed to show how NORA can perform the most common reasoning tasks. The University Ontology Benchmark (UOBM) is a benchmark ontology derived from LUBM that comes in two versions: a more expressive OWL DL (UOBM DL) and a less expressive OWL Lite (UOBM Lite) version. We used the latter, covering all language constructs of OWL Lite, for evaluation. Randomly generated test data with pre-computed correct query results were used to evaluate the completeness and soundness of the inference. The queries are tailored for this experiment and respond to different purposes. For example, Q_1 involves a simple conjunction (all undergraduate students who take a specific course), whereas Q_6 involves an inverse property (see below). The final data set contains over 210.000 statements broken up into 20 files, each representing a department in the university:

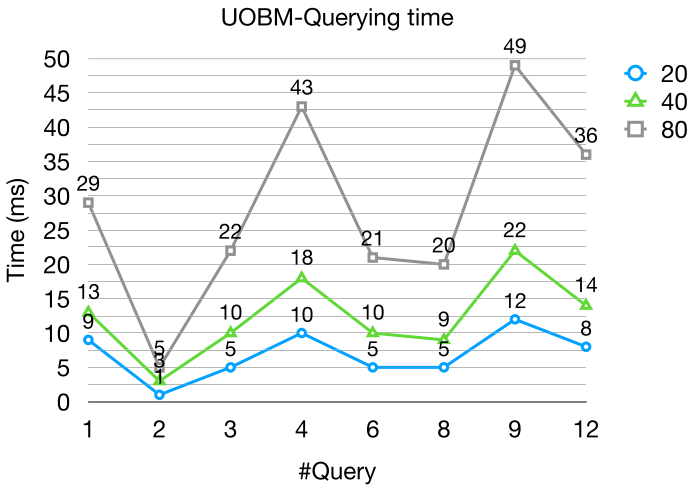

FIGURE 8 Querying time (in milliseconds) spent in answering queries using 20, 40 and 80 universities.

***Q*$_1$** Find all undergraduate students who take course http://www.Department0.University0.edu/Course0
***Q*$_2$** Find out all employees
Domain(worksFor,Employee), <a worksFor b> → <a rdf:type Employee>
Domain(worksFor,Employee), researchAssistant ⊑ ∃worksFor.ResearchGroup → researchAssistant ⊑ Employee
***Q*$_3$** Find out all students of http://www.Department0.University0.edu
Range(takeCourse,Student), GraduateStudent ⊑ ⩾1 takeCourse → GraduteStudent ⊑ Student
***Q*$_4$** All the publications by faculty of http://www.Department0.University0.edu
SubClass: Faculty = FullProfessor ⊔ AssociateProfessor ⊔ … ⊔ ClericStaff, Publication = Article ⊔ … ⊔ Journal
***Q*$_6$** All alumni of http://www.University0.edu
Inverse(hasAlumni,hasDegreeFrom), <a hasDegreeFrom b> → <b hasAlumnus a>
***Q*$_{12}$** All students who take course taught by http://www.Department0.University0.edu/FullProfessor0
GraduateStudent ≡ ∀takesCourse.GraduateCourse ⊓ ⩾1.takesCourse, Domain(takesCourse,Student) → Student ⊒ GraduteStudent

Experimental results show that our proposal is sound, that is, the precision is 1. Furthermore, we could also answer all queries correctly. Figure 8 shows the execution time for running the queries in a single node using different numbers of universities. The data points in the graph showcases the trend in execution time as the volume of the data increases.

6 | CONCLUSIONS AND FUTURE WORK

This article presents NORA, a scalable OWL reasoner backed up by Apache Cassandra. NORA follows a materialisation approach to perform reasoning. Thus, the reasoning engine will generate new data (*inferences*) and put them back into the Knowledge Base (the Apache Cassandra database). The inference rules implementation follows a fix point approach using Apache Spark. Thus, the execution of each reasoning block will generate new knowledge injected into the Knowledge Base from the Spark nodes. The reasoning process is evaluated iteratively until no new inferred knowledge is derived.

Using NoSQL as the storage layer is a conscious decision to provide scalability and reliability to the system. This design decision is not without concessions, as Cassandra offers no built-in join mechanism natively found in other SQL systems. This lack of relational features are supplemented by Apache Spark, albeit at the cost of higher computational time. We have tested the proposal with two well-known approaches: the Lehigh University Benchmark (LUBM) and the University Ontology Benchmark (UOBM). The initial experimentation shows promising results in terms of scalability and execution time (using LUBM) and completeness (using UOBM). This proposal has been implemented and publicly made available, so it is open to improvement by the community.

Firstly, we plan to compare different NoSQL/SQL storage layers, allowing users to choose the most suitable option for materialization during the reasoning process. This includes examining drop-in replacements for Cassandra, such as ScyllaDB.[††] Additionally, we intend to explore the performance of SQL systems using join operators, in order to determine the efficiency and effectiveness of such approaches in the context of our reasoner.

Furthermore, we will continue to develop new rule sets to support semantically richer fragments of OWL, including OWL DL. This encompasses remodeling existing rule sets to enhance the system's overall performance, identify bottlenecks, and optimize the reasoning process. Lastly, we are currently investigating an approach to cache RDDs, enabling the reuse of intermediate results in subsequent stages of the reasoning process, which could lead to improved efficiency and faster response times.

AUTHOR CONTRIBUTIONS

Antonio Benítez-Hidalgo conducted the implementation of the system and writing of the manuscript. Ismael Navas-Delgado supervised the design of the proposal and collaborated and reviewed the manuscript. María del Mar Roldán-García led and supervised the design of the proposal, and collaborated and reviewed the manuscript.

ACKNOWLEDGEMENTS

Our acknowledgements to the principal investigator of the research group José F. Aldana-Montes.

FUNDING INFORMATION

This work has been partially funded by grant (funded by MCIN/AEI/10.13039/501100011033/) PID2020-112540RB-C41, AETHER-UMA (A smart data holistic approach for context-aware data analytics: semantics and context exploitation). Antonio Benítez-Hidalgo is supported by Grant PRE2018-084280 (Spanish Ministry of Science, Innovation and Universities). Funding for open access charge: Universidad de Málaga / CBUA.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Antonio Benítez-Hidalgo https://orcid.org/0000-0002-4396-8359

REFERENCES

1. Berners-Lee T, Hendler J, Lassila O. *The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities*. The Scientific American. ScientificAmerican.com 2001.
2. Harris S, Seaborne A. *SPARQL 1.1 Query Language*. W3C Recommendation. W3C; 2013 https://www.w3.org/TR/2013/REC-sparql11-query-20130321/
3. Hitzler P, Krötzsch M, Parsia B, Patel-Schneider P, Rudolph S. *OWL 2 Web Ontology Language Primer*. 2nd ed. W3C; 2012.
4. Gruber TR. Toward principles for the design of ontologies used for knowledge sharing. *Int J Human Comput Stud*. 1995;43(5):907-928. doi:10.1006/ijhc.1995.1081
5. Krishnan A. *Making search easier: How Amazon's Product Graph is helping customers find products more easily*. Amazon Blog; 2018.
6. He Q, Chen BC, Agarwal D. *Building The LinkedIn Knowledge Graph*. LinkedIn Blog; 2016.
7. Bellomarini L, Fakhoury D, Gottlob G, Sallinger E. Knowledge graphs and enterprise AI: The promise of an enabling technology. *IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE; 2019:26-37.
8. Lakshman A, Malik P. Cassandra: A decentralized structured storage system. *SIGOPS Oper Syst Rev*. 2010;44(2):35-40. doi:10.1145/1773912.1773922
9. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. *HotCloud'10*. USENIX Association; 2010:10.

[††] https://www.scylladb.com.

10. Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI'12*. USENIX Association; 2012:2.
11. Sirin E, Parsia B, Grau BC, Kalyanpur A, Katz Y. Pellet: A practical OWL-DL reasoner. *J Web Semant*. 2007;5(2):51-53. Software Engineering and the Semantic Web. doi:10.1016/j.websem.2007.03.004
12. Shearer R, Motik B, Horrocks I. *HermiT: A Highly-Efficient OWL Reasoner*. OWLED; 2008.
13. Haarslev V, Hidde K, Möller R, Wessel M. The RacerPro knowledge representation and reasoning system. *Semantic Web*. 2012;3:3. doi:10.3233/SW-2011-0032
14. Abburu S. Article: A Survey on Ontology Reasoners and Comparison. *Int J Comput Appl*. 2012;57(17):33-39.
15. Matentzoglu N, Leo J, Hudhra V, Sattler U, Parsia B. A survey of current, stand-alone OWL reasoners. *Informal Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015) co-located with the 28th International Workshop on Description Logics (DL 2015), Athens, Greece,* 2015;6:68-79.
16. Mar Roldan-Garcia dM, Aldana-Montes J. A survey on disk oriented querying and reasoning on the semantic web. *22nd International Conference on Data Engineering Workshops (ICDEW'06)*. IEEE; 2006:58.
17. Roldan-Garcia MM, Aldana-Montes JF. DBOWL: Towards a scalable and persistent OWL reasoner. *2008 Third International Conference on Internet and Web Applications and Services*. IEEE; 2008:174-179.
18. Khadilkar V, Kantarcioglu M, Thuraisingham B, Castagna P. Jena-HBase: A distributed, scalable and effcient RDF triple store. *International Semantic Web Conference*. International Semantic Web Conference, Vol 2012; 2012.
19. Papailiou N, Konstantinou I, Tsoumakos D, Koziris N. *H2RDF: Adaptive Query Processing on RDF Data in the Cloud*. Association for Computing Machinery; 2012. doi:10.1145/2187980.2188058
20. Ladwig G, Harth A. CumulusRDF: linked data management on nested key-value stores. *Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10th International Semantic Web Conference (ISWC2011)*. 2011.
21. Mutharaju R. *Very large scale OWL reasoning through distributed computation*. Springer; 2012:407-414.
22. Wu K, Haarslev V. A parallel reasoner for the description logic ALC. *Description Logics*. 2012.
23. Antoniou G, Batsakis S, Mutharaju R, et al. A survey of large-scale reasoning on the Web of data. *Knowl Eng Rev*. 2018;33:e21. doi:10.1017/S0269888918000255
24. Maier F, Mutharaju R, Hitzler P. *Distributed Reasoning with EL++ Using MapReduce*. Wright State University; 2010.
25. Urbani J, Kotoulas S, Oren E, Harmelen vF. Scalable distributed reasoning using MapReduce. *The Semantic Web—ISWC 2009*. Springer Berlin Heidelberg; 2009:634-649.
26. Gu R, Wang S, Wang F, Yuan C, Huang Y. Cichlid: Efficient Large Scale RDFS/OWL Reasoning with Spark. *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE; 2015:700-709.
27. Liu Y, McBrien P. SPOWL: spark-based OWL 2 reasoning materialisation. *BeyondMR'17*. Association for Computing Machinery; 2017.
28. Mohamed H, Fathalla S, Lehmann J, Jabeen H. A scalable approach for distributed reasoning over large-scale OWL datasets. *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management—KEOD*. INSTICC. SciTePress; 2021:51-60.
29. Curé O, Kerdjoudj F, Faye D, Le Duc C, Lamolle M. On the potential integration of an ontology-based data access approach in NoSQL stores. *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*. IEEE; 2012:166-173.
30. Michel F, Faron Zucker C, Montagnat J. *Bridging the Semantic Web and NoSQL Worlds: Generic SPARQL Query Translation and Application to MongoDB*. Springer; 2019:125-165.
31. Reyes-Álvarez L, Roldan MM, Aldana MJ. Tool for materializing OWL ontologies in a column-oriented database. *Softw Pract Exp*. 2018;49. doi:10.1002/spe.2645
32. Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. *J Web Semant*. 2005;3(2):158-182. Selected Papers from the International Semantic Web Conference, 2004. doi:10.1016/j.websem.2005.06.005
33. Ma L, Yang Y, Qiu Z, Xie G, Pan P, Liu S. Towards a Complete OWL Ontology Benchmark. 2006;4011:125-139.

AUTHOR BIOGRAPHIES

Antonio Benítez-Hidalgo is a Ph.D. student at the University of Málaga. He graduated from the University of Málaga (2018), and received his MSc in Software Engineering and Artificial Intelligence (2019). His research lines includes Semantic Web Technologies, large-scale data processing and Big Data management within the research group KHAOS.

Ismael Navas-Delgado Computer Engineer (2002), PhD by the University of Málaga (2009) and Master in Cell Biology and Molecular Biology (2008). His research is developed within the ITIS Software participating in multiple

research projects (16) and technology transfer activities (5). His research activity focuses on the integration of data through the use of semantic technologies and their application to Life Sciences.

María del Mar Roldán-García is Assistant Professor in the Computer Science Department of the University of Málaga. Her research interests include scalable linked data reasoning, the development of domain ontologies and semantic web-based applications, and the use of domain knowledge within big data analytics algorithms.

How to cite this article: Benítez-Hidalgo A, Navas-Delgado I, Roldán-García MM. NORA: Scalable OWL reasoner based on NoSQL databases and Apache Spark. *Softw Pract Exper*. 2023;1-16. doi: 10.1002/spe.3258