UNIVERSITY OF LJUBLJANA FACULTY OF COMPUTER AND INFORMATION SCIENCE

Vanč Levstik

Expressiveness and Scalability of Semantic Web Systems

GRADUATE THESIS

UNIVERSITY STUDY PROGRAMME OF COMPUTER AND INFORMATION SCIENCE

MENTOR: doc. dr. Dejan Lavbič

Ljubljana 2016

Vanč Levstik

Izraznost in skalabilnost sistemov Semantičnega spleta

Diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2016

Created and edited with LaTeX.

The Faculty of Computer and Information Science issues the following thesis:

The availability of Semantic Web technologies is not an issue in academic environment, where they are mainly employed and researched. But there is a problem in knowledge transfer and Semantic Web technologies employment in industry environment. In the following thesis student should review the current state of the art of systems supporting the development of semantically enriched applications and evaluate systems in the context of suitability for use in the real industry problems. The focus should be mainly on the relationship between the expressiveness and the scalability of these systems. The analysis should be confirmed with a case study. The most important representatives of the individual category of semantic systems should be included in the evaluation process. Finally a critical assessment and proposal for solution of Semantic Web system's employment for solving real industry problems is expected.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo::

Tehnologije Semantičnega spleta so že nekaj časa na voljo in v uporabi predvsem v akademskem okolju, medtem ko je prenos omenjenih tehnologij v prakso omejen. V okviru diplomske naloge preglejte trenutno stanje sistemov za podporo razvoju semantično bogatih aplikacij in jih ovrednotite v kontekstu primernosti uporabe za realne probleme. Osredotočite se predvsem na razmerje med izraznostjo in skalabilnostjo omenjenih sistemov ter svojo analizo potrdite na študiji primerov. V evaluacijo vključite najbolj vidne predstavnike semantičnih sistemov iz posameznih kategorij pristopov in na koncu podajte kritično oceno in predlog za uporabo semantičnih sistemov na realnih problemih v gospodarstvu.

Declaration of Thesis Authorship

Undersigned Vanč Levstik, with ID number **63080090**, author of graduate thesis with title:

Expressiveness and Scalability of Semantic Web Systems

I confirm that:

- I worked on the thesis myself under the mentorship of doc. dr. Dejan Lavbič,

- electronic issues of thesis, title (slov., eng.), abstract (slov., eng.) and keywords (slov., eng.) same as in printed version of thesis

- I agree with publishing electronic version of thesis under ”Dela FRI” university archive.

Ljubljana, 6. March 2016   Signed:

Contents

Acronym	Full meaning
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Web Ontology Language
SPARQL	SPARQL Protocol and RDF Query Language
URI	Universal Resource Identifier
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
SQL	Structured Query Language
DSL	Domain Specific Language
CPU	Central Processing Unit
ARQ	Automatic Repeat Query
NoSQL	Non Structured Query Language
XML	EXtensible Markup Language
TURTLE	Terse RDF Triple Language
W3C	World Wide Web Consortium

Abstract

In this thesis we focus on Semantic Web systems, their expressiveness and scalability. We go through technologies and standards of Semantic Web and compare their expressiveness. Our goal is to research and compare Semantic Web systems and define which are suitable for solving real industry problems. After defining several industry related use cases, we perform a comprehensive analysis and evaluation of selected systems on given problems. Results show that Semantic Web systems are a good option for complicated problems needing high expressiveness. Scaling shows to be a problem, so we propose hybrid solutions as the best choice for industry problems.

Keywords: Semantic Web, scalability, expressiveness, Semantic Web Systems, querying

Povzetek

V tem delu se posvetimo sistemom Semantičnega spleta, predvsem njihovi izraznosti in skalabilnosti. Pregledamo različne tehnologije in standarde Semantičnega spleta ter primerjamo njihovo izraznost. Cilj diplomskega dela je raziskati sisteme Semantičnega spleta in ugotoviti, kateri med njimi so primerni za uporabo v problemih industrije. Na koncu se posvetimo analizi nekaj izbranih sistemov in pregledamo kako se obnašajo pri reševanju teh problemov. Rezultati pokažejo, da so sistemi Semantičnega spleta dobra rešitev za zahtevne probleme ki potrebujejo visoko izraznost. Sistemi imajo probleme pri skaliranju, zato za uporabo v industriji predlagamo uporabo hibrida različnih sistemov.

Ključne besede: Semantični splet, skalabilnost, izraznost, sistemi Semantičnega spleta, poizvedbe

Razširjeni povzetek

Uvod

Semantični splet se je pojavil kot koncept leta 2001 [3]. Splet je sestavljen iz dokumentov, ki jih razumemo ljudje. Ideja semantičnega spleta pa je narediti splet razumljiv tudi računalnikom. Semantični splet se ni popolnoma uveljavil in je s strani nekaterih ocenjen kot neuspeh [44]. Semantični splet je sestavljen iz različnih tehnologij in standardov. Tehnologije in sistemi semantičnega spleta se lahko uporabljajo tudi v sistemih, ki ne temeljijo na njem. Primer sistema v industriji, ki uporablja gradnike semantičnega spleta, je recimo Googlov graf znanja.

V diplomskem delu raziščemo sisteme semantičnega spleta, ugotovimo kakšno dodatno izraznost nam prinesejo in kako ta izraznost vpliva na skalabilnost sistema. Naš cilj je raziskati ali so ti sistemi pripravljeni za uporabo v industriji in kateri med njimi so najbolj primerni za uporabo. Pričakujemo, da nam višja izraznost prinese nižjo skalabilnost, zato raziščemo, kateri sistemi nam prinesejo najboljše razmerje med izraznostjo in skalabilnostjo.

Tehnologije semantičnega spleta

V drugem poglavju raziščemo tehnologije Semantičnega spleta. Predstavimo gradnike semantičnega spleta, pokažemo kako izgledajo podatki. Posvetimo se predvsem vprašanju, kakšno izraznost nam prinesejo različne tehnologije, kot so RDF format za zapis trojčkov [23], RDFS za opis grafa sestavljenega iz trojčkov [42] in OWL za opis ontologij [22]. Ugotovimo, kako nam različni načini sklepanja s

pomočjo OWL profilov pomagajo pri večji izraznosti. Predstavimo tudi različne načine poizvedb podatkov, kjer je glavni način za poizvedbe v semantičnem spletu preko povpraševalnega jezika SPARQL [45]. Opišemo tudi jezike poizvedb Gremlin [19], Elasticsearch DSL [12] ter SQL [10].

Analiza sistemov

V tretjem poglavju se najprej lotimo izbire sistemov za podrobno analizo. Izberemo po en sistem iz različnih skupin podatkovnih baz. Izberemo sistem Jena, [15] kot primer odprtokodnega sistema ki ga trenutno uporabljajo številni razvijalci in je podprt z mnogimi orodji. GraphDB [37] kot primer plačljivega sistema, ki nam obljublja visoko izraznost. RDFox [39] je izbran kot sistem s poudarkom na visoki skalabilnosti, kar je dokazal v obstoječih meritvah sistemov [36]. Cayley [8] izberemo kot primer podatkovne baze grafov, ki ponuja visoko skalabilnost. Kot primer klasične relacijske podatkovne baze izberemo PostgreSQL [20], predvsem zaradi dobrih rezultatov v primerjavi z ostalimi relacijskimi podatkovnimi bazami [47]. Elasticsearch [12] izberemo kot primer podatkovne baze dokumentov. Gre za odprtokodno rešitev, ki je pogosto v uporabi v industriji in ima močno podporo razvijalcev.

Sisteme smo bolj natančno analizirali in pregledali obstoječe raziskave in meritve. Poleg izraznosti in skalabilnosti, smo pogledali tudi kako zahtevni so za uporabo, ali so plačljivi, katera orodja so na razpolago za delo z njimi. Rezultati so vidni na sliki 1, kjer je višja vrednost boljša v vseh dimenzijah.

Študija primera

V četrtem poglavju se glede na rezultate iz tretjega poglavja odločimo za podrobnejšo analizo treh sistemov. Izberemo sisteme RDFox, Cayley in Elasticsearch.

Problem definiramo kot postavitev sistema za odgovarjanje na geografska vprašanja, kjer je naša domena celoten geografski svet.

Podatki, ki jih uporabimo v študiji primera, so pridobljeni iz podatkovnega direktorija Geonames [51]. Gre za podatke o geografskih lokacijah, kot so države,
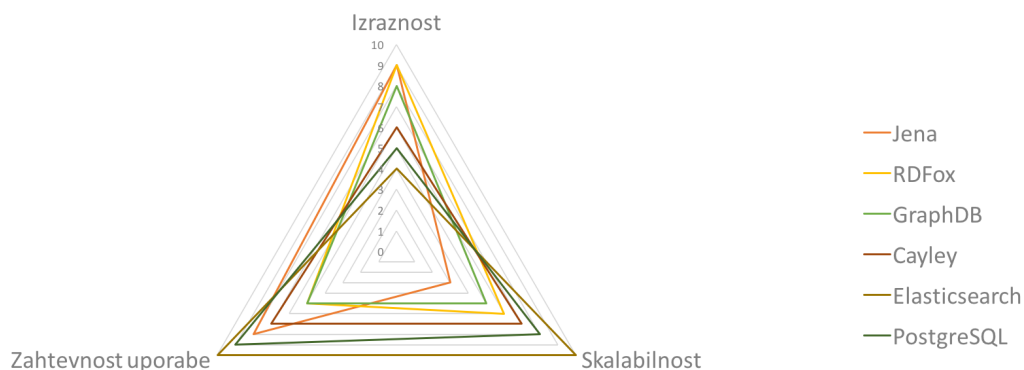
Slika 1: Primerjava sistemov

mesta, gore, jezera... Celotni podatki obsegajo več kot 150 milijonov RDF trojčkov.

Pokažemo, kako smo postavili delovno okolje za te sisteme in kako pripravimo sisteme, ko povečamo količino podatkov ter s tem potrebujemo večjo skalabilnost. Opišemo tudi arhitekturo vsakega sistema in jih vizualiziramo.

Izberemo vprašanja, na katera želimo odgovoriti s pomočjo naših podatkov:

- želimo dobiti seznam vseh mest,

- želimo seznam vseh mest iz izbrane države,

- želimo seznam vseh geografskih lokacij iz izbrane občine,

- želimo seznam vseh geografskih lokacij v bližini izbranega mesta,

- želimo seznam vseh mest ki so X km oddaljena od naše lokacije.

Na vsa vprašanja poskusimo odgovoriti s pomočjo poizvedb v vseh 3 sistemih, ki jih testiramo v naši študiji primera. Na vprašanje najprej odgovorimo na manjši količini podatkov in nato še na večji. Pogledamo, ali je bil sistem zmožen odgovoriti na problem, v kakšnem času mu je to uspelo na manjši in večji količini podatkov. Vse poizvedbe tudi pokažemo in jih opišemo.

Primerjamo vse rezultate in jih tudi združimo v odločitveni model. Ta nam pomaga k odločitvi, kateri sistem je najbolj primeren za uporabo v industriji. Skalabilnost in izraznost imata višjo utež v našem odločitvenem modelu. Pogledamo tudi kako zahtevna je postavitev sistema, koliko časa potrebujemo, da v sistem naložimo vse podatke in kako zahtevna je uporaba sistema ter pisanje poizvedb za razvijalce. Elasticsearch, se izkaže kot najboljša rešitev naše študije primera. Rezultati odločitvenega modela, ki smo ga uporabili, so vidni v tabeli 1.

Tabela 1: Rezultati študije primera

Dimenzije odločanja	Uteži	RDFox	Cayley	Elasticsearch
Dimenzija		1	2	3
Izraznost	5.0	10	8	7
Skalabilnost	5.0	3	2	10
Čas nalaganja podatkov	1.0	10	1	3
Zahtevnost postavitve sistema	1.0	4	10	8
Zahtevnost pisanja poizvedb	1.0	7	6	10
Rezultati		86.0	67.0	106.0

Zaključek

V zaključku potrdimo našo hipotezo, da višja izraznost prinaša tudi nižjo skalabilnost. Ugotovimo, da so sistemi Semantičnega zanimiva rešitev za probleme, ki to visoko izraznost potrebujejo, medtem ko skalabilnost še vedno ni enakovredna drugim sistemom z nižjo izraznostjo.

Kot zanimivo idejo za nadaljnje delo omenimo možnost implementacije hibridnega sistema, ki bi združeval različne pod-sisteme z različno izraznostjo. Tako bi

lahko probleme, ki ne potrebujejo polne izraznosti, dodelili skalabilnim sistemom z nizko izraznostjo, kot je recimo Elasticsearch. Tiste s potrebo po visoki izraznosti pa bi dodelili sistemu kot je RDFox. Sistem bi tako za uporabnika ali razvijalca deloval kot en sam sistem, čeprav bi uporabljal različne podsisteme.

Chapter 1

Introduction

Semantic Web has first appeared as a term in 2001 [3]. World Wide Web consists of documents that are mostly only readable by humans. Idea of Semantic Web is to extend them and make them readable and easy to understand by computers as well. Through Semantic Web technologies and solutions, we are able to add semantic meaning to objects of the web and make meaningful connections between those objects. "Web 3.0" has often been another term for Semantic Web and possibilities it would bring.

Semantic Web has standardised many technologies and standards, main ones being Universal Resource Identifier (URI), Resource Description Framework (RDF), and Web Ontology Language (OWL). These technologies help us express semantics of resources on the web and connections between them. Transition of web into Semantic Web has never really happened and all concepts are not fully realised. There are notions that Semantic Web as a concept has failed [44].

Although technologies are not fully embraced by World Wide Web there are many applications of it, that can be seen on the web and further. Large search engines, such as Google, are extensively using it to enrich their results and make meaningful semantic decisions about documents on the web. Term knowledge graph is used to describe knowledge systems that are often using Semantic Web technologies underneath. Ontologies and ontology languages are used by scientists, e.g. at European Organization for Nuclear Research (CERN).

Semantic Web technologies are still not embraced by industry as a whole as a solution to their problems. Scalability is often singled out as an unknown with these systems and technologies. Industry is still avoiding Semantic Web systems because scaling of them is not researched in full. High expressiveness often comes with lower scalability and it can be hard to decide what level of system's expressiveness is needed for different industry problems.

We want to research these technologies from an industry applications point of view and investigate what problems we can solve with them. Thesis will focus on Semantic Web systems, their expressiveness and how different levels of expressiveness affect scalability. Our goal is to research current state of the art solutions and what levels of expressiveness can they offer us. We will leverage that against scalability with help of existing research and benchmarks. We will focus on a few selected solutions and test their expressiveness and scalability. We will try to define what level of expressiveness is needed for use in industry level problems and what solutions are best fitted for these use cases.

Case study of selected systems on a real data and problems will enable us to see, how these systems behave. We will try to decide if Semantic Web technologies in current state can bring value to industry applications. Which ones have shown to be best fitted for our selected use case.

In Chapter 2 we will look into layers of Semantic Web technologies from data modeling and query answering point of view. Chapter 3 will focus on Semantic Web systems, existing research and comparison of these systems. In Chapter 4 we will evaluate selected systems, their expressiveness and scalability through case study and present our results with the help of a decision model. We will conclude with Chapter 5 in which we will present our conclusions and discuss challenges and future work.

Chapter 2

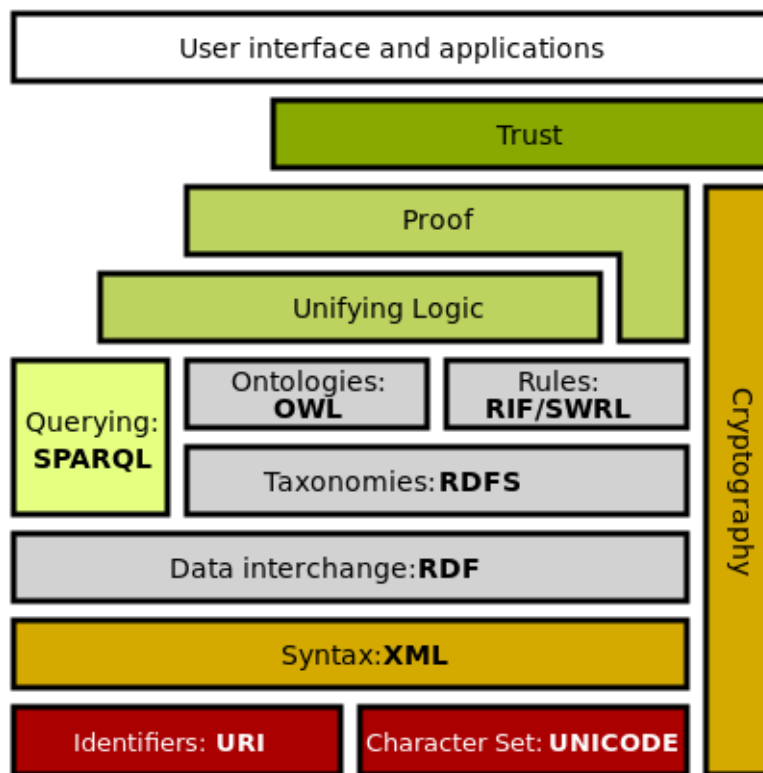# Semantic Web Technologies

Figure 2.1: Semantic Web Stack

In this chapter we will go through layers of Semantic Web depicted in Figure 2.1. Each layer brings some additional expressiveness, but as a consequence can

bring additional scalability problems.

2.1   Data Modeling

2.1.1   RDF

The Resource Description Framework (RDF) is a framework for expressing information about resources [23].

RDF uses triples that allow us to make statements about resources. Each resource in RDF graph is an IRI (Internationalized Resource Identifier), which is an unique Unicode string that defines a resource [29]. IRIs are a generalization of URIs (Uniform Resource Identifier) that are often used in the same context. These statements (or triples) are formatted as:

$$< subject >< predicate >< object >$$

A statement is a relationship between two resources. Subject is related to object with a relationship that is called property. All three parts construct a single RDF triple.

RDF can be serialized in many different formats, main ones being:

- Turtle and TriG

- JSON-LD (JSON based)

- RDFa (for HTML embedding)

- N-Triples and N-Quads (human readable line-based exchange formats)

- RDF/XML

We will not focus on them specifically, but all of them are equal in terms of expressiveness.

An example of an RDF triple which tells us that University of Ljubljana is of "type" Universities in Slovenia [1]:

---

[1]Additional prefixes in query examples are left out for better readability

```
University_of_Ljubljana  rdf:type  UniversitiesInSlovenia  .
```

2.1.2 RDFS

RDF Schema provides a data-modelling vocabulary for RDF [7]. RDF Schema semanticaly extends RDF. It allows us to describe groups of related resources and relationships between them. Its class and property system are in many cases similar to object-oriented programming principles [42]. There are cases where RDF on its own cannot model our relations and RDFS can do it using constructs such as:

- $rdfs : subClassOf$

- $rdfs : range$

- $rdfs : domain$

Main functionalities that RDF Schema adds are subclass hierarchy, property hierarchy and domain and range definitions of these properties [1]. That can still be quite limited for many use cases.

An example of RDFS triple that extends on previous RDF triple:

```
UniversityOfLjubljana  rdf:type  UniversitiesInSlovenia  .
UniversitiesinSlovenia  rdfs:subClassOf   University  .
```

We can observe that we get the information about hierarchy, from which we can get the information that University of Ljubljana is also a University, as all Universities in Slovenia are subclasses of University.

2.1.3 OWL

To increase expressiveness OWL Web Ontology language (OWL) has been defined by W3C. OWL uses RDF Schema and extends it with additional features that can add expressiveness. Some of extensions that OWL adds are:

- Local scope of properties

- Disjointment of classes

- Boolean combinations of classes

- Cardinality restrictions

- Special characteristics of properties

There are different OWL 2 profiles as defined by W3C OWL Working Group [22]. We will focus on OWL 2 profiles in this thesis, shortly mentioning OWL 1 profiles as well.

An example of our data extended with OWL triple which adds the information that resource URI UniversityOfLjubljana talks about the same resource as another resource URI for University of Ljubljana in Wikidata:

```
UniversityOfLjubljana rdf:type UniversitiesInSlovenia .
UniversitiesinSlovenia rdfs:subClassOf University .
UniversityOfLjubljana owl:sameAs wikidata:Q1377 .
```

That allows us to get additional information about same resource from different URI, therefore giving us more information and increased expressiveness.

OWL Full

Entire OWL language is called OWL full and uses all of the OWL language primitives. It is fully upward compatible with RDF and RDFS, syntactically and semantically [1]. Expressive power of it is so large that it is virtually impossible to achieve complete and efficient use of reasoning.

OWL 2 EL

The OWL 2 EL profile is a profile suitable for applications using ontologies with large number of properties and classes. It can capture expressive power of such on-

tologies and provide consistency, class expression subsumption and instance checking in polynomial time [34].

OWL 2 QL

The OWL 2 QL profile is designed so that it allows complete query answering in LOGSPACE related to the size of data. It provides many main features to express conceptual models, like UML (Unified Modeling Language) and ER (Entity-Relationship) diagrams [34]. Efficient query rewriting can also be used to achieve using OWL 2 QL on relational (SQL) databases [41].

OWL 1 DL and OWL Lite

We mention OWL 1 DL profile as well which is also a subset of full OWL profile and can be counted as one of OWL 2 profiles. OWL 1 DL places a number of constraints on OWL language. It requires separation between classes, data types and other property types, individuals, data values and the built-in vocabulary [32]. That means that most RDF(S) vocabularies cannot be used with OWL 1 DL. OWL Lite has all restrictions of OWL 1 DL and also forbids usage of some additional constructs. Idea of OWL Lite is being the minimal possible subset that achieves useful expressiveness and giving high efficiency.

OWL 2 RL

The OWL 2 RL profiles can be used for applications that require highly scalable reasoning without losing too much expressiveness. It is somewhere between OWL 2 and RDFS at expressiveness level [34]. It is a profile that stands out as one that can be used in industry-level applications using reasoning [52]. There are also existing use cases of using it through relational database back-end and achieving high efficiency and scalability [30].

Relational and document store

We also want to explore using relational and document store data for storing semantic web data. Relational and document stores are scalable and well tested in real industry environments. There is no reasoning provided and loss of expressiveness, compared to using layers of Semantic Web standards, is expected.

2.2   Querying Data

2.2.1   SPARQL

SPARQL 1.1 (SPARQL Protocol and RDF Query Language) is a set of specifications that provide languages and protocols to query and manipulate RDF graph content on the Web or in an RDF store [21]. Latest version of SPARQL is 1.1 which has all the features of 1.0 version as well. It is most used querying language for Semantic Web data. SPARQL can be used over data stored in XML, JSON, CSV or TSV format and can cover different level of expressiveness mentioned in previous section. There are also subsets of SPARQL which have lower expressiveness but better scalability, SPARQL-LD can be used for OWL-LD specific data [45].

Example SPARQL query that would return names of all universities in Slovenia:

```
SELECT ?name
WHERE {
  ?university rdf:type UniversitiesInSlovenia .
  ?university name ?name
}
```

2.2.2   Gremlin

Gremlin is a graph traversal machine and language designed, developed, and distributed by the Apache TinkerPop project [43]. Gremlin enables both declarative

and imperative querying. It promises high scalability and possibilities of graph traversal. Gremlin is not directly related to Semantic Web, but promises similar expressiveness using same data-sets.

Query example with same results as SPARQL example:

```
g.V()
.Has("rdf:featureClass","UniversitiesInSlovenia")
.Out("name").All()
```

2.2.3 Elasticsearch Query DSL

Query DSL is a query language used with Elasticsearch. Elasticsearch [12] is a search server operating on document data, that is installed using Apache Lucene as back-end [31]. It supports leaf queries, that can be used to look for particular value on particular field. More complex problems can be answered using compound queries which can combine different leaf queries to give us more expressive power.

Elasticsearch Query DSL does not operate on triples, but on document stored data only. Query with similar results to previous examples would be:

```
{
    "query" : {
        "match" : { "rdf:type" : "UniversitiesInSlovenia" }
    },
    "fields" : ["name"]
}
```

2.2.4 SQL

SQL is a query language designed for managing data held in a relational database [10]. SQL can also be used for querying RDF/OWL data with query rewriting [9]. SQL is a well known and widely used language. Similar to Elasticsearch Query

DSL, SQL is not a language written for Semantic Web. It can provide very robust and scalable queries that have less semantic expressiveness than SPARQL.

Similarly to Elasticsearch SQL does not operate on triples, SQL operates over relational data and query similar to previous ones would use two SQL tables and look like this:

```sql
SELECT universities.name
FROM universities
LEFT JOIN countries
ON universities.country_id=countries.id;
WHERE countries.name='Slovenia';
```

Chapter 3

Semantic Systems Analysis

In this chapter we will focus on comparing different systems for storing and querying semantic data. We will compare them on expressiveness they can provide, focusing on previous chapter. Scalability will be another dimension on which we will compare them, using existing benchmarks and other testing that has been done. We will add a third dimension, ease of use, that will focus on ease of development, access to source code, current tooling available and support provided. At the end of the chapter we will select a few of the most promising solutions and include them in our case study.

3.1 Systems Selection

In selecting the systems for our analysis we reviewed previous research work and also wanted to get a good split among different kinds of databases.

We wanted to select a widely used and well understood open source system with support for different Semantic layers. Choices that stand out are Jena and Virtuoso [48], we opted for **Jena** for the better modularity it provides as well as strong support behind it from Apache project [15].

Proprietary Semantic Web solutions was another group we wanted to look into. Allegrograph [25], Oracle [38] and GraphDB [37] were choices here. **GraphDB** was chosen for providing better expressiveness options than other two systems [33].

Semantic Web system with high focus on scalability was another type we wanted to explore. **RDFox** [39] was chosen for its good results in existing benchmarks [36].

Graph databases with no direct support for layers and technologies of Semantic Web were another group. Titan [2], Cayley [8] and Neo4j [50] were our main choices here. We opted for **Cayley** because of being open-source and used by Google knowledge graph which promises high scalability [24].

We wanted to get a relational database system for their wide usage among developers. **PostgreSQL** [20] was a clear choice here as it is open source, widely used, and showing good results in various scaling benchmarks [47].

Document store systems was another one we want to explore. **Elasticsearch** [12] being a choice here for its high scalability, ease of use and widespread use among developers [18].

3.2   Jena

Jena is a Java framework that can be used for building Semantic Web applications. There are different back-end solutions that can be used for storing data. Figure 3.1 shows that Jena supports different back-ends. Jena SDB [14] is a solution using relational databases as a back-end, but it is not being actively developed anymore. Jena TDB [15] is a native triple store that can be used as part of Jena framework or also exposed to external application using Fuseki component. JenaTDB promises to be better supported and has better focus on semantic expressiveness.

3.2.1   Expressiveness

Jena has support for RDFS and more memory intensive OWL reasoning. Pellet [46] with OWL DL reasoning can also be used for extending Jena functionalities. Querying is supported through ARQ framework that supports most of the SPARQL 1.1 features.
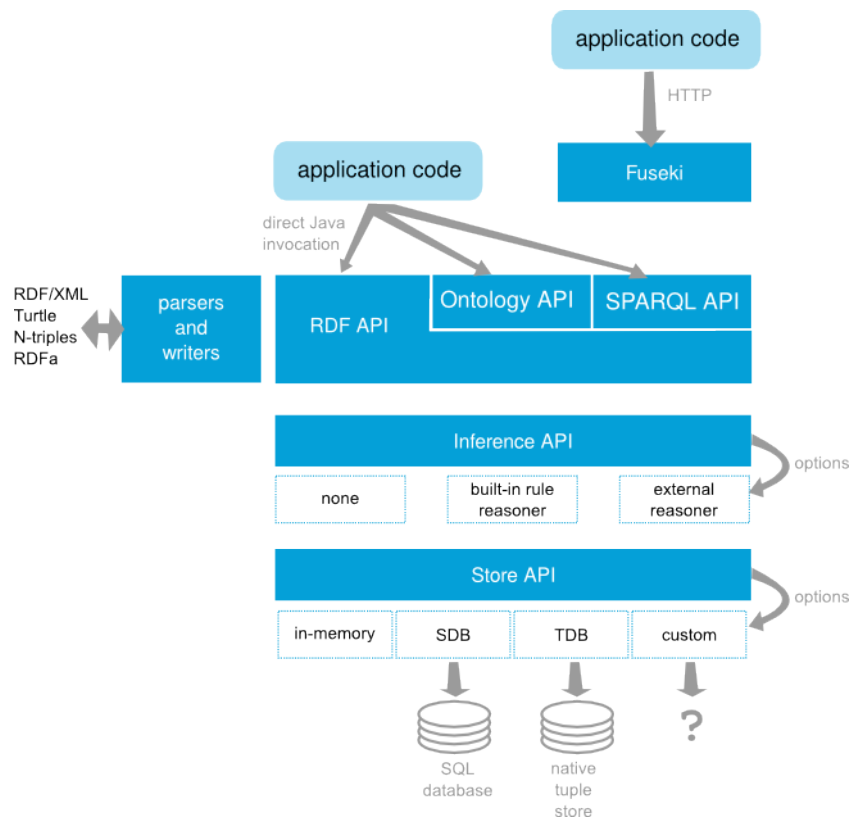
Figure 3.1: Jena Architecture

3.2.2 Scalability

JenaTDB (and JenaSDB) is not especially fast and scalable even when using only RDFS without any extensions with OWL [5]. There are possibilities for extending Jena using H-base as a back-end [27]. Benchmarks show that on higher scale of data being used, JenaTDB can produce memory problems which H-Base implementation can work with. There is also possibility of using OWL DL in this version, making it more scalable at high expressiveness.

3.2.3 Ease of Use

Jena is written in Java programming language that is widely used and adopted by developers. Project is supported by Apache Foundation and open sourced, giving it strong tooling and on-going support and trust.
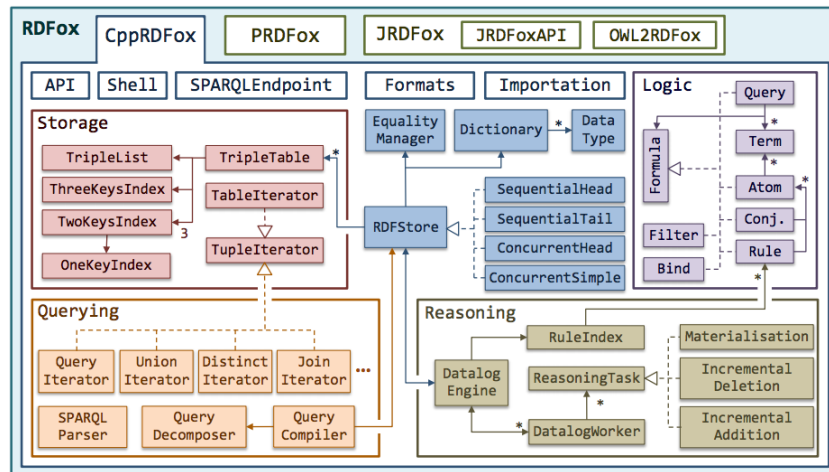
3.3   RDFox

Figure 3.2: RDFox Architecture

RDFox is a highly scalable in-memory RDF triple store that supports shared memory parallel datalog reasoning [39]. Figure 3.2 shows the engine part of RDFox that is written in C++ programming language. It shows how different parts for reasoning, querying, and storing data are structured.

3.3.1   Expressiveness

RDFox supports RDF reasoning and can also support OWL 2 RL profile reasoning which gives it more expressiveness, although not getting the full OWL 2 expressiveness [36]. OWL 2 RL should give us expressiveness that most of the queries need. Queries are done with SPARQL, there is support for most of the SPARQL 1.1 features as well.

3.3.2   Scalability

RDFox has shown high scalability capabilities, compared to DBRDF and OWLIM-Lite [40]. Handling and using parallelization and memory to full extent has shown to be used very well in RDFox [35].

3.3.3 Ease of Use

RDFox is developed and supported by University of Oxford. It is copyrighted under academic license and open sourced. Written in C++, but supported with bindings in different programming languages such as Java and Python. As system is quite new, tooling and developer familiarity is not as strong as at some other systems.
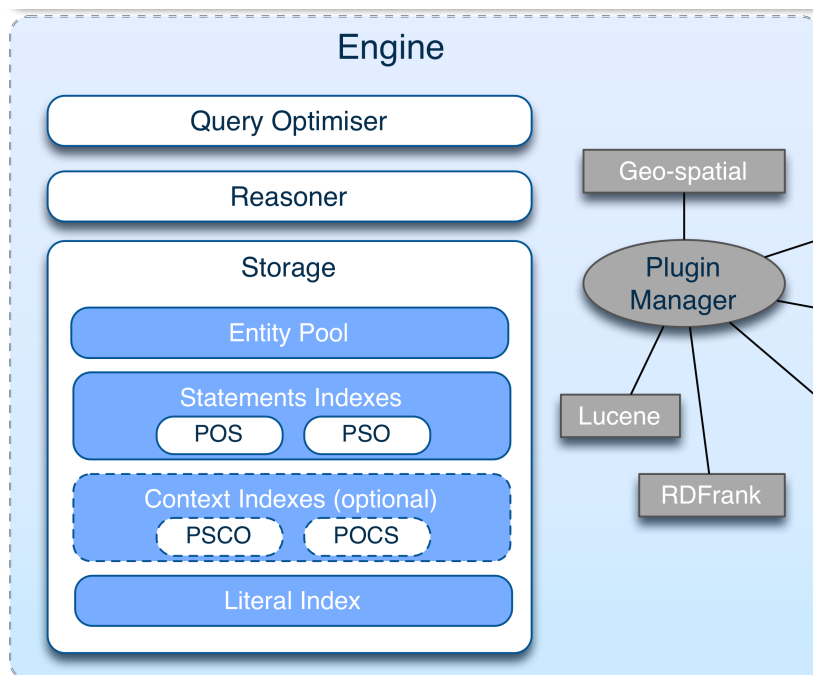
3.4 GraphDB

Figure 3.3: GraphDB Architecture

GraphDB or OWLIM as it used to be called, is described as a family of semantic repositories that provide storage, inference and novel data-access features delivered in a scalable, resilient, industrial-strength platform [28].

3.4.1   Expressiveness

GraphDB offers different levels of expressiveness. It can be used with RDFS only reasoning, OWL 2 RL or OWL 2 QL with increasing expressiveness but also decreasing performance [28].

3.4.2   Scalability

Different versions of OWLIM have shown to be one of the best choices for scalability among different triple store solutions [33].

3.4.3   Ease of Use

GraphDB is closed source and company supporting it offers different editions from free to enterprise version [37]. There are different tools available and semantic service as a suite is provided as well. Figure 3.3 shows engine architecture of GraphDB, we can observe architecture supports modularity with plugins as well.

3.5   Cayley

Cayley is an open-source graph inspired by the graph database supporting Google knowledge graph [8]. Its goal is to be something in the middle of classic triple store and scalable NoSQL graph databases. It is not directly connected to specific database back-end and allows for choosing different back-ends that supports different scalability and query speed needs.

3.5.1   Expressiveness

Cayley can load RDF valid data, but does not have support for any semantic reasoners as it is using its own query language that builds on Gremlin query language [19]. Gremlin operates on triples, but it does lose on some additional expressiveness that is provided by Semantic Web reasoning.

3.5.2 Scalability

Cayley promises high speed, as query language it uses has shown to be fast and scalable [24]. It can use different back-ends which allows best one to be chosen for scalability needs and allows further improvements to scalability later.

3.5.3 Ease Of Use

Cayley is fully open source and supported by Google. It is written in Go programming language and supports multiple databases as back-end. It is closely connected with Google knowledge graph, giving it strong support and promise.

3.6 Elasticsearch

Elasticsearch is widely used in different scalable web applications. It is not used as often in Semantic Web applications, but there are successful use-cases that are using it for that as well [49]. Elasticsearch is a search server that can be used on top of data. There are existing examples of Elasticsearch being used for indexing Semantic Web data [26].
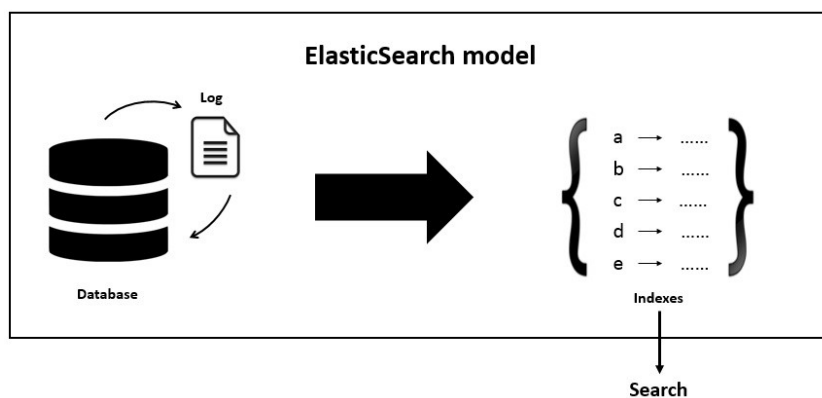
3.6.1 Expressiveness

Figure 3.4: Elasticsearch Architecture

Elasticsearch is based on documents and not directly tied to any back-end or reasoning expressiveness it can use. As seen in Figure 3.4 database is storing data, which is then indexed to Elasticsearch. It can be used with relational data or RDF/OWL triple data. As it is not directly connected and created for Semantic Web, there is a loss of expressiveness expected. It operates on JSON files and does not use Semantic Web query languages or reasoning.

3.6.2   Scalability

Elasticsearch is highly scalable and widely used in various applications [12]. As we lose some expressiveness there is a possibility of queries becoming more complicated than queries answering same problems in more expressive systems.

3.6.3   Ease of Use

Elasticsearch is fully open sourced and supported by Elastic. It is written in Java using Apache Lucene[13] for search. It is easy to set up. Elasticsearch is already widely used among developers and there is strong tooling and documentation behind it. Elasticsearch is available through Software As A Service platforms as well such as AWS Elasticsearch [1]

3.7   PostgreSQL

PostgreSQL is an open source client/server relational database. It compares well to other major commercial databases such as Oracle, Sybase, DB2 [11]. Figure 3.5 show us simplified architecture of the database, we can see that it operates on shared memory and fetches data from disk when needed.

3.7.1   Expressiveness

PostgreSQL as a relational database does not support any standards of Semantic Web technologies. RDF notation or any kind of OWL reasoning is not available.

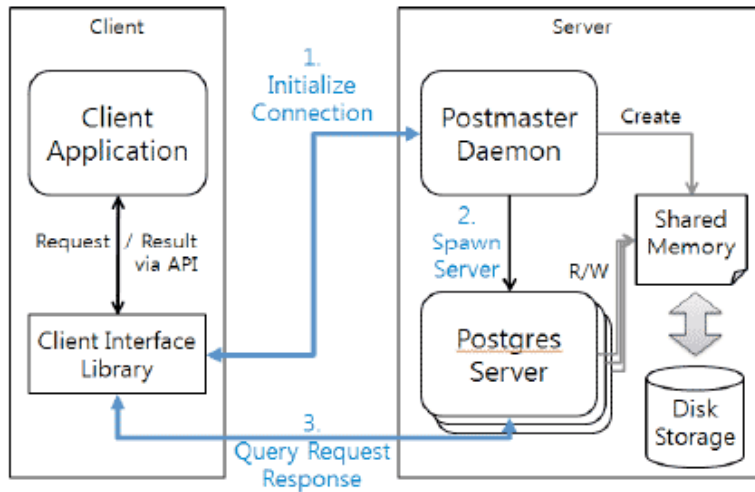[1]https://aws.amazon.com/elasticsearch-service/

Figure 3.5: PostgreSQL Architecture

SQL as a query language is available to us and can help us make relational queries. There is no concept of triples, so similarly to Elasticsearch we are losing expressiveness offered by SPARQL and reasoning.

3.7.2 Scalability

Many benchmarks and tests has shown PostgreSQL to be one of the fastest and most scalable relational databases [11]. With help of different specific indexes, query speed can be significantly scaled and increased as well.

3.7.3 Ease of Use

PostgreSQL is open source and supported by development group behind it. It is highly expandable and there are bindings available for many major programming languages. PostgreSQL is a SQL based query language, which is used widely by developers. There are many tools available for handling the database and wide group of users provide lots of existing knowledge about PostgreSQL uses.

3.8 Comparison
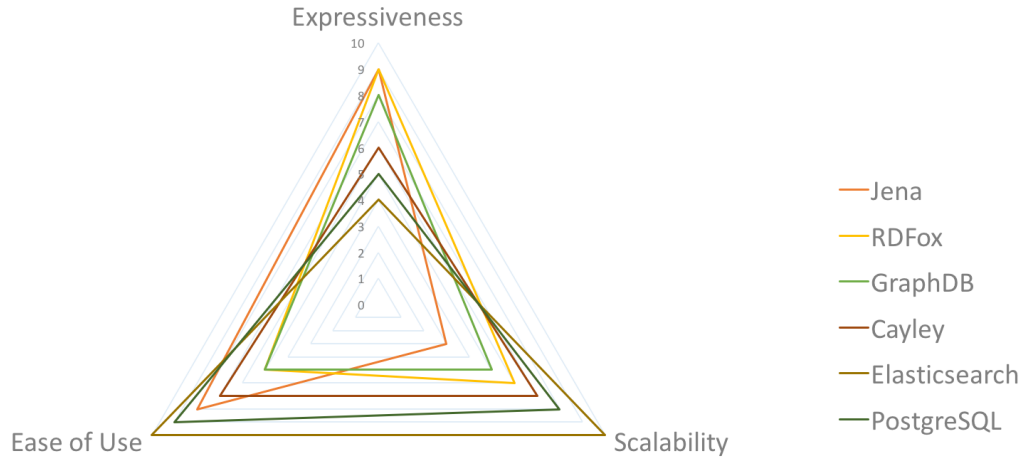
Figure 3.6: Semantic Systems Comparison

We graded the systems against each other ourselves. Results of our analysis are visualised in Figure 3.6. Results are normalized on a scale of 1 to 10, with 10 being highest score relative compared to other systems.

We can observe both Jena and RDFox having highest expressiveness. While Jena gives us ease of use, RDFox scores much higher at scalability potential. GraphDB provides a bit lower expressiveness, average scalability, ease of use is hampered by pricing and closed source model.

Elasticsearch promises highest scalability potential and highest ease of use as well, while having lower expressiveness. PostgreSQL is scalable as well, but trails Elasticsearch, while providing similar level of expressiveness and ease of use.

Cayley has shown to be in the middle on all three scores and is another interesting candidate for case study research.

Chapter 4

Case Study: Evaluating Expressiveness and Scalability

Idea of the case study is to compare selected solutions presented in previous chapter. We will compare them based on a simulated real use case and focus on different solutions' expressiveness and scalability. Apart from scalability and expressiveness we will take a look at some other factors such as how difficult it is to prepare a system, how long it takes for data to be loaded, how complex the query language used is. Our goal is to research and select the system that has the best combination of mentioned factors. System that could be best applied to real industry problems.

4.1   Problem Definition

We are trying to present geographical information and data connected to it in a meaningful manner. We want to have a knowledge graph about geography with focus on cities with which we can make useful and expressive queries. Queries need to be sufficiently quick and scale well with increasing quantity of data.

4.2   Test Cases

From analysis in previous chapter we have selected three systems that we will focus on and see how they can be used to solve our problem.

- **RDFox** as a native triple store solution, which we can use with different levels of expressiveness and has shown high scalability in existing benchmarks.

- **Elasticsearch** as it is proven to be highly scalable, although possibly with losing expressiveness. We want to try Elasticsearch with RDF/OWL data as base.

- **Cayley** as a graph database that has high focus on scalability and can give us lots of expressiveness through its own query language as well.

4.3   Test Data

For test data we will use open source geographical GeoNames data [17] which can be used semantically with GeoNames ontology [51]. We will use subsets of data for different tests and also extend it with additional rules where we would find appropriate. Full data-set contains about 150 million RDF triples which will enable us to research some high scalability problems. That amounts to 10,113,356 features, feature in this context is a single geographical entity such as city, country, beach, lake... We obtained the data using the RDF dump provided by GeoNames [1]

An example Geonames entry for city of Ljubljana[2]:

---

[1]http://download.geonames.org/all-geonames-rdf.zip
[2]Reachable at http://sws.geonames.org/3196359/about.rdf

```
<gn:Feature rdf:about="http://sws.geonames.org/3196359/">
...
<gn:name>Ljubljana</gn:name>
<gn:featureClass rdf:resource="http://www.geonames.org/ontology#P"/>
<wgs84_pos:lat>46.05108</wgs84_pos:lat>
<wgs84_pos:long>14.50513</wgs84_pos:long>
<gn:parentFeature rdf:resource="http://sws.geonames.org/3239318/"/>
<gn:parentCountry rdf:resource="http://sws.geonames.org/3190538/"/>
<gn:nearbyFeatures
rdf:resource="http://sws.geonames.org/3196359/nearby.rdf"/>
<rdfs:seeAlso rdf:resource="http://dbpedia.org/resource/Ljubljana"/>
...
</gn:Feature>
```

We only show a subset of more than 100 triples that give us information about this feature. We can observe some basic predicates such as name, latitude and longitude. Some of the other interesting predicates are *parentFeature* which tells us what is the first parent of this feature, *seeAlso* which gives us a link to another URI with more information about our resource and *nearbyFeatures* where we can see what features are close to selected feature.

4.4 Systems Architecture and Setup

Basic architecture of our systems is shown in Figure 4.1. We start from Geonames data which we have to convert to appropriate format depending on the system being used. We load the data into system's back-end data-store where it is persisting. Data is reasoned or indexed in the next step into the part of system that does the query answering. Clients and in our case tests conducted are then making queries to the query answering part of the system, which is the only part of the system available to clients.

For all the tests we were using machine with OS X operating system on 2.6 GHz Intel Core i5, 16 GB 1600 MHz DDR3 main memory and 256 GB SSD hard
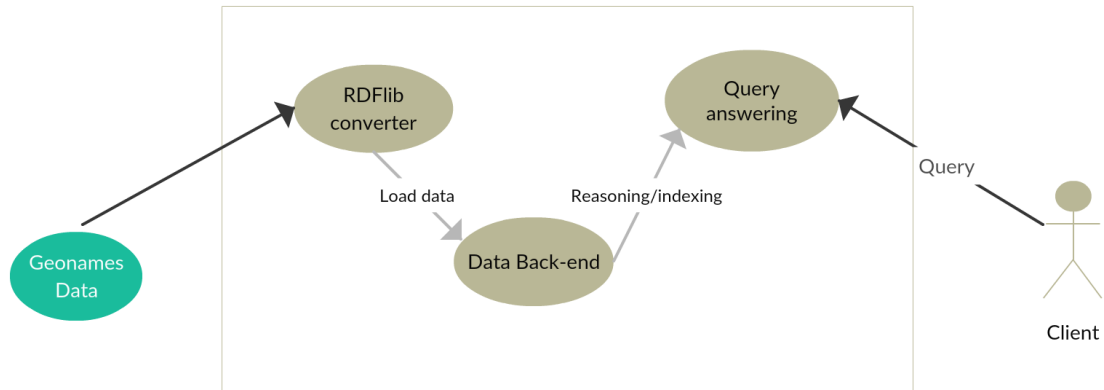
Figure 4.1: Systems Architecture

drive. Using the tool RDFLib [3] we were able to convert the data to be used in different formats and for different systems.

RDFox
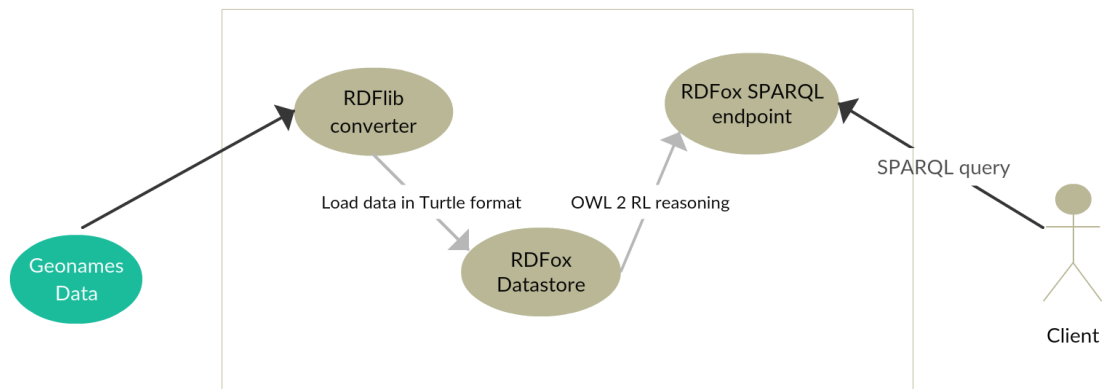
Figure 4.2: RDFox Architecture

RDFox only supports data stored in TURTLE format, so we needed to convert it from RDF/XML. We are using OWL 2 RL profile for reasoning, which promises to give us right balance between expressiveness and scalability. As RDFox is still under development, there were manual steps that needed to be taken to set it up

---

[3]https://github.com/RDFLib

fully. As listed in Figure 4.2 we loaded the data to RDFox datastore in TURTLE format, data was then reasoned to SPARQL endpoint where it was available for queries from clients.

Elasticsearch
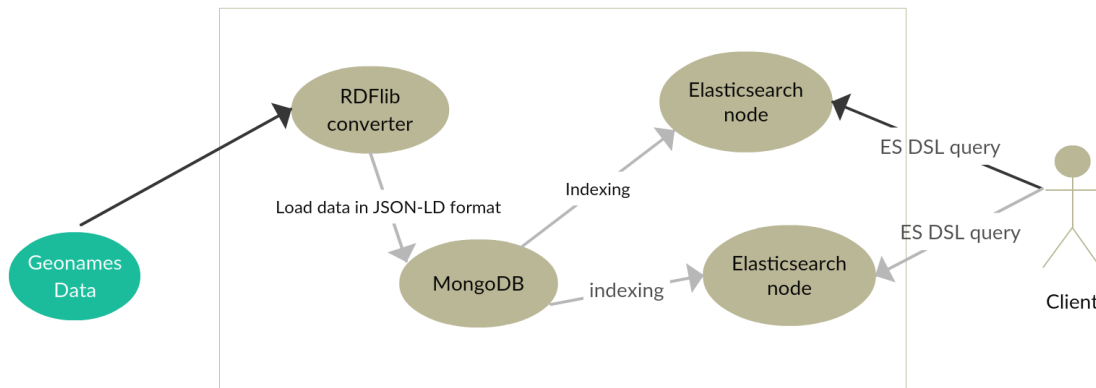
Figure 4.3: Elasticsearch Architecture

Elasticsearch only supports JSON format, converting data to JSON-LD was a slow task that could have an effect on scalability if use-case would require lots of conversions between formats. ElasticSearch supports multiple back-ends as well as a starting point for indexing. As seen in Figure 4.3 we selected MongoDB as a scalable key-value store. Multiple Elasticsearch nodes can be used for query answering, we provide details on that in 4.4.1.

Cayley

Cayley supports N-quads format only, which needed additional conversions using RDFLib. There are also multiple databases available to be used as back-end for Cayley. As listed in Figure 4.4 we opted for Bolt [6] as a back-end.

4.4.1 Systems Scaling

Each of the systems have different techniques to deal with scaling. When we performed the evaluation on smaller amount of triples we used the default settings
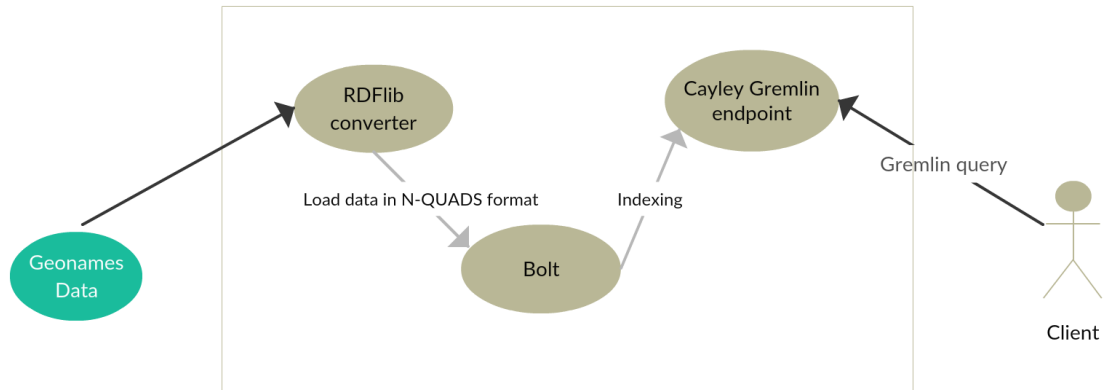
Figure 4.4: Cayley Architecture

and configuration that made minimal use of available hardware. For larger data-set we employed different scaling capabilities on both loading/reasoning level as well as query answering level.

- **RDFox** has support for different Datastores with increase in scale [36]. For the smaller data-set we used "Sequential" store type were reasoning is done with one thread, and also only one query at a time should be issued. It can store up to 2^{48} triples. For larger dataset we used "ParallelComplexWW" store type which employs a complex indexing scheme designed for highly-efficient data access. It can store up to 2^{48} resources and up to 2^{48} triples.

- **Cayley** is using in-memory, ephemeral store for small data set. With increase scale of data we switched to Bolt database which promises high performance on large amounts of data [6]. With Bolt we also get multi-threading support for querying and loading data which enables us to scale on increased data size. Cayley should be able to use much more of available hardware using that datastore.

- **Elasticsearch** supports many options to scale with increasing data size. Elasticsearch can operate on multiple nodes and each node can operate on multiple shards [18]. Each shard contains different part of data, which enables better support for parallelization and better use of available CPU time

and memory. For smaller subset of data we operated on 1 node containing 1 shard. When we moved to full scale of data we moved to 4 nodes, each of them containing 1 shard of data. We also gave the process more memory available with increasing heap size for the underlying Java process.

We designed the scaling process on all systems in such a way, that adding additional resources such as CPU and memory is possible and trivial afterwards. Although we could not fully set up same environments as big industry resources can provide, we tried to simulate scaling with similar configurations as would be in effect using those resources.

4.4.2 Data Load Times

Before we started testing query execution, we had to load all the data in RDF form to our systems. We presumed that data is already in the format that the system can use and didn't include conversion in load times. We still include any internal conversion to internal data types if applicable.

Listed below in Table 4.1 are times needed for data loading to complete. Any post-load processing such as reasoning is also included in load times. We are measuring time until the system is ready to start receiving and answering queries.

Table 4.1: Load times

	10,000 triples	150 mio triples
RDFox	0.1 ms	55 s
Cayley	1.2 s	585 min
ElasticSearch	5 s	122 min

RDFox has shown to be very quick, making good use of multi-threading for loading and reasoning in full data-set, while both ElasticSearch and Cayley have needed much longer, especially Cayley has shown that load and reasoning time is very slow and does not scale well with increasing amount of data. Elasticsearch load time is a split between preparing data in MongoDB and indexing it to Elas-

ticsearch. It is also worth mentioning that on full data-set, Elasticsearch was using
multiple nodes and shards, which made loading times slower as expected.

4.5 Answering Problems

We will go through examples of questions we want to answer with our system
and try to see if all systems in test are able to give us enough expressiveness to
answer each problem. As RDFox promises us highest level of expressiveness we will
construct SPARQL query used in RDFox first. We will then try to reach the same
level of expressiveness in both Cayley using Gremlin queries and ElasticSearch
using ES DSL queries. All the query times will be compared at different scale of
data used. Query times were measured using Apache Jmeter [4]. We made 1000
queries per each problem and measured an average of all timings to get the final
time that is presented in our thesis.

4.5.1 List all cities by name

First task for our system is a query that returns all the cities and return their
names. We should not need high level of expressiveness for this query, we presume
to observe problems with scalability as number of results should be very high.

- **RDFox SPARQL query**: [5]

```
PREFIX gn: <http://www.geonames.org/ontology#>
SELECT ?name WHERE {
    ?city gn:featureClass gn:P. ?city gn:name ?name}
}
```

---

[5]We will be omitting prefixes on subsequent queries for easier readability

- **Cayley Gremlin query**: [6]

```
g.V().
Has("http://www.geonames.org/ontology#featureClass",
"http://www.geonames.org/ontology#P")
.Out("http://www.geonames.org/ontology#name").All()
```

- **ElasticSearch DSL query**:

```
{
    "query" : {
        "match" : { "featureClass" : "P" }
    },
    "fields" : ["name"]
}
```

All three systems are able to return as the answer in a single query, queries are concise and easy to understand as well. Only basic knowledge of Geonames ontology is required to construct the query.

Table 4.2: List all cities query

	10,000 triples	150 mio triples	150 mio with pagination
RDFox	122 ms	575 s	409 ms per page
Cayley	140 ms	586 s	/
ElasticSearch	30 ms	58 s	51 ms per page

Table 4.2 shows times taken for queries on all three systems. Interestingly RDFox is performing quicker than Cayley, although both do not scale well on this

---

[6]See footnote 5

problem and query times on full data sets are very high. As expected Elasticsearch has shown to be more scalable by a factor of 10 on full data set.

On full data set we also tested queries with pagination, using limit and offset. Results on that are also part of table 4.2. Elasticsearch has been almost as performant as on a small dataset. RDFox has been slower, but still scaled well, while Cayley does not have any support for advanced pagination.

4.5.2 What cities are part of country X

We want to select all cities from selected country and list the city names. We start the problem from the name of the country, for example purposes we will use: "Republic of Slovenia".

- **RDFox SPARQL query**

```
SELECT ?city ?name WHERE {
    ?city gn:featureClass gn:P .
    ?city gn:parentCountry ?country.
    ?city gn:name ?name.
    ?country gn:name "Republic of Slovenia" }
```

- **Cayley Gremlin query**:

```
var country = g.V()
.Has("name", "Republic of Slovenia").ToValue
g.V().Has("featureClass","P")
.Has("parentCountry", country)
.Out("name").All()
```

- **ElasticSearch DSL query** On Elasticsearch we have to do a sub-query
  first to get the right country code of our country:

```
"query": {
    "bool": {
      "must": [
        {"match" : {"featureClass" : "A"}},
        {"match" : {"name" : "Republic of Slovenia"}},
    ]},
"fields" : ["countryCode"]}
```

We can then use results from the first query to construct the second query,
that returns us the results we want:

```
"query": {
    "bool": {
      "must": [
        {"match" : {"featureClass" : "P"}},
        {"match" : {"countryCode": "SI"}}
    ]},
"fields" : ["name"]}
```

On this problem we can see the increased expressiveness of SPARQL as it only
needs one query to get the answer. We had to use a sub-query to get the same
result in Cayley through Gremlin queries. For Elasticsearch query, we needed to
know what the country code was first, which had to be achieved by making another
query before-hand.

Table 4.3: Cities from selected country [7]

	150 mio triples	150 mio with paging
RDFox	921 ms	421 ms per page
Cayley	150 ms	/
ElasticSearch	20 ms	15 ms per page

RDFox has shown to be slow on big amount of data, Cayley being quicker by a factor of 10, while Elasticsearch by a factor of 100x. With paging RDFox scaled better. Cayley as mentioned before has no paging capabilities. Elasticsearch has shown to scale really well, especially with using paging queries it is by an order of magnitude quicker than other solutions.

4.5.3   Features from selected municipality

That problem was not solvable with basic RDF data provided by Geonames dump. Available data only enabled us to get the location of RDF document which contains information about the children features of each feature. For this problem we can use the concept of Linked Data which allows us to create typed links between data from different sources [4].

- **RDFox SPARQL Query** First query gives us the city URI and linked document for children features URI that we can later use:

```
SELECT ?city ?document WHERE {
    ?city gn:featureClass gn:A
    ?city gn:childrenFeatures ?document.
    ?city gn:name "Ljubljana. }
```

In this example the document is
http://sws.geonames.org/3239318/contains.rdf . RDFox tooling allows us to

---

[7]10,000 triples results are omitted from here on, as we focus on results when we scale data

dynamically extend the data with new triples we got from the document.
Second query after we extend the data with new data returns us names of
all of the children features:

```
SELECT ?name WHERE {
    ?feature gn:parentFeature
    http://sws.geonames.org/3239318
    ?feature gn:name ?name. }
```

- **Cayley Gremlin query**:

  Similarly to the RDFox query we need to extend the data from external
  sources first. We get the document with the following query:

```
g.V().Has("featureClass","A")
.Has("name", "Ljubljana")
.Out("parentFeatures").All()
```

  Cayley has no native support for extending data with external linked data,
  so for adding information from a separate document we had to provide a
  manual script for updating data, it could not have been done dynamically
  mid-query. Final query we make for getting all "parentFeatures" is:

```
g.V()
.Has("parentFeature", "http://sws.geonames.org/3239318")
.All()
```

- **ElasticSearch DSL query** Elasticsearch does not support triple notation
  or has any connection with triple data. Children features of municipality is
  a problem that Elasticsearch with our data set does not have enough expres-
  siveness for solving. Data and index would need to be extended manually to
  support it.

Table 4.4: Features from selected municipality

	Getting linked document (150 mio triples)	Linking document and query
RDFox	421 ms	634 ms + 2.1 s
Cayley	140 ms	234 ms + 6 s
ElasticSearch	/	/

Both RDFox and Cayley could solve a problem using linked data document. Results shown in 4.4 show us that both were quick with retrieving the document used for extending data. Extending data was dynamic and faster in RDFox. Any quantifiable measures here are depending on external resources, in this case on Geonames repository, so they can depend on external data-set capabilities.

For Cayley we had to create a bespoke script that helped us reload the system with new data, which made it not very scalable on multiple queries.

Elasticsearch was unable to solve that problem, so there are no timings provided for it.

4.5.4   Get features that are near the selected city

Similarly to the previous problem, that problem is also solvable using Linked Data principles, as Geonames ontology has information on nearby features as well. As "nearby" can be defined by distance and not just ontology defined predicates, there are other ways of solving that problem. It is worth noting that Linked Data data-sets are not always reliable and they depend on data quality ensured by data-set provider.

- **RDFox SPARQL Query** Similarly to 4.5.3 we first need to extend our data with linked document. We are omitting that part here. Second query after we extend the data with new data, returns names of all nearby features:

```
SELECT ?name WHERE {
    ?feature gn:nearby
    http://sws.geonames.org/3239318
    ?feature gn:name ?name. }
```

- **Cayley Gremlin Query** as in 4.5.3 we had to extend our data in Cayley as well. After extending we could use this query to get all nearby features:

```
g.V("http://sws.geonames.org/3239318")
.Out("nearby").All()
```

- **Elasticsearch DSL Query** Elasticsearch does not support "nearby" predicate in triple notation with linked data. Partial level of expressiveness can be reached using queries based on location fields which are storing latitude and longitude. First query we make is:

```
"bool": {
    "must": [
        {"match" : {"featureClass" : "P"}},
        {"match" : {"name": "Ljubljana"}}]
    }
```

We use the results to make another query searching by geometry fields provided by first query. For selecting nearby features we can make a manual decision on what we perceive as nearby. For our example we will be using 20 km as radius of nearby features.

```
"bool": {
  "must": {"match_all" : {}},
  "filter" : {
      "geo_distance" : {
          "distance" : "20km",
              "pin.location" : {
                  "lat" : 46.05, //from previous query
                  "lon" : 14.51 //from previous query
              }
          }
      }
  }
```

That approach does have some drawbacks, as it leaves defining nearby features on a user, instead of provider of data. Nearby is not always related to distance, as it doesn't take in effect other geographical information such as rivers, mountains, bridges.

RDFox and Cayley were both able to solve the problem using Linked Data, while Elasticsearch was able to provide a partial solution relying on user deciding what is classified as nearby and not using full capabilities and expressiveness provided to us by data-set.

Table 4.5: Features nearby to selected city

	Full query on 150 mio triples
RDFox	934 ms + 2.2 s
Cayley	321 ms + 6 s
ElasticSearch	9 ms + 14 ms

Similarly to the previous problem, both RDFox and Cayley were sufficiently quick on first query, but needed long time for extending data and were reliant on externally hosted data.

Elasticsearch solved problem using different approach which made it faster and scaled well on full data-set.

4.5.5 Get cities that are X km from my current location

For this problem we will take arbitrary location defined in geographical latitude and longitude. We will try to find all features that are under X km away from the arbitrary location. We will set radius of 20km from selected location as our example.

- **RDFox SPARQL Query** As current location is not something defined by RDF notation triples, we have to go through filtering feature provided by SPARQL. We cannot define distance in km, we have to rely on distances in latitude and longitude degrees. We will use location of University of Ljubljana, Faculty of Computer and Information Science (46.050107,14.4667993) as our current location. We will construct the query for locating all features in a box with +-0.2 longitude and latitude difference:

```
SELECT ?name WHERE {
    ?city geo:lat ?latitude .
    ?city geo:long ?longitude .
    ?city gn:name ?name .
    ?city gn:featureClass gn:P
      filter(
        ?latitude > 46.050107−0.2  &&
        ?latitude < 46.050107+0.2 &&
        ?longitude > 14.4667993−0.2  &&
        ?longitude < 14.4667993+0.2)
    }
```

It is worth saying that if our current location was part of our Geonames data-set, our query could be done differently, using nearby location predicate

instead of filtering.

- **Cayley Gremlin Query** Cayley Gremlin query language does not support value comparison yet, so it does not enable us using current location. Only way to successfully construct that query would be to use location already in data-set, similarly to the previous problem 4.5.4.

- **Elasticsearch DSL Query** Using Elasticsearch DSL, we can construct the query similarly to how we have done it in our previous problem 4.5.4

```
"bool": {
  "must": {"match_all" : {}},
  "filter" : {
      "geo_distance" : {
        "distance" : "20km",
          "pin.location" : {
              "lat" : 46.050107,
              "lon" : 14.4667993
          }
      }
    }
}
```

Table 4.6: Cities nearby to current location

	10,000 triples	150 mio triples
RDFox	156 ms	845 ms
Cayley	/	/
ElasticSearch	9 ms	15 ms

RDFox was able to answer the problem, but did not scale well with increasing number of triples. Cayley did not provide enough expressiveness for solving the

problem. Elasticsearch proved to be expressive enough and scales with minimal losses when increasing our data-set size.
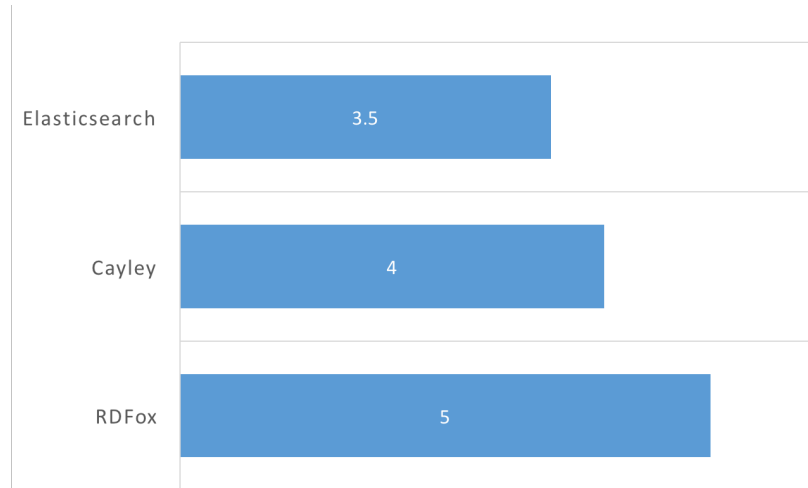
4.6 Results

Figure 4.5: Problems Solved

As seen in Figure 4.5, RDFox was the only system able to solve all 5 of provided problems. Cayley was unable to solve one, while Elasticsearch was not expressive enough for one problem and solved another one with a partial solution. RDFox is clearly the system with most expressive power from the systems in our case study.
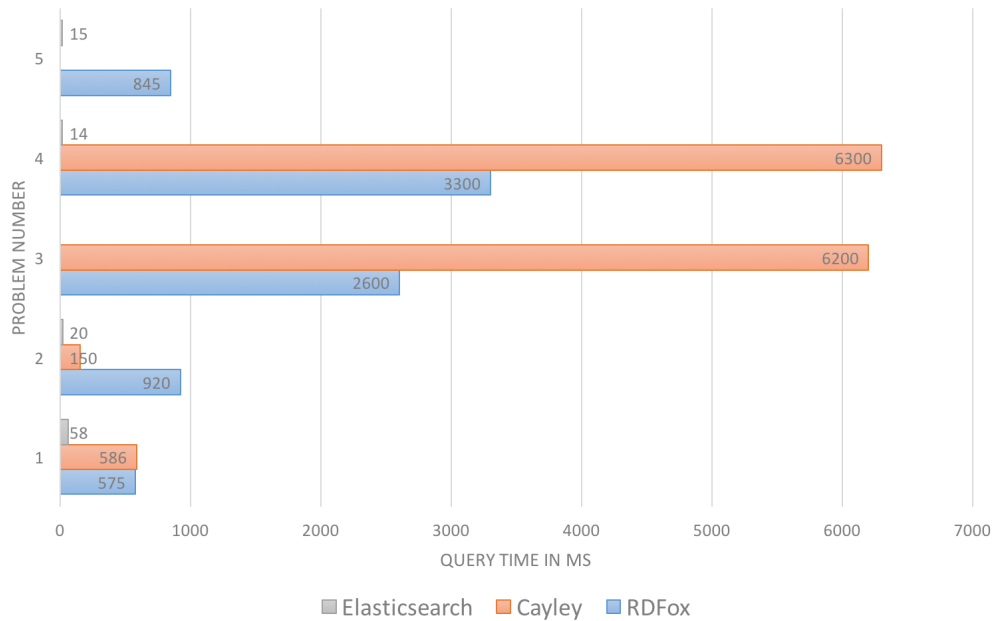
Figure 4.6: Query Timings

Figure 4.6 shows query times taken on each problem from 4.5 at highest scale of data. If system could not produce a solution for a problem, there is no query time listed. We can observe Elasticsearch having the fastest queries on all problems it was able to solve, while Cayley query times were slowest on all but one problem. RDFox was slower than Elasticsearch, but was able to solve all the problems. We normalized each result timings on every problem relative between systems and scoring unsolved problem as 0. We averaged scores across all problems into one, grading Elasticsearch with a score of 10, RDFox with 3 and Cayley with 2 for scalability.
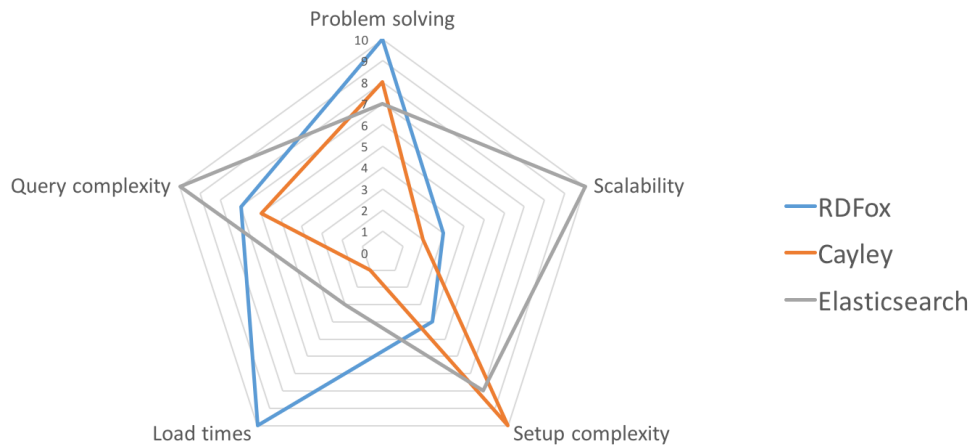
Figure 4.7: Semantic Systems Comparison

Expressiveness (problem solving) and scalability are our main dimensions that we discussed above. Others are setup times/complexity for system, load times of data. We added another dimension for how complex the queries constructed are for developers.

We collated the research from the case study and according to it, graded all the systems on different dimensions. Radar chart in Figure 4.7 shows how all three systems score in different dimensions that were presented in the case study. Problems solved are normalized from number of problems solved by each system to 0-10 scale as seen in Figure 4.5. Data load times, setup times and scalability are normalized from time taken to a scale of 0-10, for scalability we take in account if system was unable to solve the problem at all. Query complexity and setup complexity were graded by ourselves for each system from 0-10 depending on complexity of constructing queries and setting up the system as part of the case study. Higher score is better on all dimensions.

Elasticsearch has shown best results in scalability and query complexity, while having lower setup complexity and load times. As mentioned before, it also had lowest number of problems solved.

Cayley was scored best in setup complexity while scoring lower for all other

dimensions, especially scalability was lower than expected before our case study was conducted.

RDFox was the only system able to solve all problems. It scored first in data load times and low in scalability, proving that highest expressiveness system is most of the times lowest in scalability scores.

Table 4.7: Semantic Systems Decision Model

Decision Dimensions	Weigths	RDFox	Cayley	Elasticsearch
Dimension	Wt.	1	2	3
Expressiveness	5.0	10	8	7
Scalability	5.0	3	2	10
Data Load Times	1.0	10	1	3
Setup Complexity	1.0	4	10	8
Query Complexity	1.0	7	6	10
Weighted Scores		86.0	67.0	106.0

We added results to a decision model that can be seen in Table 4.7 with higher weights for expressiveness and scalability and lower weights for other dimensions. Elasticsearch has shown to be the best fit for our problems, although low expressiveness score is problematic. If high expressiveness is needed, then RDFox is showing to be the best choice, with Cayley scoring lowest in cumulative score as well as both expressiveness and scalability.

Chapter 5

Conclusions

Semantic systems has shown to provide high level of expressiveness, but there were problems with scalability once data was larger and queries were operating on a big amount of triples. Fully semantic reasoning supported systems like RDFox have shown that they are not fully ready for industry solutions that need quick responses and strong scalability. We can observe that there is a clear correlation between system being more expressive and by result of that system being less scalable.

We have observed that semantic data can be used in non purely semantic systems that are built for scalability first, great example being Elasticsearch and Cayley to some extent.

We see big opportunities in using highly expressive semantic systems only for problems needing that high expressiveness. Depending on the problem presented there are different choices of a system that would best suit the problem solution. Systems researched all have different purposes and solve different problems.

For industry problems, depending on the problem you are trying to solve we propose a combination systems presented. For less expressive queries, highly scalable systems such as Elasticsearch is a better choice, while for a subset of problems that are more difficult and need more expressiveness, RDFox could be a choice. Hybrid solution of both would be best suited for industry problems.

5.1   Challenges

Semantic data quality was not the main topic of our case study, but previous works have shown that Semantic data can be un-maintained and of low quality [16]. While using the Geonames data-set for our case study, it has shown that there are other possible rules and features missing in both ontology and set of triples provided. With better data we could use higher expressiveness provided to us by Semantic systems. There are lots of potential in these systems, but if data is not structured and prepared correctly this addition to expressiveness and reasoning is not used in full and systems using relational or document data can offer us better scaling on this level.

5.2   Future work

In the thesis we focused on testing scalability with increasing amounts of data. We didn't fully research how scalability works on the systems tested with increasing processing power. We weren't able to dive into full industry scale hardware support to use multi-threading, higher memory and CPU to large industry level scale. With more resources available, further research and comparison between Semantic systems could be conducted.

There are different aspects important to industry problems that we didn't touch in depth here. Ease of use for developers was mentioned briefly, support for easy deployments could be another, fault tolerance, future plans of the team behind the system and other aspects could be researched more in depth and are good ideas for future work.

Thesis focused on different systems and tested them as a whole. Through testing different problems we found that different systems behaved better for specific problems. For problems that needed low expressiveness, systems with high expressiveness availability did not scale as well as those with lower expressiveness. On the other hand there were problem that only highly expressive systems could solve. Idea for future work would be to implement and test a hybrid solution which would join multiple systems together and use them together on same set of

data. Different systems are better suited for different problems. Solution could focus on hiding and abstracting these decisions from the developer. Wrapping all underlying systems in one query language. That way, instead of having to decide when to use purely Semantic Web system, hybrid solution could delegate harder problems to these systems and delegate other problems to a more scalable system.

Bibliography

[1] Grigoris Antoniou and Frank Van Harmelen. "Web ontology language: Owl". In: *Handbook on ontologies*. Springer, 2004, pp. 67–92.

[2] Think Aurelius. *Titan*. 2016. URL: http://thinkaurelius.github.io/titan/.

[3] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The semantic web". In: *Scientific american* 284.5 (2001), pp. 28–37.

[4] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data-the story so far". In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), pp. 205–227.

[5] Christian Bizer and Andreas Schultz. *The berlin sparql benchmark*. 2009.

[6] Bolt. *Bolt*. 2016. URL: https://github.com/boltdb/bolt.

[7] Dan Brickley and Ramanathan V Guha. "RDF vocabulary description language 1.0: RDF schema". In: (2004).

[8] Cayley. *Cayley*. 2016. URL: https://github.com/google/cayley.

[9] Eugene Inseok Chong et al. "An efficient SQL-based RDF querying scheme". In: *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment. 2005, pp. 1216–1227.

[10] Chris J Date and Hugh Darwen. *A Guide To Sql Standard*. Vol. 3. Addison-Wesley Reading, 1997.

[11]   Korry Douglas and Susan Douglas. *PostgreSQL: a comprehensive guide to building, programming, and administering PostgreSQL databases*. SAMS publishing, 2003.

[12]   Elastic. *ElasticSearch*. 2016. URL: `https://www.elastic.co/use-cases`.

[13]   The Apache Software Foundation. *Apache Lucene*. 2016. URL: `https://lucene.apache.org/`.

[14]   The Apache Software Foundation. *Jena SDB*. 2016. URL: `https://jena.apache.org/documentation/sdb`.

[15]   The Apache Software Foundation. *Jena TDB*. 2016. URL: `https://jena.apache.org/documentation/tdb`.

[16]   Christian Fürber and Martin Hepp. "Using semantic web resources for data quality management". In: *Knowledge Engineering and Management by the Masses*. Springer, 2010, pp. 211–225.

[17]   Geonames. *Geonames*. 2015. URL: `http://www.geonames.org/`.

[18]   Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. " O'Reilly Media, Inc.", 2015.

[19]   Gremlin. *Gremlin*. 2016. URL: `https://gremlindocs.com/`.

[20]   The PostgreSQL Global Development Group. *PostgreSQL*. 2016. URL: `http://www.postgresql.org/`.

[21]   The W3C SPARQL Working Group. *SPARQL 1.1 Overview*. 2013. URL: `http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`.

[22]   W3C Owl Working Group et al. "OWL 2 Web Ontology Language Document Overview". In: (2009).

[23]   W3C Working Group. *RDF 1.1 Primer*. 2014. URL: `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

[24]   Florian Holzschuher and René Peinl. "Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. ACM. 2013, pp. 195–204.

[25] Franz Inc. *Allegrograph*. 2016. URL: `http://franz.com/agraph/allegrogr aph/`.

[26] Thomas Johnson. "Indexing linked bibliographic data with JSON-LD, Bib-JSON and Elasticsearch". In: *Code4lib Journal* 19 (2013), pp. 1–11.

[27] Vaibhav Khadilkar et al. "Jena-HBase: A distributed, scalable and efficient RDF triple store". In: *Proceedings of the 11th International Semantic Web Conference Posters & Demonstrations Track, ISWC-PD*. Vol. 12. Citeseer. 2012, pp. 85–88.

[28] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. "OWLIM–a pragmatic semantic repository for OWL". In: *Web Information Systems Engineering WISE 2005 Workshops*. Springer. 2005, pp. 182–192.

[29] Graham Klyne and Jeremy J Carroll. "Resource description framework (RDF): Concepts and abstract syntax". In: (2006).

[30] Vladimir Kolovski, Zhe Wu, and George Eadon. "Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system". In: *The Semantic Web–ISWC 2010*. Springer, 2010, pp. 436–452.

[31] Rafal Kuc and Marek Rogozinski. *ElasticSearch server*. Packt Publishing Ltd, 2013.

[32] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.10 (2004), p. 2004.

[33] Vladimir Mironov et al. "Benchmarking triple stores with biological data". In: *arXiv preprint arXiv:1012.1632* (2010).

[34] Boris Motik et al. "Owl 2 web ontology language: Profiles". In: *W3C recommendation* 27 (2009), p. 61.

[35] Boris Motik et al. "Parallel materialisation of datalog programs in centralised, main-memory RDF systems". In: *Proc. AAAI*. 2014, pp. 129–137.

[36] Yavor Nenov et al. "RDFox: A Highly-Scalable RDF Store". In: *The Semantic Web-ISWC 2015*. Springer, 2015, pp. 3–20.

[37] Ontotext. *GraphDB*. 2015. URL: http://ontotext.com/products/graphd
      b/editions/.

[38] Oracle. *Oracle Database*. 2016. URL: http://www.oracle.com/technetwor
      k/database/index.html.

[39] University of Oxford. *RDFox*. 2015. URL: http://www.cs.ox.ac.uk/isg/
      tools/RDFox/.

[40] University of Oxford. *RDFox tests*. 2014. URL: http://www.cs.ox.ac.uk/
      isg/tools/RDFox/2014/AAAI/.

[41] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. *Efficient query answer-
      ing for OWL 2*. Springer, 2009.

[42] W3C Recommendation. *RDF Schema 1.1*. 2014. URL: http://www.w3.org/
      TR/2014/REC-rdf-schema-20140225/.

[43] Marko A Rodriguez. "The Gremlin graph traversal machine and language
      (invited talk)". In: *Proceedings of the 15th Symposium on Database Program-
      ming Languages*. ACM. 2015, pp. 1–10.

[44] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. "The semantic web re-
      visited". In: *Intelligent Systems, IEEE* 21.3 (2006), pp. 96–101.

[45] Evren Sirin and Bijan Parsia. "SPARQL-DL: SPARQL Query for OWL-DL."
      In: *OWLED*. Vol. 258. 2007.

[46] Evren Sirin et al. "Pellet: A practical owl-dl reasoner". In: *Web Semantics:
      science, services and agents on the World Wide Web* 5.2 (2007), pp. 51–53.

[47] Gregory Smith. *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd,
      2010.

[48] OpenLink Software. *Virtuoso*. 2016. URL: http://virtuoso.openlinksw.
      com/dataspace/doc/dav/wiki/Main/.

[49] Pedro Szekely et al. "Building and Using a Knowledge Graph to Combat
      Human Trafficking". In: *The Semantic Web-ISWC 2015*. Springer, 2015,
      pp. 205–221.

[50] Neo Technology. *Neo4j*. 2016. URL: http://neo4j.com/.

[51] Bernard Vatant and Marc Wick. *Geonames ontology*. 2012.

[52] Yujiao Zhou et al. "Making the most of your triple store: query answering in OWL 2 using an RL reasoner". In: *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2013, pp. 1569–1580.