Unconventional Store Systems for RDF Data

Comparison between Registry Systems used as Semantic Web RDF Data Stores

Master Thesis

Iliya Enchev

September 2012

Thesis supervisors:

Prof. Philippe Cudre-Mauroux
Dr. Gianluca Demartini
eXascale Infolab

UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

eXascale Infolab
Department of Informatics
University of Fribourg (Switzerland)

eXascale Infolab

"There is no real ending. It's just the place where you stop the story."

- *Frank Herbert*

Acknowledgements

Many people have supported the successful realization of this work and I would like to pay them here the deserved tribute. My supervisor *Prof. Dr. Philippe Cudre-Mauroux* have always guided and encouraged me during the time I worked on this thesis. He has always been helpful and supportive in times of difficulties and doubts.
The whole *eXascale Infolab* at the University of Fribourg headed by my supervisor have created an accommodating and friendly working atmosphere. They have always granted me their support, helped me with technical issues or given me some valuable ideas during our lunch breaks. Thanks to *Gianluca Demartini, Marcin Wylot, Djellel Difallah, Roman Prokofyev, Jigé Pont, Alberto Tonon, Mariusz Wisniewski, Michael Luggen.*
I would also like to express my gratitude to my girlfriend Julie, my parents, my family and friends who have put up with me during the time I was working on my thesis.

Abstract

The efficient reasoning about entities across distributed information sources and their efficient resolution are crucial prerequisites if the prescriptions of the Semantic Web movement are to come into reality. These goals along with some of the most important principal and technical issues of the Internet and the Web are presented in this work.
Entity registry systems provide capabilities like storing, serving and resolving data entities which could greatly facilitate the Semantic Web. Five such systems or approaches are presented in this work - Domain Name System (DNS), Digital Object Architecture (DOA), Entity Name System, Chord DHT and CoralCDN. Further four data storage solutions are considered for their qualities in the context of handling structured RDF data entities and are put to extensive benchmarking tests with the help of a developed Java benchmarking suit. These storage solutions are AllegroGraph, Open Chord, Apache Cassandra and MySQL.

Keywords: Semantic Web, RDF, OWL, N-Triples, Linked Data, Linking Open Data, Digital Object Architecture (DOA), DNS, Entity Name System (ENS), Chord, DHT, Entity Registry, Apache Cassandra, Key-value store, Benchmarking, Performance, Distributed store, Cluster

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

Since mankind started gathering information about the world around and begun recording it on different sources, the need for order appeared as a necessity if one wanted to reuse this information. That brought the first inscriptions on walls in caves, stone columns etc. until the book appeared and then naturally the libraries. The libraries show very well the need for ordered information and are to some extent very similar to database systems. Nowadays in the digital world the amounts of information have a much different measurement. There are certain fields like science, finance, commerce, where not a single system or machine or a library is suited for the purpose of storing and providing the information to the public. That is where entity registries come into light. They are intended to store organize and provide useful services on huge amounts of data.

1.1. Motivation and Goals

The World Wide Web which is the biggest application of the Internet provides unimaginable amount of information resources and knowledge, which are foundation of many applications that bring added value to humanity. Although the Web was originally designed with the goal to be an integrated whole, the great part of its resources are scattered separately across the Internet. The motivation of this thesis is to explore the means for integrating the separate knowledge bases.

For this purpose the goals of this work are to:

- outline some of the more important issues and implications of the Web
- establish the state of the art of software systems with main functionality - managing digital entities in a network
- provide a hands-on comparison and measurement of several types of entity management systems with their performance, main features and characteristics

1.2. Organization

The rest of the work is organised as follows: in the 2-nd section several principal issues and approaches concerning digital entities on the internet, are presented. The 3-rd section presents 4 systems that provide in one way or another entity registry functionalities and attack specific problems. These systems are presented with their more important features. A benchmarking test suit is then presented in the 4-th section and the results of some of the executed tests is shown in the form of charts and tables and comments.

1.3. Notations and Conventions

- Formatting conventions:
  - **Bold** and *italic* are used for emphasis and to signify the first use of a term.
  - SansSerif is used for web addresses.
- The present report is divided in Chapters. Chapters are broken down into Sections. Where necessary, sections are further broken down into Subsections, and Subsections may contain some Paragraphs.
- Figures, Tables and Listings are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j*.
- As far as gender is concerned, I systematically select the feminine when possible.
- Source code is desplayed as follows:

```
Matrix3f rotMat = new Matrix3f();
rotMat.fromAngleNormalAxis( FastMath.DEG_TO_RAD * 45.0f, new Vector3f( 1.0f, 0.0f, 0.0f));
box.setLocalRotation( rotMat );
```

2
Data Challenges in Today's Internet

In this chapter several problems and approaches of different nature are considered. Some of these are principle or design-related other are technical. What they have in common is that they are related in a certain way to distributed entity registry systems and data integration on the Web in general. The following paragraphs are ordered where possible from higher, broader conceptual level to more detailed one.

2.1. Exponential Growth of Data on the Internet

With the advance of technology more and more fields of live and science become data-driven. This means that data is being created everywhere for many different purposes

and is being used in many different ways, (e.g. sensors gathering climate information, astronomers taking pictures of space for discovering new objects etc). According to IBM 90% of the data in the world today has been created in the last two years alone [6]. Some of the data generated remains for internal use by its creators, but a substantial share is generated with the intention to be shared with the public through the internet.
This growth together with the developing need to effectively utilize the available data is probably the cause of most of the principle or technical challenges that are further discussed in this work.

2.2. Information Islands

With the growth of data on the Internet many knowledge bases are being created or expanded constantly. Every knowledge representation has its purpose and is best at presenting a certain domain (e.g. science, medicine, commerce, etc.). Logically a more powerful representation is created by the unifications of more domain based representations. In theory if we broadly consider most of the artefacts on the Internet as some kind of knowledge representations, they should already be connected as they are on the Internet. Those artefacts that are relevant to humans are connected through the Web (HTTP), and those that are for machine consumptions follow some special protocols (e.g. RPC, FTP or BitTorrent). In most cases though, this connectedness is not enough. The Web provides means of access mainly to humans, and large scale automatic access, aggregation and/or analysis of data taken from different Web sources by machines is mostly impossible. The case with the machine intended protocols is also not solved as there is large diversity of protocols and means of network operations which globally are difficult to integrate.

Some aspects of the isolated information islands are tackled by several approaches that follow.

2.3. Metadata – Common Standards for (Machine) Readability

A solution dealing with the above mentioned issue is the metadata. Metadata is a broad term that has different meanings in different contexts. Generally it is defined as “data about data”. This definition does not fit well though in the context of the Internet and the Web. Tim Berners-Lee defines metadata as “machine understandable information about web resources or other things” [26]. “Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource” as stated by [32]. It is useful for resource discovery and can make different protocols interoperate more easily by providing the necessary details in human- or machine-understandable form. Metadata can also play an important role in choosing the protocols that are used when certain information is exchanged. With its help a system could decide adaptively which sorts of protocols to use to efficiently distribute information, dynamically as a function of the readers.

2.4. Semantic Web

Although the Web was designed as an information space of interlinked hypertext documents that should leverage the communication for human as well as for machines, there exist a major obstacle. In reality most of the information on the Web is designed for human consumption.

Generally what is being presented in most cases on a web page is derived from well-organized database structure, nevertheless the end result is not evident to automatic machines browsing the Web [42]. For the solution of this problem there are two obvious approaches. One goes in the direction of making the machines behave like humans, which delves in the fields of artificial intelligence, machine learning and data mining. The other approach goes in direction of changing the data that is exchanged on the Web so it is expressed in a machine processable form with the help of metadata. This is the realm of the Semantic Web [42].

The Semantic Web should be a global space for the seamless integration of countless knowledge bases into an open, decentralized and scalable knowledge space. For this purpose metadata data models are used to conceptualize and model the information provided by separate web resources across the Web. The most widely accepted such model is *Resource Description Framework* (RDF). It is based upon the idea of making statements about resources, either Web resources - pages, documents etc. or anything else that can be named. These statements are in the form of *subject-predicate-object* expressions and are also known as *triples* in RDF terminology.

The subject in a triple denotes the resource, the predicate denotes characteristic or aspect of the resource and expresses a relationship between the subject and the object. A simple example could be the sentence “Elephants are grey” which in RDF triple form would be presented as a *subject* denoting the concept or the animal species “elephant", a *predicate* denoting “has a color", and an *object* denoting the color “gray”. The concept of a *triple* is very simple at a first glance but in the same time is naturally suited for creating powerful knowledge representations in the form of labelled, directed multi-graph. Imagine when chaining multiple triples, then much more complicated statements can be described. It comprises a family of specifications and uses a variety of syntax formats among which are OWL[1] and RDFS[2].

It is widely considered [8] that much progress has been done to make the vision of Semantic Web happen, but it is still far from practical reality. The Semantic Web offers a solution for the above mentioned information islands problem, as it is capable of providing a common language and schemas for the description of arbitrary resources among many individual knowledge bases. Still there is a series of hindrances – logical and technical, that have to be overcome if the Semantic Web should really work.

2.5. Universal Identifiers

The correct functioning of the Semantic Web and Resource Registry Systems in general is based on correct identification through unambiguous unique identifiers. With the

[1]Web Ontology Language
[2]RDF Schema

explosive development of the Web the most widespread identifiers have become the URIs. They are also adopted for the purpose of the Semantic Web (RDF[3] and OWL). Linked data [4] which is one of the biggest semantic web initiatives (see 2.7) is based around URIs.

Ideally the integration of information islands into a global knowledge space should be based on the usage of URI for referring to any type of resource [8]. Key factor here is that "the global scope of URIs promotes large-scale network effects: the value of an identifier increases the more it is used consistently" [5]. Indeed if the same identifier is used for naming nodes in more than one knowledge representation (RDF/OWL) graphs, these nodes can be collapsed thus unifying the graphs and integrating the knowledge the graphs convey [8]. For this to work in reality there are two important prerequisites that need attention (as pointed out by Bouquet et al in "ENS" [8]): on the one hand, the usage of the same URI for two or more different resources should be avoided, as this creates ambiguity also known in information integration as "false positives"; on the other hand the unnecessary URI aliases (i.e. associating different URIs with the same resources) brings division into the web of related resources causing "false negatives" in information integration. The authors of ENS propose a solution of these two problems, focusing more on the second one as it has greater influence in reality.

2.6. Dereferencing URIs

Any HTTP URI that is used to denote a resource part of an RDF triple should be dereferenceable, meaning that HTTP clients can look it up using the HTTP protocol and retrieve a description or representation of the resource that is identified by the URI. This applies to URIs that are used to identify classic HTML documents - information resources, as well as URIs that are used in the Semantic Web context to identify real-world objects and abstract concepts [20].

Descriptions of resources are usually embodied in the form of Web HTML documents, which are intended to be read by humans. Descriptions that are intended for consumption by machines are represented as RDF data.

When a URI identify a real-world object, it is essential to not confuse the objects themselves with the Web documents that describe them. For this reason it is a common practice to use different URIs to identify real-world object and the document that describes it, in order to eliminate disambiguation. This way separate statements about an object and a document that describes the object can be made. For example the year a building was built can be much different from the creation date of a document that describes this building. This differentiation through use of separate URIs is crucial for the coherence of the Web of Data [20].

Together with the difference between the real objects and their descriptions there can also be different representations of the same resource (represented by the same URI). As was mentioned above the Web is intended to be an information space usable by humans as well as by machines and depending on who is interested in a certain resource there can be different, e.g. HTML for humans and RDF for machines. Still this should be the same data, only the appearance, the way the data is presented should change and this

[3]Resource Description Framework
[4]http://linkeddata.org/

can be achieved by an HTTP mechanism called *content negotiation* [18]. There are two types of content negotiation – server-driven and agent-driven.

By server-driven negotiation the selection of the best representation is made by an algorithm located at the server. Selection is based on the available representations of the response (possibilities such as language, content coding, formats like JSON, NTriples, etc.) and the contents of particular header fields in the request message or on other information pertaining to the request (e.g. network address of the client) [18, sec. 12.1].

In the case of agent-driven negotiation, selection of the most appropriate representation for a response is performed by the user agent after receiving an initial response from the origin server. Selection is based on a list of the available representations of the response included within the header fileds or entity-body of the initial response, with each representation identified by its own URI. Selection from among the representations may be performed automatically or manually by the user selecting from a generated menu [18, sec. 12.2]. There is also a combination of both negotiation types which is called transparent negotiation.

The above explained mechanisms are suited mainly for information resources. For dereferencing real-world objects there are two other strategies. These strategies are called *303 URIs* and *hash URIs*. They both ensure that objects and the documents that describe them are not confused, and that humans as well as machines can retrieve appropriate representations [20].

2.6.1. 303 URIs

Real-world objects, like houses or people, can not be transmitted over a network using the HTTP protocol. That is why it is not possible to directly dereference URIs that identify real-world objects. The 303 URIs strategy provides one solution in which the server when asked for a real world object, responds with the HTTP response code *303 See Other* and the URI of a Web document which describes the real-world object. This is called a *303 redirect*. In a second step, the client dereferences this new URI and gets a Web document describing the real-world object [20].

Dereferencing a HTTP URI that identifies a real-world object or abstract concept involves a procedures consisting of four steps (as pointed by [20, sec. 2.3.1]):

1. The client performs a HTTP GET request on a URI identifying a real-world object or abstract concept. If the client is a Linked Data application and would prefer an RDF/XML representation of the resource, it sends an *Accept: application/rdf+xml* header along with the request. HTML browsers would send an *Accept: text/html* header instead.
2. The server recognizes that the URI identifies a real-world object or abstract concept. As the server can not return a representation of this resource, it answers using the HTTP *303 See Other* response code and sends the client the URI of a Web document that describes the real-world object or abstract concept in the requested format.
3. The client now performs an HTTP GET request on this URI returned by the server.
4. The server answers with a HTTP response code *200 OK* and sends the client the requested document, describing the original resource in the requested format.

2.6.2. Hash URIs

One widely acknowledged disadvantage of 303 URI strategy is that it requires two HTTP requests to retrieve a single description of a real-world object. In this sense a more efficient solution, which avoids the two requests is provided by the hash URI strategy.

The hash URI strategy is based around the characteristic that URIs can contain a *fragment*, a special part that is separated from the base part of the URI by a hash symbol (#) [13].

When a client wants to retrieve a hash URI, the HTTP protocol requires the fragment part to be stripped off before requesting the URI from the server. This means a URI that includes a hash cannot be retrieved directly and therefore does not necessarily identify a Web document. This enables such URIs to be used to identify real-world objects and abstract concepts, without creating ambiguity [13, 20].

2.6.3. Choosing the right dereferencing strategy

Based on the specific characteristics of the two dereferencing approaches, they are used in different circumstances. Some considerations on this topic made in [20] are provided bellow.

Hash URIs have the advantage of reducing the number of necessary HTTP round-trips, which, in turn, reduces access latency. The downside of the hash URI approach is that the descriptions of all resources that share the same non-fragment URI part are always returned to the client together, irrespective of whether the client is interested in only one URI or all. If these descriptions consist of a large number of triples, the hash URI approach can lead to large amounts of data being unnecessarily transmitted to the client [20].

303 URIs, on the other hand, are very flexible because the redirection target can be configured separately for each resource. There could be one describing document for each resource, or one large document for all of them, or any combination in between. It is also possible to change the policy later on [20].

As a result of these factors, 303 URIs are often used to serve resource descriptions that are part of very large data sets, such as the description of an individual concept from DBpedia, an RDF-ized version of Wikipedia, consisting currently of 3.77 million entities, described by over 1.89 billion RDF triples [11], more information about DBpedia is presented in section 2.7.5.

Hash URIs are often used to identify terms within RDF vocabularies, as the definitions of RDF vocabularies are usually rather small, maybe a thousand RDF triples, and as it is also often convenient for client applications to retrieve the complete vocabulary definition at once, instead of having to look up every term separately. Hash URIs are also used when RDF is embedded into HTML pages using RDFa. Within the RDFa context, hash URIs are defined using the RDFa *about=* attribute. Using them ensures that the URI of the HTML document is not mixed up with the URIs of the resources described within this document.

A combination of the advantages of the 303 URI and the hash URI approach is also possible. By using URIs that follow a *http://domain/resource#this* pattern, for instance, *http://biglynx.co.uk/vocab/sme/Team#this*, the data that is returned as a description of a resource can be flexibly configured and the second HTTP request can still be avoided , as the *#this* part, which distinguished between the document and the described resource, is stripped off before the URI is dereferenced [13, 20].

More thorough and detailed materials on dereferencing URIs can be found in [14],[13], [20].

Once an URI is guaranteed to be dereferenceable, the content of its representation has to be reached (through HTTP), which can be located anywhere on a server on the Web. This is called resolving or resolution and in the current Internet infrastructure is caried out by the Domain Name System which is considered in a later section.

2.6.4. Quality of URIs

After considering the technical characteristics and mechanisms for dereferencing URIs it is important to pay attention to some aspects which are related to the quality and usability of URIs. These considerations are important to the proper and efficient functioning of the Web as well as the Linked Data initiatives. Bellow are several important points that concern the quality of an URI and its persistence for long time [3].

- URIs should not reflect the underlying file system where resources are stored. URI space should be abstract and well organized reflecting the conceptual structure of which a resource is a part.
- Similar rule applies to the underlying technology with the help of which a resource is exposed, the name of the technology should not appear in an URI, (e.g. pages produced by scripts having cgi in their URI).
- It is generally advised not to include information about author, subject, status (e.g. "old" or "new"), access target (e.g. team, public etc,), file name extension.

It should also be noted that the amount of information put into a URI plays important role. The more information there is in an URI, its longevity decreases, and if there is little information, the URI can't be dereferenced to a resource, it becomes just a plain identifier [29]. Last but not least a very important implication which has impact of the URIs is that one does not buy an URI but only rents it. This is one of the limitations of the Web and DNS, but it does not look like to be overcome any time soon.

2.7. Linked Data

Linked data is an initiative that strives to connect related data that was not previously linked, or using the Web to lower the barriers to linking data currently linked using other methods [22]. It is a set of best practices which are adopted by ever increasing number of data providers, leading to the creation of a global data space containing billions of assertions - the Web of Data [7]. Linked data is based upon standard Web technologies such as HTTP and URIs and has its best practices for exposing, sharing and connecting pieces of data, information, and knowledge on the Semantic Web. It embodies many of the above stated principles and tackles with some of the major problems, like the *information islands*.

2.7.1. Main Principles

The Linked Data movements has its own main principles, which again have certain similarities to what was explained above.

- **Use URIs as names for things.** Any resource of significance should be given a URI. (Or more generally an identifier) (What sorts of things can be resources? A very wide variety. The URI concept itself puts no limits on this. However, URIs are divided into schemes, such as http: and telenet:, and the specification of each scheme determines what sort of things can be resources in that scheme. Schemes are discussed later.) This means that no information which has any significance and persistence should be made available in a way that one cannot refer to it with a URI.
- **Use HTTP URIs so that people can look up (dereference) those names.** Dereferencing is explained in the next section.
- **When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)**
- **Include links (RDF statements) to other URIs. so that they can discover more things.**

2.7.2. Technology Stack

Linked data being an effort and a direction in the development of the Web and not a technological revolution, relies on two technologies that are fundamental and well known to the Web: Uniform Resource Identifiers (URIs) and the HyperText Transfer Protocol (HTTP).

The entities that are identified by URIs should use the *http://* scheme, as stated by the second principle, so they can be looked up simply by dereferencing the URI over the HTTP protocol. This way the HTTP protocol provides a simple and yet universal mechanism for retrieving resources that can be serialized as a stream of bytes (such as a photos of people, or any digital content), or retrieving descriptions of entities that cannot themselves be represented as a digital resource and sent across the network (such as the people themselves) [7].

URIs and HTTP are complemented by RDF, introduced above, which is critical to the Web of Data. In comparison to HTML which provides a means to structure and link documents on the Web, RDF provides a generic, graph-based data model with which to structure and link data that describes things from the real world.

As was explained earlier the RDF model encodes data in the form of *subject*, *predicate*, *object* triple statements. The subject and object of a triple are both URIs that each identify a resource, or a URI and a string literal respectively. The predicate specifies how the subject and the object are related, and is also represented by a URI [7].

The RDFS (RDF Schema), which is a vocabulary definition language based on RDF and the Web Ontology Language (OWL) provide a basis for creating vocabularies that can be used to describe entities in the world and how they are related. Vocabularies are collection of classes and properties and are expressed in RDF, using terms from RDFS and OWL, which provide varying degrees of expressivity in modelling domains of interest. By the free publishing of vocabularies to the Web of Data there exist plenty of sparse information which in turn can be connected by RDF triples that link classes and properties in one vocabulary to those in another, thereby defining mappings between related vocabularies [7].

An important prerequisite to utilize the numerous RDF vocabularies and datasets and their expressiveness is the capability of querying the RDF data. For this reason the World Wide Web Consortium (W3C)[5] has standardised the SPARQL query language for RDF data.

2.7.3. SPARQL

SPARQL is the de-facto standard query language for RDF. It can be used to express queries across diverse data sources in the form of retrieving and manipulations with stored data. As was explained above RDF data is based around the notion of triples, so in its simple essence SPARQL queries consist of triple patterns (required and optional), which can have the logical operation *conjunction* and *disjunction* added to them. The result of SPARQL queries can be results set or RDF graphs.

A simple SPARQL query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The graph pattern may consist of a single triple pattern with a single or multiple variables, or multiple patterns matching actual resources (identified by URIs) or literal values (strings, integers, etc.) Bellow is an example of a query with multiple matches and multiple solutions provided by W3C [37].

```

Data:

@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .


Query:

PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }

Query Result:

       name                        mbox
"Johnny Lee Outlaw"    <mailto:jlow@example.com>
"Peter Goodguy"        <mailto:peter@example.org>
```

Listing 2.1: Query with multiple patterns and mutlipe matches.

There are four query forms, which use the solutions from the pattern matching to form result sets or RDF graphs [37]:

[5]http://www.w3.org/ – the main international standard organization for the World Wide Web

- SELECT returns all, or a subset of, the raw values bound in a query pattern match in a result set form
- CONSTRUCT returns an RDF graph constructed by substituting variables in a set of triple templates.
- ASK returns a boolean true/false indicating whether a query pattern matches or not.
- DESCRIBE returns an RDF graph that describes the resources found.

There is also a SPARQL Update language for RDF graphs which is complementary to the SPARQL Query language, and utilizes its syntax constructs. The SPARQL Update is not yet endorsed as a standard by the W3C and is still in progress (a working draft) until it reaches a final status as official recommendation. The SPARQL Update provides INSERT, UPDATE and DELETE capabilities. More information is provided in the W3C working draft [38].

2.7.4. Main Efforts

The most well known embodiment and application of the Linked Data principles is the Linking Open Data[6] project, a grassroots effort, started in the beginning of 2007, supported by the W3C Semantic Web Education and Outreach Group. The original and ongoing aim of the project is to bootstrap the Web of Data by identifying existing data sets that are available under open license, converting these to RDF according to the Linked Data principles, and publishing them on the Web.

In its early stages the project was mainly supported by researchers and developers in universities and small companies. With time the project has gathered considerable attention from the public and many large organizations have been involved, such as the BBC, Reuters, the British government, the Library of Congress, the French National Library (Bibliothèque nationale de France), etc. This groth is made possible by the open nature of the project, where anyone can participate simply by publishing a data set according to the Linked Data principles and interlinking it with existing data sets. To give an idea for the scale and range of the Web of Data resulting from the Linking Open Data project Figure 2.1. is provided. Each node in this diagram represents a distinct data set published as of September 2011, with the arrows representing the level of connection between data sets.

The content of the Web of Data cloud is diverse in nature, comprising data about domains such as: geographic locations (Linked GeoData), companies (IBM), people, books, scientific publications (ACM), films (LinkedMDB), music television and radio programmes, genes, proteins, drugs and clinical trials, online communities, statistical data, census results, etc [7].

The size of the nodes on Figure 2.1. are proportional to the number of triples that each dataset contains. The arcs between the nodes indicate the links that exist between items in the connected data sets. Heavier arcs roughly correspond to a greater number of links between two data sets and the direction of the links indicate the outward links from one data set to another.

[6]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

The exact number of triples in the Web of Data is difficult to estimate, due to the fact that much of the data is being generated by wrappers around existing relational databases or APIs and therefore first need to be crawled before it can be counted or analyzed [1]. One possibility for calculating the size of the Web of Data can be the estimation of the size of each of the participating data sets, which is provided by the Linking Open Data community [24]. According to these statistics, the Web of Data contains around 19.5 billion RDF triples.

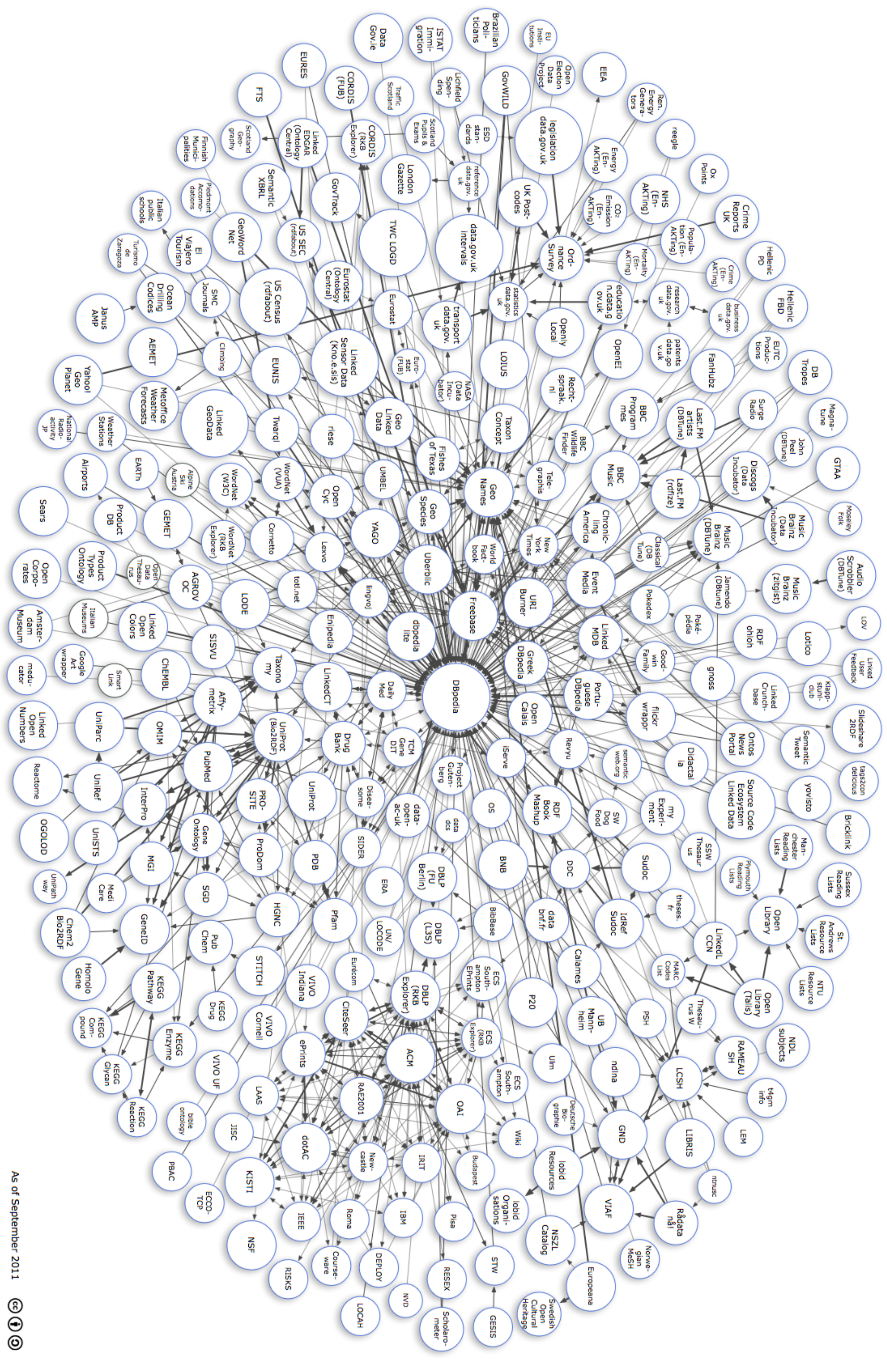


Figure 2.1.: Graph representing the data sets building up Linked Open Data together with their relationships as of September 2011 [23].

Domain	**Data Sets**	**Triples**	**Percent**	**RDF Links**	**Percent**
Cross-domain	41	4,184,635,715	13.23	63,183,065	12.54
Geographic	31	6,145,532,484	19.43	35,812,328	7.11
Government	49	13,315,009,400	42.09	19,343,519	3.84
Media	25	1,841,852,061	5.82	50,440,705	10.01
Publications	87	2,950,720,693	9.33	139,925,218	27.76
Life sciences	41	3,036,336,004	9.60	191,844,090	38.06
User-generated content	20	134,127,413	0.42	3,449,143	0.68
	295	31,634,213,770		503,998,829	

Table 2.1.: Number of data sets, amount of triples, and amount of RDF links per topical domain, as of October 2011 [39].

2.7.5. Popular Datasets

The data sets that form the Web of Data are classified into several topical domains which are listed in table Table 2.1. The number of triples as well as their links per domain are also shown in the table. The number of RDF links refers to out-going links that are set from data sources within a domain to other data sources [20]. Some very thorough summary statistics about the data sets that are catalogued within the Linking Open Data Catalog are provided by *State of the LOD Cloud* document [39].

The central interlinking hub of the Web of Data as seen on Figure 2.1 is DBpedia. It is a community effort to convert the human readable Internet encyclopedia Wikipedia[7] into structured information accessible for the Web of Data. It describes around 3.77 million entities, out of which 2.35 million are classified in a consistent Ontology, including 764,000 persons, 573,000 places (including 387,000 populated places), 333,000 creative works (including 112,000 music albums, 72,000 films and 18,000 video games), 192,000 organizations (including 45,000 companies and 42,000 educational institutions), 202,000 species and 5,500 diseases [11].

The central point of DBpedia in the Linked Data cloud is due to several factors, among which are its cross-domain nature , its constant up-to-date status and its community agreement, guaranteed by its dependence on Wikipedia, and its rich language base. Because it contains terms from many different domains it is the “glue” in the Web of Data which is used by many domain specific datasets. Some of the domain oriented datasets are LinkedGeoData – a conversion from the OpenStreetMap project, which provides information about more than 1 billion nodes and 100 million ways and the resulting RDF data comprises approximately 20 billion triples; the datasets provided by the British Broadcasting Corporation (BBC) on topics such as music, wild life, sport; government datasets provided by Australia, New Zealand, U.K. and USA.

2.7.6. Tools and Applications for Linked Data

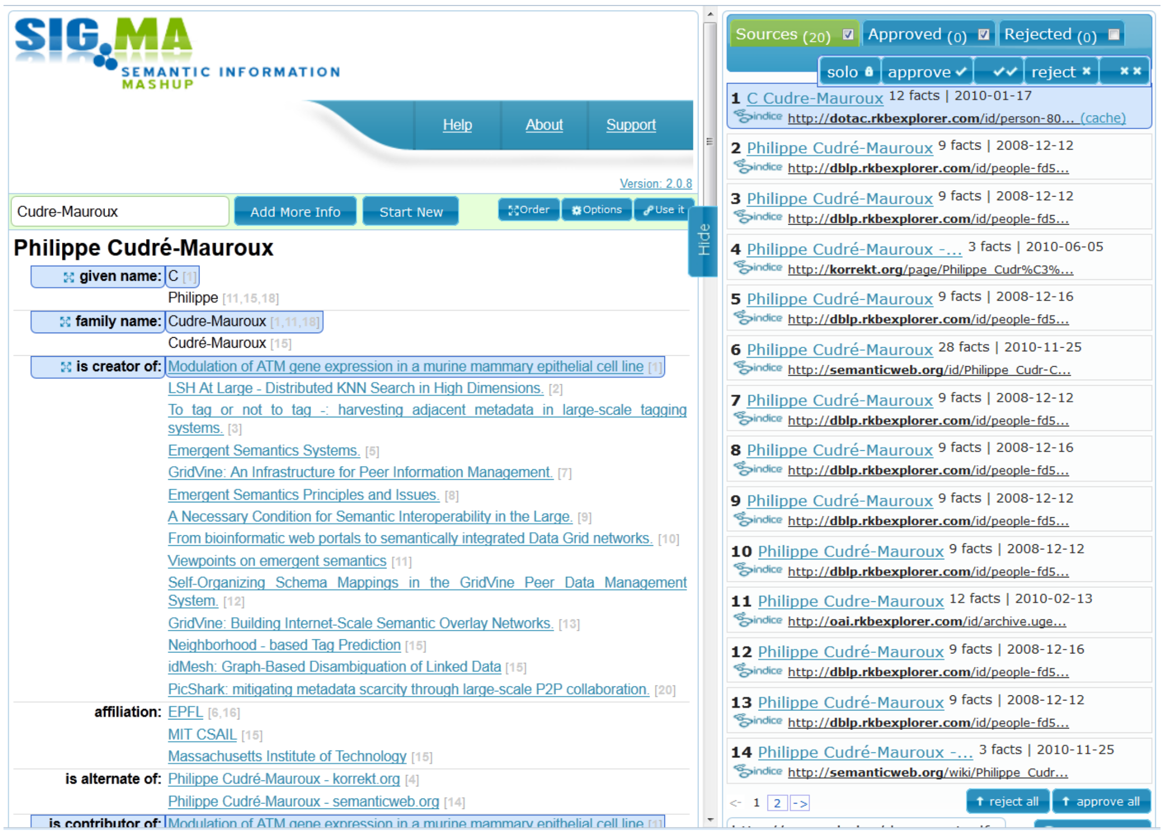
There are a variety of tools and applications that support the publishing and consumption of Linked Data. The tools may support the publishing of RDF content as Linked Data

[7]http://en.wikipedia.org/wiki/Wikipedia

on the Web or provide conversion of non-RDF data with making transparent some of the technical details such as content negotiation and ensuring the adherence to the Linked Data community best practices.

All tools support dereferencing URIs into RDF descriptions. In addition, some of the tools also provide SPARQL query access to the served data sets and support the publication of RDF dumps. Some of the more popular publishing tools are: D2R Server - tool for publishing non-RDF data as Linked Data with the help of declarative mapping; Virtuoso Universal Server - provides serving for RDF data with SPARQL interface; AllegroGraph - provides storing and serving of RDF data, querying and update capabilities, integration with clients for different programming platforms.

Another indivisible part of the Linked Data application stack are the Linked Data browsers and search engines. The browsers allow users to navigate in-between statements about a certain resource and to explore the different data sources that provide relevant information, following links expressed as RDF triples. Linked Data search engines similar to web search engines serve as the place where navigation process - browsing begins. The Linked Data search engines crawl Linked Data from the Web by following RDF links and provide query capabilities over aggregated data. Figure 2.2. provides an example of the Sig.ma search and browsing functionalities developed by the Digital Enterprise Research Institute (DERI).



Cudre-Mauroux

Philippe Cudré-Mauroux

given name: C [1]
Philippe [11,15,18]

family name: Cudre-Mauroux [1,11,18]
Cudré-Mauroux [15]

is creator of: Modulation of ATM gene expression in a murine mammary epithelial cell line [1]
LSH At Large - Distributed KNN Search in High Dimensions. [2]
To tag or not to tag -: harvesting adjacent metadata in large-scale tagging systems. [3]
Emergent Semantics Systems. [5]
GridVine: An Infrastructure for Peer Information Management. [7]
Emergent Semantics Principles and Issues. [8]
A Necessary Condition for Semantic Interoperability in the Large. [9]
From bioinformatic web portals to semantically integrated Data Grid networks. [10]
Viewpoints on emergent semantics [11]
Self-Organizing Schema Mappings in the GridVine Peer Data Management System. [12]
GridVine: Building Internet-Scale Semantic Overlay Networks. [13]
Neighborhood - based Tag Prediction [15]
idMesh: Graph-Based Disambiguation of Linked Data [15]
PicShark: mitigating metadata scarcity through large-scale P2P collaboration. [20]

affiliation: EPFL [6,16]
MIT CSAIL [15]
Massachusetts Institute of Technology [15]

is alternate of: Philippe Cudré-Mauroux - korrekt.org [4]
Philippe Cudré-Mauroux - semanticweb.org [14]

is contributor of: Modulation of ATM gene expression in a murine mammary epithelial cell line [1]

1 C Cudre-Mauroux 12 facts | 2010-01-17
indice http://dotac.rkbexplorer.com/id/person-80... (cache)

2 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

3 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

4 Philippe Cudré-Mauroux -... 3 facts | 2010-06-05
indice http://korrekt.org/page/Philippe_Cudr%C3%...

5 Philippe Cudré-Mauroux 9 facts | 2008-12-16
indice http://dblp.rkbexplorer.com/id/people-fd5...

6 Philippe Cudré-Mauroux 28 facts | 2010-11-25
indice http://semanticweb.org/id/Philippe_Cudr-C...

7 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

8 Philippe Cudré-Mauroux 9 facts | 2008-12-16
indice http://dblp.rkbexplorer.com/id/people-fd5...

9 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

10 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

11 Philippe Cudre-Mauroux 12 facts | 2010-02-13
indice http://oai.rkbexplorer.com/id/archive.uge...

12 Philippe Cudré-Mauroux 9 facts | 2008-12-16
indice http://dblp.rkbexplorer.com/id/people-fd5...

13 Philippe Cudré-Mauroux 9 facts | 2008-12-12
indice http://dblp.rkbexplorer.com/id/people-fd5...

14 Philippe Cudré-Mauroux -... 3 facts | 2010-11-25
indice http://semanticweb.org/wiki/Philippe_Cudr...

Figure 2.2.: Sig.ma Linked Data search results for Prof. Philippe Cudre-Mauroux.

2.8. SPARQL Endpoints

Linked Data browsers and search engines are important for the realization of the capabilities of well organized, interlinked Semantic Web data by humans, but as was stated above, Semantic Web is about machine readability and consumption of information. The way machines interact with the information is through interfaces and APIs, where querying data from RDF datasets and receiving results from different kinds of Linked Data applications plays an important role. The usual way queries are executed over exposed datasets is through SPARQL endpoints.

Linked Data sources usually provide a SPARQL endpoint for their dataset(s). In most cases this is a query processing service that supports the SPARQL query language which was presented above. SPARQL queries are usually sent as an HTTP GET request to the SPARQL endpoint with parameter the query. An example is shown in listing Listing 2.2. The endpoints act like RESTful Web services and can produce results in different formats: XML, JSON, plain text, RDF/XML, NTriples, OWL, N3 etc. The ACCEPT HTTP header can be used for indication of the preferred result format. It is usually the task of the calling software to process the returned response into a meaningful presentation for human users or for further consumption by machine entities. An exhaustive list of SPARQL Endpoints on different data sets is provided by the W3C under http://www.w3.org/wiki/SparqlEndpoints.

```
http://dbpedia.org/sparql?default-graph-uri=http://dbpedia.org&query=select+distinct+?Concept+
    where+{[]+a+?Concept}+LIMIT+100&format=text/html&timeout=0&debug=on
```

Listing 2.2: SPARQL Query request on the dbpedia SPARQL endpoint.

There are also a number of libraries for accessing SPARQL Endpoints, written in different languages. To name a few: Jena for Java, which is also used for the benchmarking in chapter 4, Sesame also for Java, SPARQL Wrapper (Python), SPARQL JavaScript Library. The libraries provide utility methods for addressing an endpoint, creating a query, executing a query, receiving the result in an useful form, etc.

The simplest queries on a SPARQL endpoint are over single data set. This is important but does not really introduce an improvement for the Semantic Web compared to for example what a relational data base could do for a data set with exposed interface for SQL queries. SPARQL endpoints provide some compelling advantages for the Linked Data. Issuing queries over multiple datasets; using logical constructs as conjunction and disjunction, inference and reasoning are some of the important features that distinguish SPARQL endpoints that implement them.

As explained above SPARQL query language is based around triple patterns, which are in the form of subject predicate object. When there are multiple patterns are superimposed they form a conjunctive query. With the usage of the UNION construct in SPARQL, disjunction is achieved.

Inference and reasoning are based on the features of RDFS and OWL. OWL is divided into three sublanguages which provide different level of expressiveness: OWL Lite, OWL DL and OWL Full ordered from the least to most powerful. Based on the different logical constructs provided by the OWL subtypes and RDFS further conclusions are possible to be made. Some of the reasoning capabilities are:

- Type inheritance through rdfs:subclassOf

- Transitivity and reflexsivity through rdfs:subClassOf, rdfs:subPropertyOf
- The semantics of owl:inverseOf, which imply that e.g. if parentOf owl:inverseOf hasParent and Peter parentOf Sophie then Sophie hasParent Peter
- When an owl:sameAs relationship is asserted between two entities that are known to be classes, an owl:equivalentClass relationship is inferred between the classes

More on the reasoning and inference capabilities of RDFS and OWL can be found in [35, 33, 34]. Not all specification and reasoning capabilities are usually implemented in SPARQL endpoints. Some of the capabilities that reasoning offers are in fact powerful but may lead to having no computational guarantees as is the case with OWL Full. Reasoning deteriorated the execution times for queries as is shown later in this work

Queries over multiple data sets can be achieved by copying the data to a single data store, but this could not scale and may imply the usage of out-dated data. Another solutions are federated queries. When there are multiple data sets each providing SPARQL endpoint, federated queries can be executed using the help of mediator which distributes subqueries to the sources and then integrates the result. The queries may be in normal form and the federation can be done transparently as is the case by AllegroGraph. There is also a standard defined by W3C for federated queries. The SERVICE keyword is used to instructs a federated query processor to invoke a portion of a SPARQL query against a remote SPARQL endpoint. Again not all SPARQL endpoints provide federated querying capabilities and the standard for federated queries is relatively new and is still in draft version [36].

3
State of the Art of Entity Registry Solutions

There are several architectural approaches for building digital repositories. One main differentiation is between centralized systems like the DNS[1], or DOA[2], and peer-to-peer systems like Chord and Coral.

The centralized model divides the nodes from a network into main and subordinate nodes, into a hierarchy. In terms of network such system is connected using star or tree network topology, where the subordinate nodes connect to the main nodes/node when they have to execute a task for which they don't have the required authority or information. Generally speaking centralized approaches have always a more strict administrative structure.

Peer-to-peer systems are distributed in a network, generally without any centralized control or hierarchical organization. Each node part of a P2P[3] network runs software with equivalent functions. P2P applications could provide many functionalities, among which are: redundant storage, permanence, selection of nearby servers, authentication, anonymity, and hierarchical naming. Despite this rich set of features, in essence most of the P2P applications boil down to distributed entity repositories, which core operation is the efficient location of data items [40].

[1]Domain Name System
[2]Digital Object Architecture
[3]peer-to-peer

3.1. DNS

The Domain Name System (DNS) is a hierarchical distributed naming system providing service to any resource connected to the Internet or a private network [43]. It associates various information with domain names assigned to each of the participating entities from the network. A Domain Name Service resolves queries for these domain names into IP addresses for the purpose of locating computer services and devices. By providing a worldwide, distributed keyword-based redirection service, the Domain Name System is considered as a backbone for enabling end-points to communicate easily with each other without being concerned with the other's actual location (identified by IP-address). DNS is essential component of the functionality of the Internet and the Web. With its services it allows for the transparent movement of service providers anywhere in the world, while maintaining the required seamless connectivity to their clients.

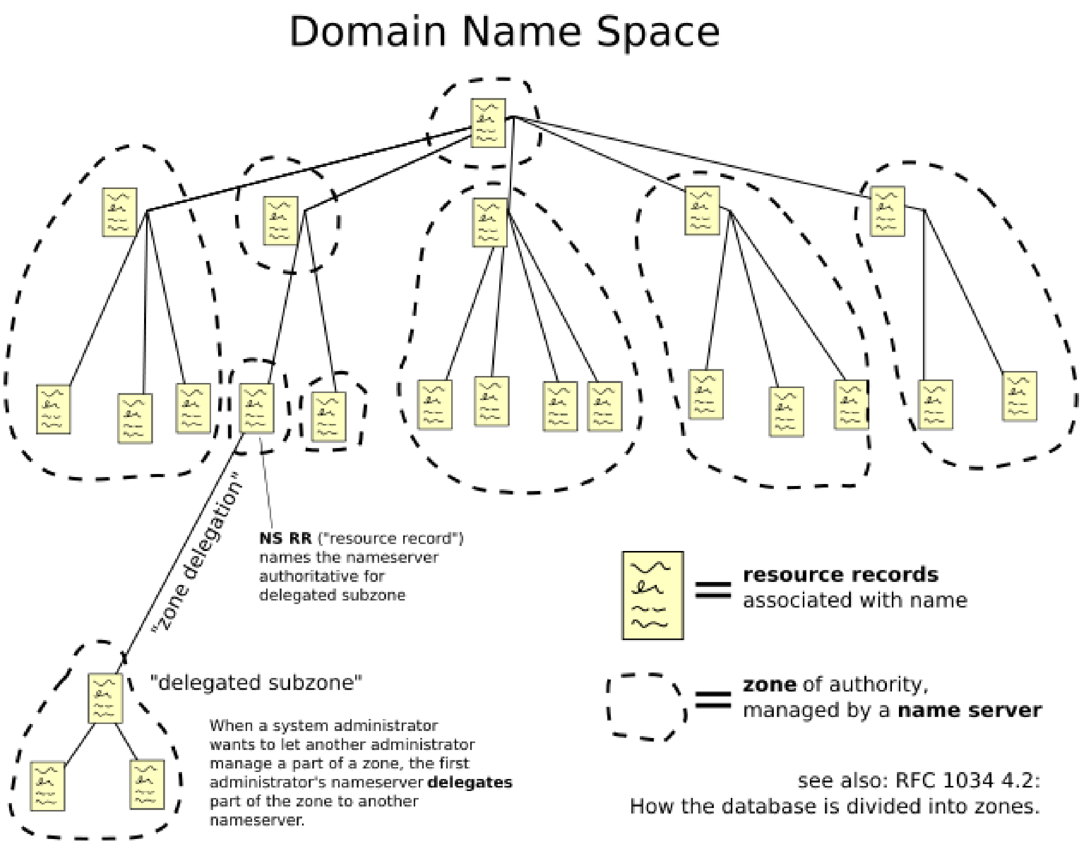

Domain Name Space
"zone delegation"
NS RR ("resource record") names the nameserver authoritative for delegated subzone
"delegated subzone"
When a system administrator wants to let another administrator manage a part of a zone, the first administrator's nameserver delegates part of the zone to another nameserver.
= resource records associated with name
= zone of authority, managed by a name server
see also: RFC 1034 4.2: How the database is divided into zones.

Figure 3.1.: The hierarchical Domain Name System, organized into zones, each served by a name server [43].

The DNS has three major components [27]:

- The *Domain Name Space* and Resource record, which are specifications for a tree structured name space and data associated with the names. Each node or leaf in the tree has zero or more resource records, which hold information associated with the domain name. The tree sub-divides into zones beginning at the root zone

Figure 3.1. A DNS zone may consist of only one domain, or may consist of many domains and sub-domains, depending on the administrative authority delegated to the manager [43]. Administrative responsibility over any zone may be divided by creating additional zones. Authority is said to be delegated for a portion of the old space, usually in the form of sub-domains, to another name server and administrative entity. The old zone ceases to be authoritative for the new zone [43].

- *Name Server* is a server that stores the DNS records. A name server may cache structure or set information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can be used to lead to information from any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information. Such servers are called *authoritative servers* for these parts of the name space. Authoritative information is organized into zones, and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone [27].
- DNS *resolver* is the client-side of the DNS. It is responsible for extracting information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly - that is *non-recursive query*, or pursue the query using referrals to other name servers - that is a *recursive query* [27].

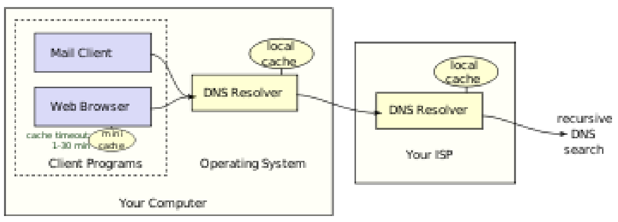


Figure 3.2.: DNS resolution sequence [43].

Figure 3.2. shows the resolution sequence by client lookup. The user would not normally communicate directly with a DNS resolver. Instead DNS resolution should happen transparently along the way, where the requested domain name would be cached, in the web browser, or the operating system or the local internet service provider (ISP). In the event that the name is not resolved the query goes to a higher leve DNS server until the query is successfully resolved or no result is returned.

In general DNS provides very fast responses as most of the time the queried domain names are cached somewhere along the way. Naturally appears the question if the DNS would be suitable for a general purpose resource naming system. The issue is that DNS is originally designed and primarily used for mapping domain names into IP Addresses to support network routing. Because of that any other use along with various DNS extensions have the potential to slow down the network address translation and affect DNS effectiveness in network routing. Even considering a separate DNS based resource naming system (which is in effect a key – value distributed storage) that could work independently of the established DNS infrastructure, this would not provide the desired results. DNS implementations do not scale well when large amount of data is associated with any particular DNS name [41, p. 13]. Another issue that prevents the usage of DNS as a general purpose resource naming service is the DNS administrative mode. DNS names are managed tipically by the network administrator at the DNS zone level. There is no provision for a per-name administrative structure. No facilities are provided other than network administrators to create or manage DNS names. This is appropriate in the case of domain name administration, but less so for general-purpose name administration.

3.2. DOA

The Digital Object Architecture (DOA) is an effort by the Corporation for National Research Initiatives (CNRI) that strives to utilize the capabilities of the Internet for the means of information management. It was designed to enable all kinds of information: public, private or combination of both in the form of digital entities, to be managed in a network over potentially very long time frames. The DOA takes into account such aspects of Information Management as identification, storage, resolution, discovery, matching, interoperability, security. Generally speaking, any information expressed in digital form can be managed within the architecture. There are three distinct components in the DOA for which there are implementations provided by CNRI (Corporation of National Research Initiative) which came up with DOA. The components are as follows:

- Resolution System (Handle System)
- Digital Object Repository (DORepository)
- Digital Object Registry (DORegistry)

3.2.1. Handle System

The principal function of the Handle System is to map known identifiers into handle records, containing useful information about the digital object being identified (e.g. IP address, public key, URL etc.). Every identifier has two parts: a naming authority (a.k.a. prefix) and a unique local name under the naming authority – suffix, separated by "/" (e.g. "10.1045/january99-bearman").

The collection of all local names defined under a certain prefix defines the local handle namespace under that prefix (something like a root zone in the case of DNS). All the local namespaces (all prefixes) define the handle namespace and a prefix can be considered as a top level domain. More namespaces for Local Handle Services (LHS) can be defined in hierarchical fashion under the Global Handle Registry (GHR), thus the handle system provides hierarchical service model. The separate LHS can work autonomously with the handles from their namespaces. The structure of the Handle system is shown on Figure 3.3.

The Handle System provides also distributed architecture. It consists of a number of individual handles services. Each of these services consists of service sites. Each service site replicates the other and can resolve all their handles. Each site may consist of one or more handle server. There are no limitations on the number of services, service sites or servers, which provides scalability and eliminates single point of failure.

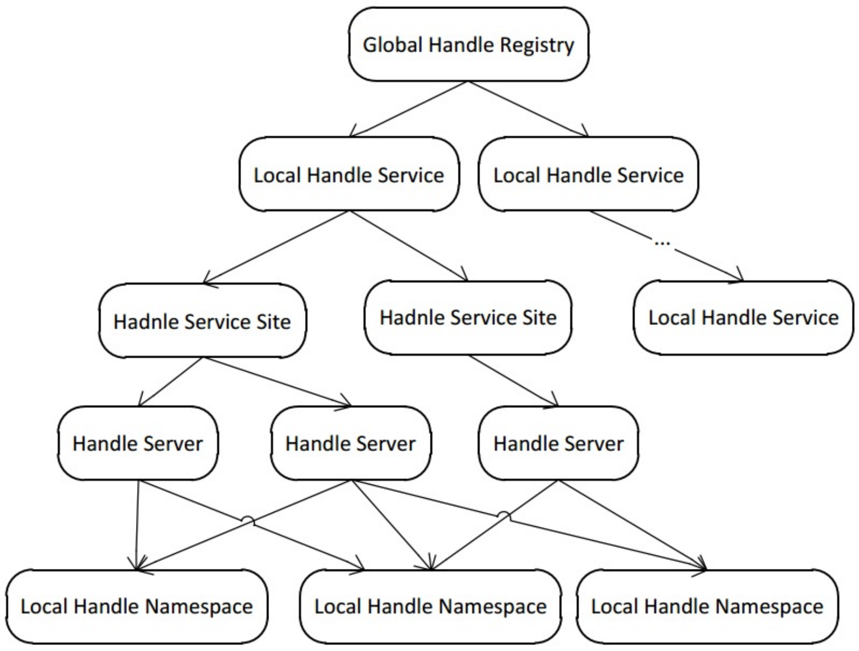


Figure 3.3.: Handle System structure.

3.2.2. DORepository

The DORepository stores digital objects and provides access to them. There are no limitations on number of repositories and size. It can work with standard storage products. Each digital object that is stored in the repository is assigned a unique persistent identifier that is registered by the Handle System. The DORepository provides the capabilities of moving objects from one storage system to another and even between different repositories only by few instructions from an administrator, while keeping all metadata and adding provenance information. (Unlike by DNS when web pages change domain names or name structure, and links become invalid, for which there is no information to the end user). The DORegistry provides services like browsing, searching, repository and federation for collections of digital objects that can be distributed across multiple sites including other DO Registries. A DO registry may manage metadata of objects from a certain repository. Another possibility is managing both metadata and actual digital object content stored by the registry, and a third scenario is managing metadata of multiple repositories.

3.2.3. DORegistry

The DO Registry can be set for different types of metadata schemata and can be customized to provide different search, federation, handle registration, event management and other services. It can be considered as yellow pages for finding digital objects across a network. The DOA relies on its own standards of identifiers etc. so it requires certain amount of integration for already existing data to utilise the capabilities of DOA.

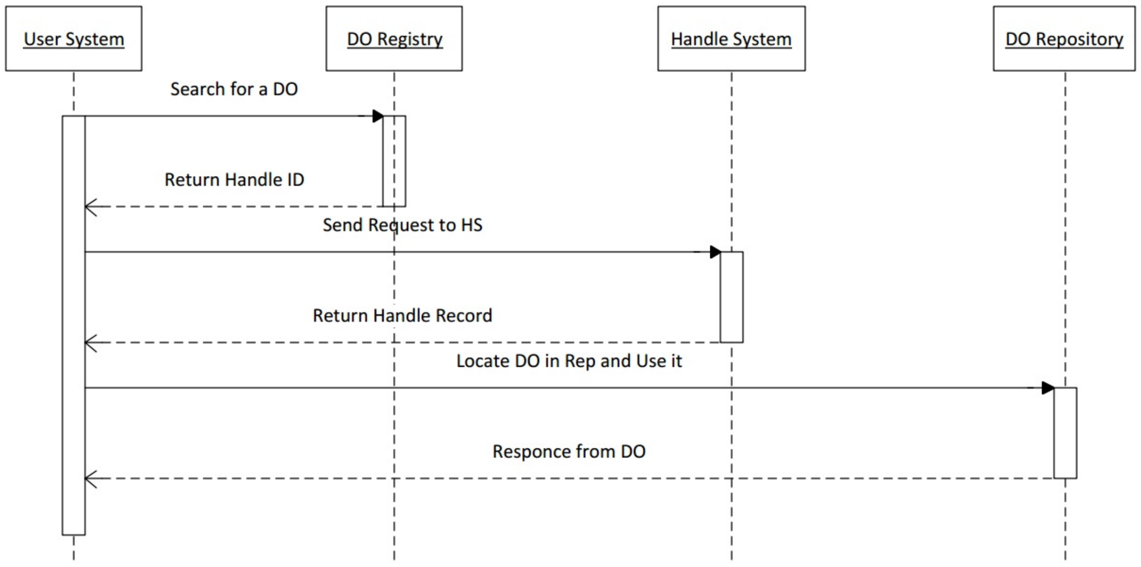


Figure 3.4.: Interaction Diagram on Digital Object search.

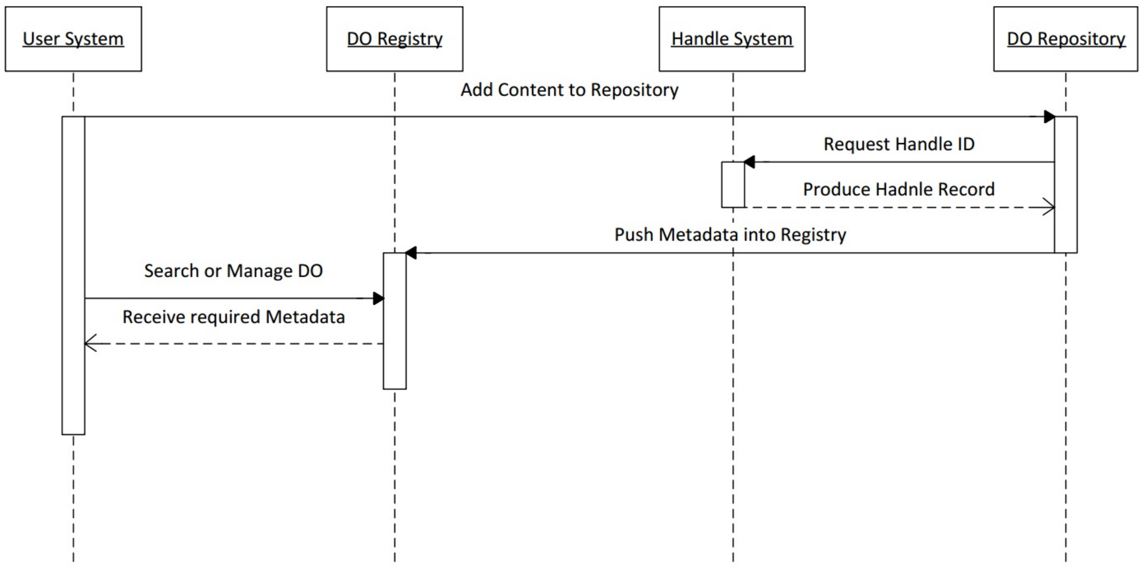


Figure 3.5.: Interaction Diagram on adding a digital object.

To visualize better the interactions between the three DOA components, there are three diagrams showing different operations on digital objects. Figure 3.4. shows digital object search. Figure 3.5. shows how an object is added directly to the DORepository and then automatically added to the Registry. Figure 3.6. shows the registration process for a digital object.

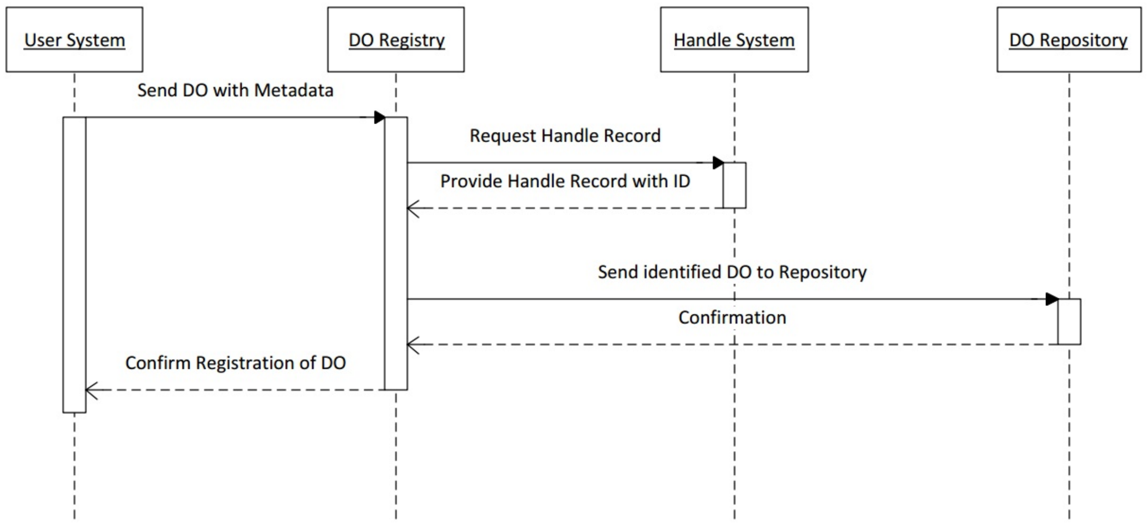


Figure 3.6.: Interaction Diagram showing registration of a DO.

3.3. ENS

The Entity Name System (ENS) addresses problems of already existing information collections. Whereas by DOA, a unique identifier is created for each entity, in the case of ENS the reuse of identifiers is aimed. The ENS tackles ad-hoc and on demand information tasks. The ENS is a web-scale infrastructure for supporting the reuse of pre-existing URIs for any type of entity across decentralized and independent RDF repositories [8].

As discussed in 2.5 the integration of separate knowledge bases can be realized when one and the same entity can be found in their knowledge graphs, thus connecting these graphs. Two main problems related to this idea are: (*i*) heterogeneity of vocabulary - the same concept or property may be referred through different URIs, and therefore may not be recognized as the same concept or property in two different vocabularies; (*ii*) entity recognition - the same real world object may be assigned different URIs in different RDF repositories, and therefore may not be recognized as the same entity. The OKKAM project is developed to address these problems[8]. The authors propose ENS - a universal repository which given any arbitrary representation of an entity (e.g. a bag of key-words, text paragraph, collection of key-value pairs, etc.), can decide if an URI for this entity is already available in the repository (using some methods for entity matching); if it is, then the ENS will return its URI (or at least a list of candidates), otherwise it will issue a new URI which will be stored in the repository.

3.4. Chord

Chord is a lookup protocol and algorithm for a peer-to-peer distributed hash table. It is designed for efficient location of nodes that store desired data items in a distributed network. From a high functional level point of view Chord supports only one operation – it maps a given key to a node from the network. Based on this operation the two most basic functions are insertion and retrieval of data items associated with a key. Other operations that support the efficiency and adaptiveness of chord are left transparent for the end user systems [40].

The core of Chord is the fast distributed computation of a hash function, which maps keys to nodes responsible for them. Chord assigns m-bit identifiers to at most 2^m keys and nodes using *consistent hashing* [21]. The consistent hashing balances load in the network - all nodes receive roughly the same number of keys. Also thanks to it when one node joins or leaves a network of N nodes this causes only $O(1/N)$ fraction of the keys to be moved to a different location which is minimal. With the help of a small amount of routing information about other nodes and communicating with other nodes, Chord improves scalability of consistent hashing by avoiding the requirement that every node should know about every other node, they only keep a so called "finger table". This way each node knows about only $O(logN)$ other nodes, and a look up requires $O(logN)$ messages [40].

The nodes are arranged in a circle that has at most 2^m nodes.Every node has a finger table of m entries which contains the identity of the first node s that succeeds the node at n by at least $2^{(i-1)}$ on the identifier circle, i.e. $s = successor(n + 2^{(i-1)})$, where $1 <= i <= m$ (all arithmetic is modulu 2^m). The finger table provides information about nodes across all the network with morefingers pointing to closer neighbours. Since the finger entries are with the power of two, each node can forward a query at least half way through the network, which facilitates the efficient node location.

By failure of nodes, Chord must ensure that each node's successor pointer is up to date. It does this using a "stabilization" protocol that each node runs periodically in the background and which updates Chord's finger tables and successors pointers. When node n runs the stabilization protocol it asks its successor for the successor's predecessor p, and decides whether p should be the new n's successor. This would be the case if p is newly joined. Additionally this procedure notifies the n's successor of n's existence, giving the successor the chance to change its predecessor to n.

The successors list mechanism can also be used to help higher-layer software to replicate data across nodes. An application using Chord might store replicas of data associated with a key at the k nodes succeeding the key. The fact that a Chord node keeps track of its successors means that it can inform the replicating software when successors come and go, so new replicas are prompted [40].

An implementation of Chord - Open Chord is tested and reviewed in the next chapter, where its adaptability is put into perspective next to other key – value solutions.

3.5. Coral

Coral is a peer-to-peer distributed content distribution system [19]. It provides scalability and load balancing which can be improved by taking into account locality of nodes. Coral utilizes the mechanism of distributed sloppy hash table (DSHT), which bears similarities to DHT, but has its differences. While a DHT provides a key-value mapping where only one value can be stored under a key, DSHT may have multiple values, and when the values for a key is retrieved it needs only return some subset of the values stored. Each node in Coral stores only some maximum number of values for a particular key. When the number of values exceeds the maximum, they are spread across multiple nodes. This characteristics of DSHT are specifically suited to locating replicated resources, where consistency is sacrificed over frequent fetches and stores of the same hash table key.

If this characteristic is considered in the context of RDF data, which is normally not replicated but scattered across multiple resources and reasoning on it requires consistent view of the whole, Coral is not suitable. It could be used by time critical applications which reason about constant changing datasets, which are out of the scope of this work.

4 Hands-on Experience with Entity Storage Systems and Benchmarking

4.1. Introduction

The practical part of this work is focused on benchmarking the performance and characteristics of four solutions which could facilitate the adoption of the Semantic Web in one way or another.

All four solutions are fundamentally different and most of the tests are aimed at emphasizing and utilizing these differences for showing what the systems do best and worst. Also because of the differences there are certainly inequalities by the tunings and configurations. The aim was to have as close as possible configuration to the original one. Where possible the storage solutions have been tested only the first time after loading (i.e. with cold cache). Still there might be inequalities due to the way data is organized in the different solutions because every system stores the data in a different way which is explained in the next section.

4.2. Dataset Transformations and Queries

The type of data that is chosen for benchmarking the four storage solutions is RDF graph in N-Triples format. N-Triples is one of the most widely used serialization formats for RDF data. It is line-based, plain text, does not provide shortcut mechanisms for URIs[1] and is very easy for parsing.

```
<http://data.linkedmdb.org/resource/actor/10000>
<http://data.linkedmdb.org/resource/movie/actor_name> "Lucien
Littlefield"^^<http://www.w3.org/2001/XMLSchema#String> .


<http://data.linkedmdb.org/resource/actor/10000>
<http://data.linkedmdb.org/resource/movie/performance>
<http://data.linkedmdb.org/resource/performance/57760> .
```

Listing 4.1: N-Triples format example.

[1]Uniform Resource Identifiers

The dataset has been transformed in key – value pairs in order to be usable with the registry systems that are the focus of this work. As is shown in the GridVine [10], a key – value storage is enough for query processing provided by most RDF query languages. Still the question on how exactly should the data be organized in a key–value or other non RDF-storage is not trivial as is also shown in CumulusRDF [25].

A common way through which RDF data is queried is with SPARQL Queries. They are based on triple patterns [2]. In total there are eight possible patterns for RDF triples: $(spo), (sp?), (?po), (s?o), (?p?), (s??), (??o), (???)$. The data in a key value store have to be arranged in a different way depending on which of the patterns is aimed. This depends on the use case and the nature of the data. In the general case, in order to make the RDF graph browsable one should be able to locate a node and to trace the edges that go out of the node and possibly into the node. As a triple pattern this translates into the union of $(s??)$ and $(??o)$ on or a query for all the triples where an entity is either a subject or an object. This case and the simpler one, where the entity is present only as a subject $(s??)$, are chosen for storing the data in the key – value stores. From here under the term *value* would be considered the simpler case, if not stated otherwise. Listing 4.2 shows an example of the union query for a professor who has an ID $prof84805$.

```
PREFIX bowl:<http://www.unifr.ch/bowlogna.owl#>

select ?s ?p ?o {{bowl:prof84805 ?p ?o.} UNION {?s ?p bowl:prof84805}}
```

Listing 4.2: Simple SPARQL entity query.

In Chord a key is an entity - URI, and a value is comprised in the first case of all the triples where this URI is a subject or object, and in the second case of all the triples where the URI is only subject.

In the case of Cassandra there are supercolumns and keys can be nested, so there is more flexibility on how to organize the data. Listing 4.3 shows how the triples are organized.

```
{ row key : { supercolumn key : { column key : value }}}
{ s : { p : { o : - }}}
```

Listing 4.3: Cassandra storage model, and configuration for storing triples.

In the case of MySQL, the triples were ordered in three columns of one table, which correspond to subjects, predicates and objects.

4.3. Datasets

Three datasets were used for testing. The first one is derived from real academic data of one of the faculties of the University of Fribourg, provided by the university administration. It will further be called *Faculty data set.* It is a part of the whole faculty data, including information about Teaching Units, Professors, Departments and Field of Studies across a period from 2003 until spring semester 2012. The dataset is comparatively small, consisting of 155 035 statements but it also has some interesting properties that are difficult to imitate with an artificially generated dataset, or one that describes a simpler more static domain.

[2]Triple patterns is an RDF triple written as a whitespace-separated list of a subject, predicate and object, where any of the three can be a variable (prefixed with "?") or an RDF term.

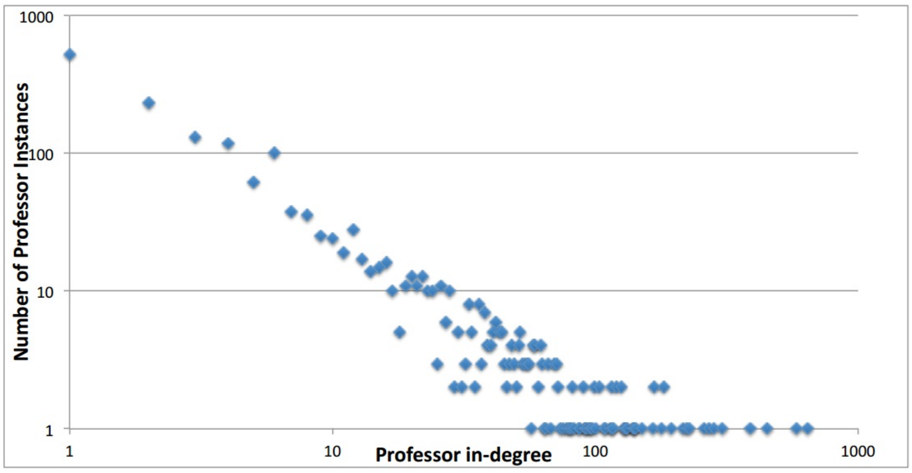


Figure 4.1.: In-degree of Professor instances [12].

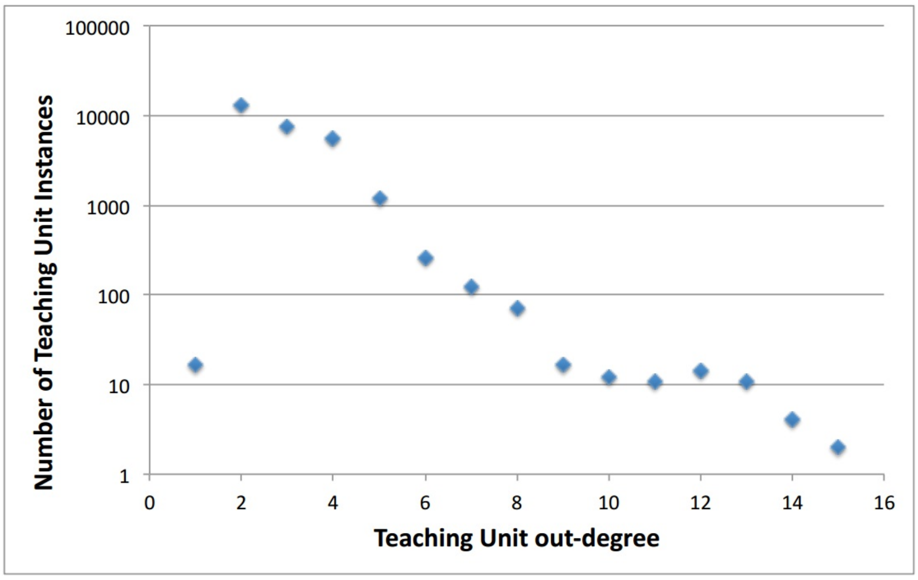


Figure 4.2.: Out-degree of Teaching Unit instances [12].

Figure 4.1. shows the distribution of professors in-degree which is mostly determined by the courses (teaching units) that they give, because the property *isTaughtByProfessor* is pointed from a teaching unit to professor. The chart shows a power law distribution, where can be seen that many professors teach only one or few courses, while few professors teach many courses (the biggest number of units per professor is 643). In the case this was found out to be due to the practice of assigning courses to administratively responsible persons, which is counterintuitive but technically not an error. Figure 4.2. shows the out-degree of teaching units. Again it is not evenly distributed, due to the fact that some courses have been given by more than one professors for longer periods of time (association with many semesters). This characteristics results in an inequality of the size of the entities, where the average number of triples per entity is around 6, but in reality there are entities with up to 2300 triples. This causes great deviations in the measurements.

Unlike the real *Faculty data set* the other two have much more even distribution. The second data set - *BowlognaBench data set* which is mostly used for the tests is generated with the help of BowlognaBench instance generator [12]. It has 1 263 252 triples with 214 245 values. The third data set, called *Linkedmdb* is taken from DBpedia and is about movies and actors, it has around 7,5 million Triples and again is relatively equally distributed in terms of number of triples per entity (between 6 and 7).

4.4. Measured Storage Systems

4.4.1. SPARQL Benchmark on AllegroGraph

For measuring the performance of an RDF store, AllegroGraph [15] was chosen for several reasons:

- Is free for usage up to a certain relatively high limit - 5 million triples.
- Is very well documented with Java (Jena and Sesame) and Python APIs, supported with very thorough tutorials. As a commercially supported product it is provided also with some technical support free of charge.
- Supports SPARQL.
- Provides some of the best standard benchmark results [4] [2]
- Has a bulk load

The classes used for benchmarking AllegroGrpah are developed with the help of its Java Jena API. For loading the datasets, the bulk loading is used. The retrieval of entity is done through SPARQL queries, which are executed one by one through the API. It should be noted that one query can be devised for more than one entity at the same time, with the help of SPARQL union, which would eliminate the network overhead and would be faster, but this is out of the scope of the measurements in this work.

4.4.2. Open Chord

Open Chord is an open source implementation of Chord DHT[3]. It is developed by the Distributed and Mobile Systems group at the University of Bamberg [31]. Open Chord exposes API for Java applications for handling arbitrary serializable Java objects (key – values) within Chord DHT. It is the most popular Java Chord implementation, it is relatively well documented and has a good amount of implemented features which follow the original Chord specification [40]. Its features include among others:

- Interface for synchronous and asynchronous of a Chord DHT.
- Compatibility with every serializable Java object
- Creation of custom keys to associate data values with.
- Transparent maintenance of Chord DHT routing.
- Transparent replication of stored data.

[3]Distributed Hash Table

Open Chord runs on the Java virtual machine. More than one node can be simulated on the same java virtual machine, or the nodes can be set up on different physical machines, each running locally on a java virtual machine. Even if being the most popular, very cleanly written Java Chord implementations, Open Chord is still a relatively small open source project and not an off-the-shelf finished application. Among its drawbacks are its lacks of customizability and also the fact that it does not provide a ready mechanism to serialize stored objects to disc after a Chord node is switched off. The latter doesn't allow to create a Chord Ring, fill it with data, and then reuse it in the future. Still being with open code, it can be used as a foundation for very robust distributed applications.

4.4.3. Apache Cassandra

Apache Cassandra is an open source distributed NoSQL database management system designed to handle very large amounts of data spread out across cluster of commodity servers. Part of its aims are to provide high availability with no single point of failure. It is based around a structured key-value store. Keys map to multiple values, which are grouped into column families. The column families are fixed when a Cassandra database is created, but columns can be added to a family ar any time. Moreover, columns are added only to specified keys, so different keys can have different numbers of columns in any given family.

The column family is similar to a table of a RDBMS[4] in that it is a container for columns and rows. However, In a relational database, tables are defined with column names and their data types, so the using applications have to conform to this schema, each row contains the same fixed set of columns [9].

In Cassandra, there are the column families that can (and should) define metadata about the columns, but the actual columns that make up a row are determined by the client application. Each row can have a different set of columns.

Column families can be static and dynamic. By dynamic column families further defined column names don't need to stick to predefined names and types, where the static column families follow the defined set of column names.

For all column families, each row is uniquely identified by its row key, similar to the primary key in a relational table. A column family is always partitioned on its row key, and the row key is always implicitly indexed. Empty row keys are not allowed.

4.4.4. MySQL

MySQL is the most popular open source relational database [28]. It has all features typical for relational databases and many ways for tuning the performance, with the help of indices, views, etc. For the purpose of this work it has not been specially configured and tuned and is used only as a comparison to the other systems which are the main point of this work.

[4]Relational Database Management System

4.5. Benchmarking Setup

The results are produced with the help of a test suit developed in Java language. There is a separate project for each of the storage systems tested.

A common part in all the projects is the dataset processing and the file handling for producing results. For parsing the N-Triples of the used datasets NxParser [30] is used. It is an open source library started by the Digital Enterprise Research Institute (DERI) [16]. With its help the triples are grouped in entities as explained in section 4.2 above.

For all the four storage systems there is a separate Java project which executes the tests. Although the projects have common parts, the server connections, the retrieval, update and insertion are all different between the four systems. Still the results that are produced from the tests follow common format and can be exported to tools which produce charts and tables.

4.5.1. Infrastructure

The machine used for single node measurements is with Intel(R) Xeon(R) CPU E5645 @ 2.40GHz, 16GB RAM, Ubuntu 10.10.

The machines that form the 6-node cluster are with Intel Core i7-2600 CPU @ 3.40GHz, 8GB RAM, Ubuntu 11.10.

4.6. Measurements

The following categories tests have been conducted:

1. Measurements of loading times by different configurations;
2. Queries for retrieval random entities;
3. Queries for update of existing values;
4. Queries for insertion of new values for existing entities;
5. Increased load and failure tests.

4.7. Loading

Loading data with AllegroGraph is done in batch, which greatly increases the performance as can be seen in Table 4.1. The network overhead has a great influence as well as can also be seen in the table. The second row per solution is for when data is loaded from the same physical machine where the storage is run and the improvement is visible. The distributed solutions like Cassandra and Chord can benefit from simultaneous loading of only a part of the whole data set. For this purpose the data is divided in 6 which is the number of available machines in the cluster, and every node loads only a sixth of the dataset. Still Chord is very slow, but Cassandra shows considerable improvement when loading 6 equal parts of the data set simultaneously. Cassandra is also tested with the large data set that has more than 7 million triples. As is seen this is of no difficulty, and surpasses the limit of the free AllegroGraph version of 5 million triples.

Storage Solution	**Time in seconds**	**Physical Size in Storage**
AllegroGraph	3 min	136 Mb + 513 default
AllegroGraph same machine	43.5 s	136 Mb + 513 default
Apache Cassandra same machine	3 min	316 Mb
Apache Cassandra	33 min	332 Mb
Ap. Cassandra cluster of 6	1 min 30 s	57 per Node, total 342 Mb
Ap. Cassandra cl. w Linkedmdb	8 min 7 s	187 per N, total 1126 Mb
MySQL	30.5 min	279 Mb + 16 Mb index
MySQL on same machine	4 min 13 s	279 Mb + 16 Mb index
Chord cluster of 6	35 min	in main memory

Table 4.1.: Loading statistics for BowlognaBench-generated dataset, with 1 263 252 triples, 214 275 entities and size of 272 Mb.

AllegroGraph, being capable to load data in batch, proves to be better in loading data than the other 3 solutions. (MySQL can also load in batch, but it needs some preparation of the data.)

4.8. Retrieval

For the retrieval first an array of random numbers is generated. The numbers are chosen between 0 and the total number of triples that are available in the dataset. Then the elements of the array are sorted, so that when the N-Triples file is read sequentially the elements corresponding to the numbers in the array can be selected. If multiple tests have to be done, more arrays of random entities are selected at once, so that the N-Triples file has to be read only once.

For the first retrieval test the systems are set up on a single machine. Figure 4.3. shows the retrieval times for single thread (client). It is clear that Chord provides very slow times, and this is similar for the updates and inserts, so its measurements are not shown further next to the other three systems. Here AllegroGraph proves to be slower than the other two systems and shows also bigger inconsistencies in the retrieval times.

In Figure 4.4. can be seen that when queried simultaneously from multiple clients at a time the systems expectedly decrease their times. This test shows a little advantage for AllegroGraph as it relative increase of response time is smaller than the other two systems, but still the best behaviour is shown by MySQL.

4.8.1. Cluster

Figure 4.5 shows that a Cassandra cluster behaves slightly better than a single node with the same amount of data. It doesn't deteriorate dramatically its response time even with substantial increase of the load. Although the deviation of the response times increases with bigger load.

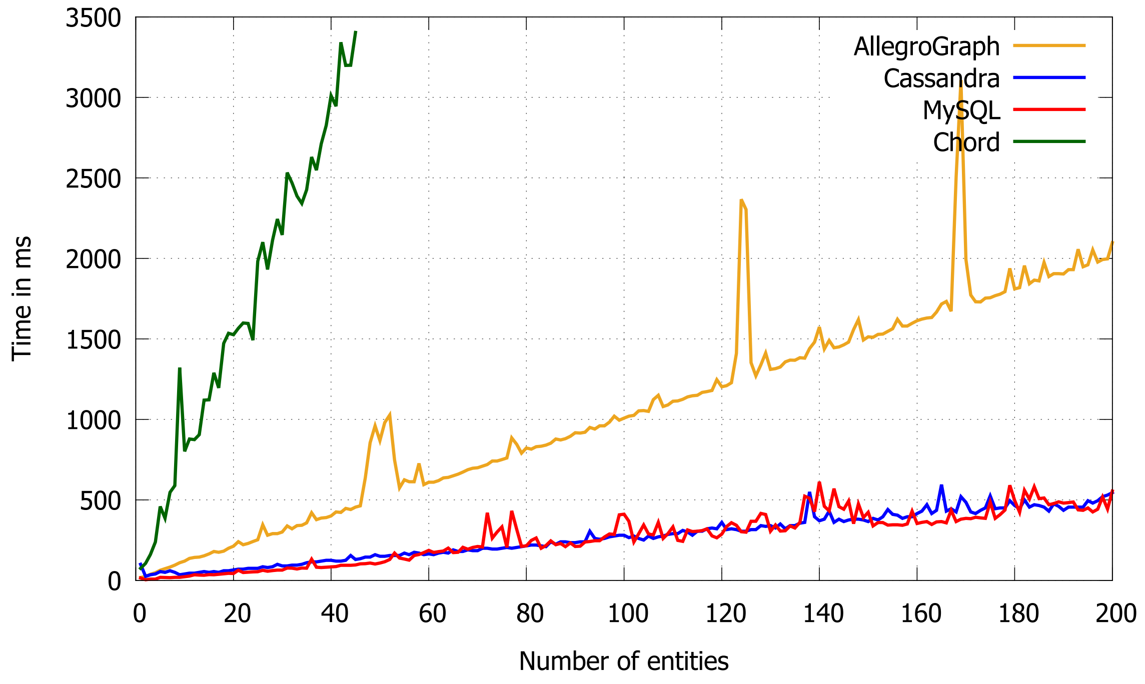


Figure 4.3.: Remote retrieval times for random entities on a single machine.

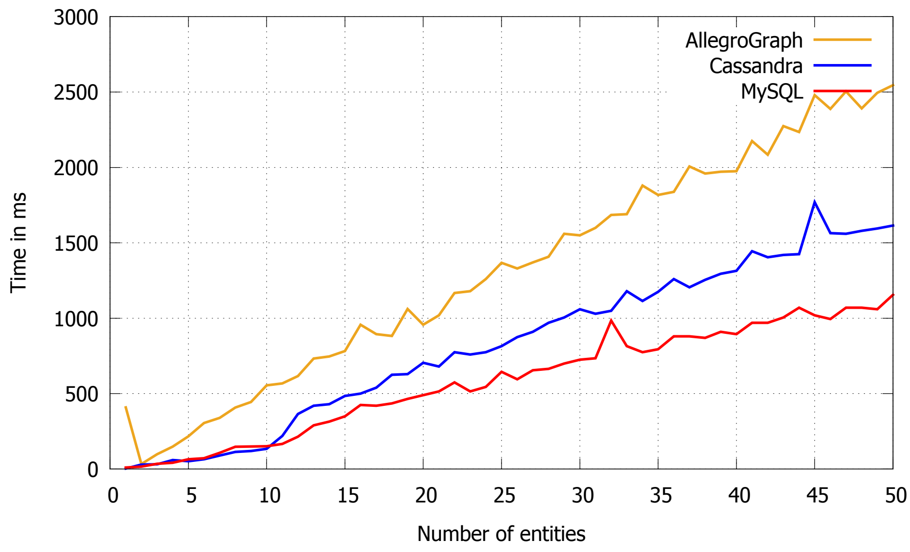


Figure 4.4.: Remote retrieval times for random entities on a single machine, executed simultaneously with 20 threads.

Unlike Cassandra, Chord has worse response times when it is run in cluster, as shown on Figure 4.6. Although this test is done with the smaller Faculty dataset the response times of Chord are much worse compared to the other systems handling with the bigger dataset.

The advantage of Chord comes into consideration when some nodes of the network fail. The following test was done with the Chord ring, constructed of 6 nodes with data loaded

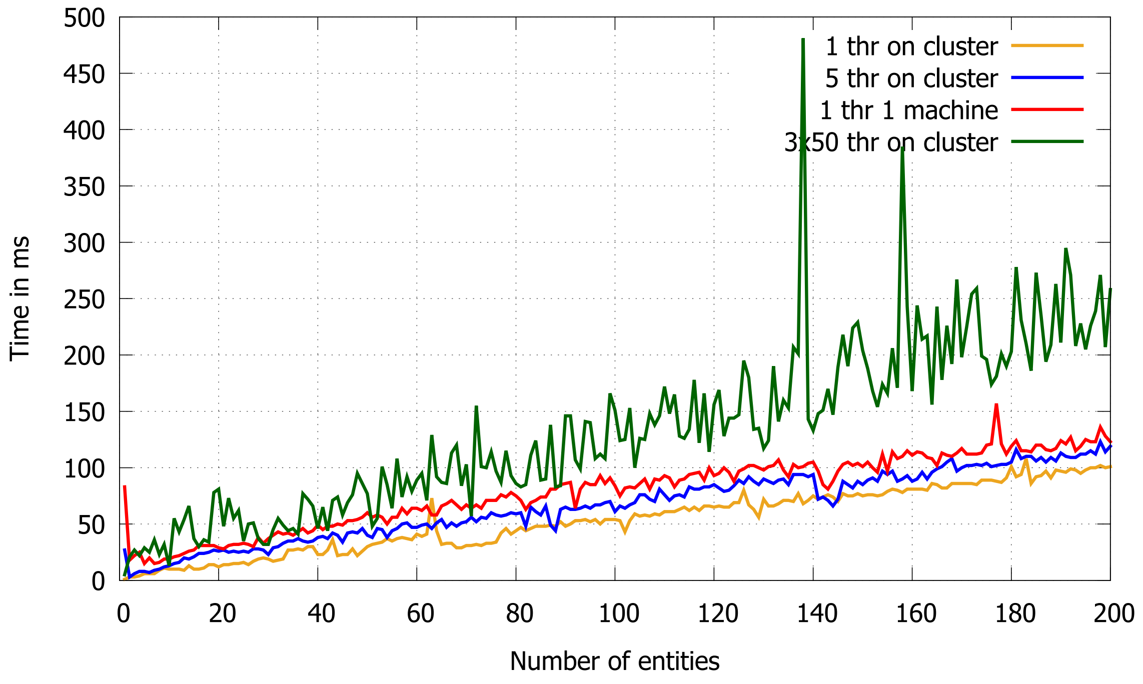


Figure 4.5.: Retrieval time for Cassandra 6-node cluster.

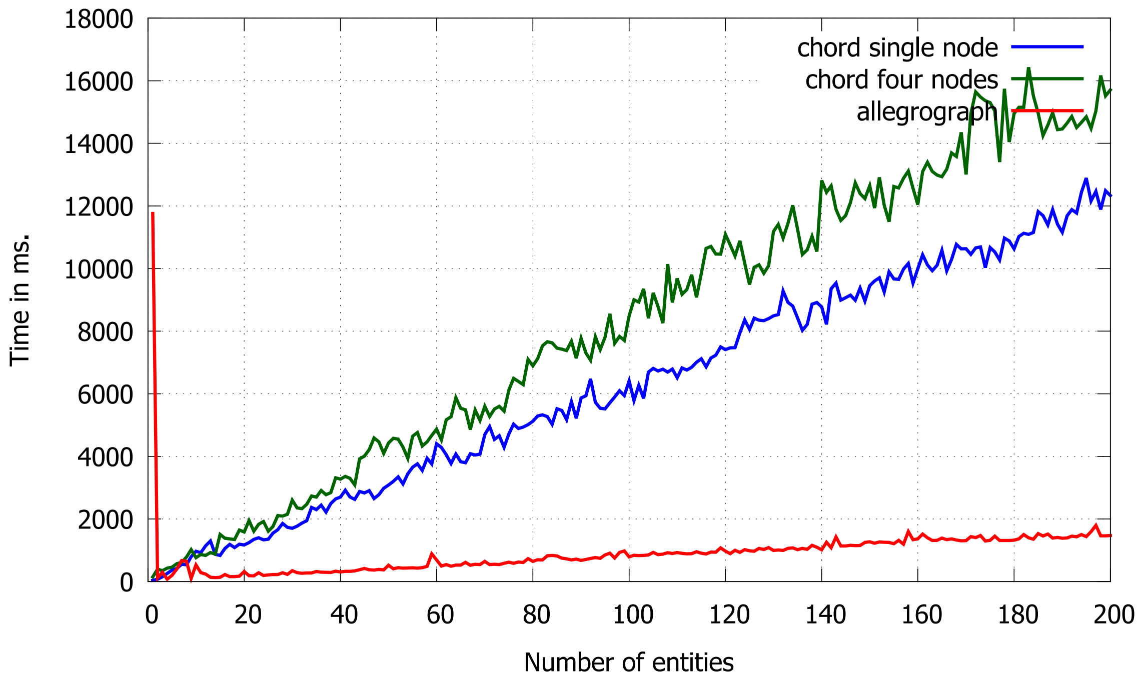


Figure 4.6.: Retrieval times for random entities, AllegroGraph and Chord.

in equal 1/6th parts from each node: executing random queries while shutting down nodes from the ring one by one. The result was that with three nodes running and three nodes down the data was still entirely retrieved. Only when the ring was down to two running nodes, the retrieved entities were around 70%. This was done with no tuning and entirely transparent.

While Cassandra can also replicate data across the servers that take part of the cluster, it needs some setting, and by default by a failure of a single cluster, no data can be retrieved. Also once the number of nodes is decided for a cluster, newly added cluster nodes might not function with proper load balancing in accordance with the cluster.

Generally the difference between Cassandra and Cord is that Cord is much more adaptable, and ad-hoc oriented than Cassandra, while Cassandra provides better performance.

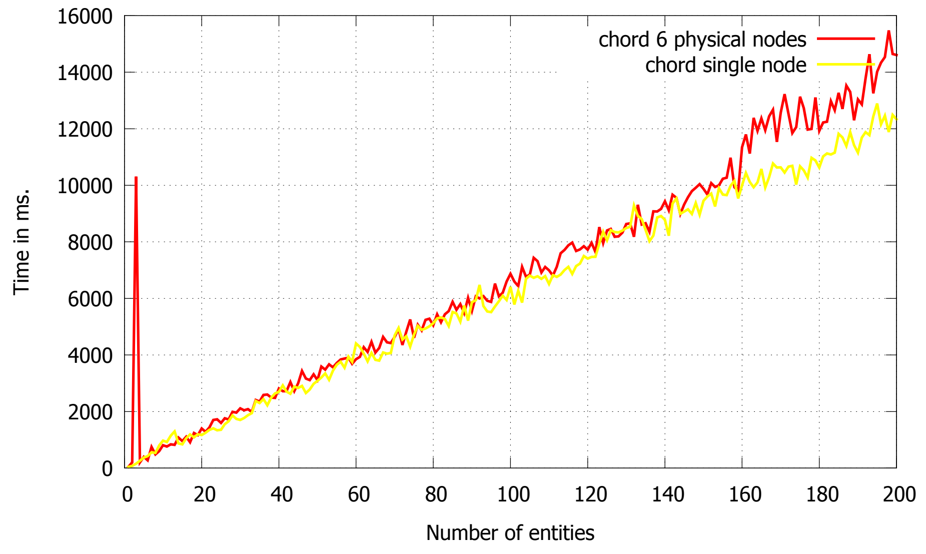


Figure 4.7.: Retrieval times for random entities, Chord single node vs. 6-nodes cluster.

Figure 4.7. shows that by increasing the number and dividing them between nodes can be faster than a single node holding the whole big data.

4.8.2. Throughput

In Figure 4.8. and Figure 4.9. the throughput of entities depending on different factors is shown for Cassandra and AllegroGraph. It is again clear that Cassandra is much faster than AllegroGraph. Both systems show better throughput when there are several simultaneous clients, but this is valid up to a point, when performance starts to decrease. AllegroGraph shows slight decrease in performance when the data set is larger, whereas Cassandra shows the opposite, probably because the larger amount of data means there is less concurrent access to the same memory regions. Contrary to what was seen in Figure 4.5. here Cassandra shows worse performance for the cluster in comparison to the single node. This is probably due to the fact that the cluster tested above has more nodes - 6 rather than 4 and the test above was conducted with better load balancing between more than one node, unlike here. The increased complexity and worse throughput in the case of reasoning by AllegroGraph is also visible.

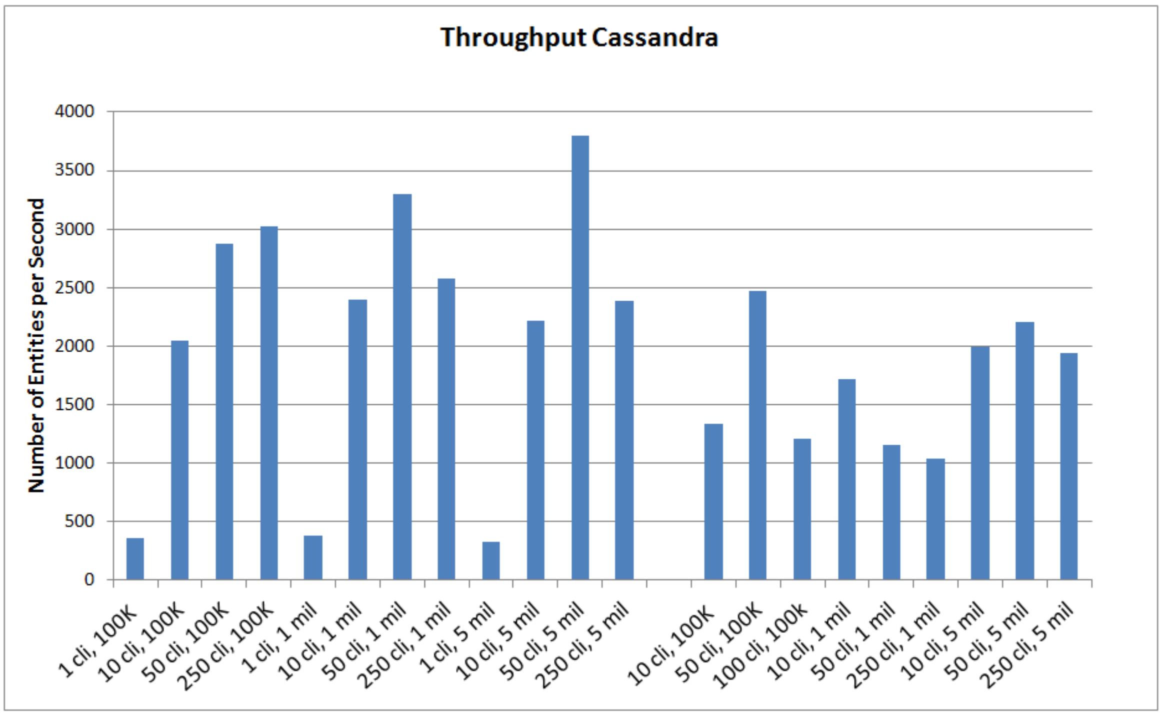


Figure 4.8.: Cassandra's throughput of entities per second, to the right is for a single node and to the left for a cluster of 4 machines.

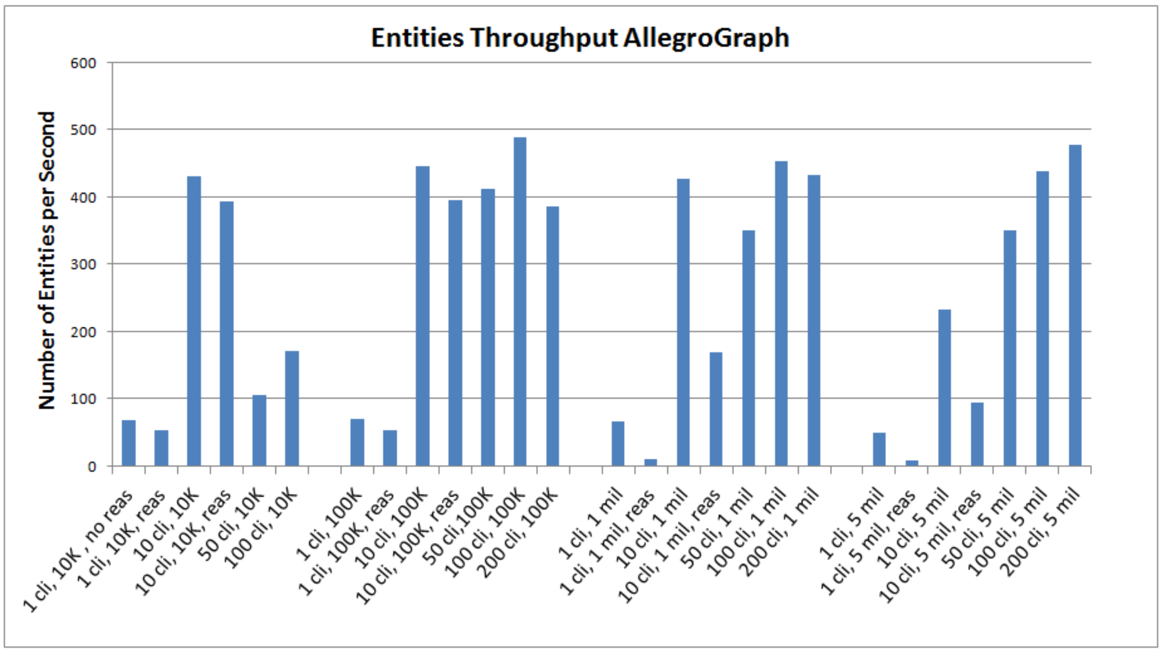


Figure 4.9.: AllegroGraph's throughput of entities per second.

4.9. Update

The update is done also on random entities over and over again but on the same number of entities and then the results have been averaged, including the deviation they show.

AllegroGraph, supporting SPARQL does not have a direct mechanism for update, and does this job by deleting an entity, a triple and then inserting a new one at its place.

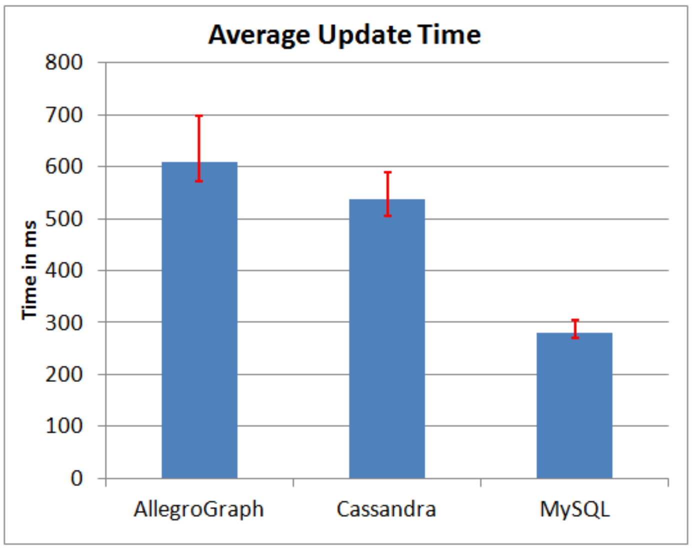


Figure 4.10.: Average update time of 100 entities on a single machine, single client with deviation.

On Figure 4.10. the update rates are shown. The deviation is bigger for the upper bound for all the systems. Still MySQL proves to be faster on a single machine Again the

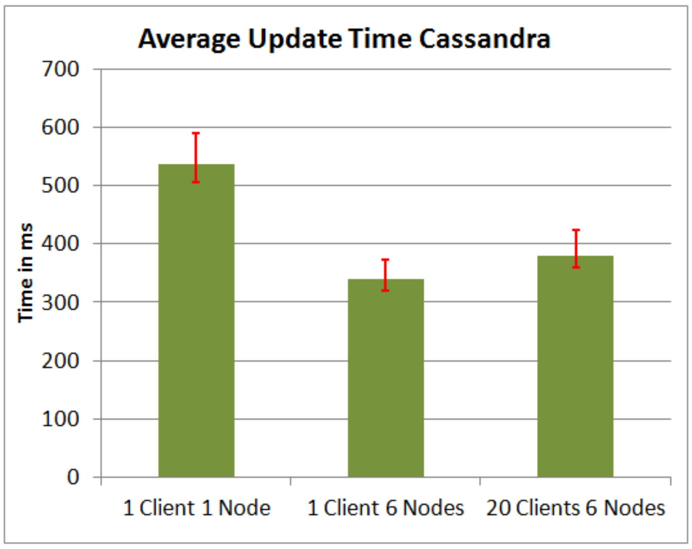


Figure 4.11.: Average update time of 100 entities for Cassandra single node and cluster.

advantage of Cassandra is when it runs as a cluster, as shown in Figure 4.11. for cluster of 6 the response time comes closely to that of MySQL from the previous table.

As before by increasing the concurrent load on Cassandra, there is no dramatic increase of the response time.

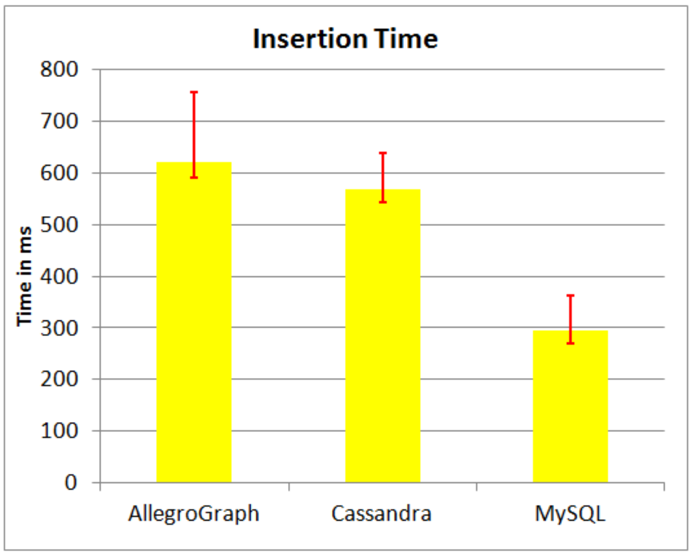


Figure 4.12.: Average insertion time of 100 Entities for Single machine from remote client.

Figure 4.12. shows statistics about insertion. The average times are slightly higher and the deviations are considerably brighter. That is most probably due to occasional reallocation of memory.

4.10. Reasoning and Federation

In an university with two or more faculties there could be the following scenario. There are two (or more) data sets generated from the relational data bases of two faculties, e.g. the faculties of humanities and sciences. There can be some scholars who give lectures at both faculties. If one needs to obtain information for such a scholar, e.g. the times during the week when the scholar gives lectures one could query consecutively the two data sets of the faculties. This would be the simplest solution. There can also be aggregation of the datasets into a single data set, which for example follows an initiative to aggregate and expose information about the whole university. There can also be federated queries directly executed on the separate data sets.

In the case of aggregation of the data sets into one and if the scholar is correctly identified with the same ID, then only a scalable issue remains, to query a bigger amount of triples. Another possibility might be that the scholar is identified differently in the different data sets, i.e. there are two entities representing the scholar. This could be solved with modifications in the data at the time of aggregation similarly to Extract, transform, load[5] . Another possible solution as pointed out in section 2.8 is provided by the OWL construct owl:sameAs, which can indicate the equivalence between two entities. In order this equivalence to be utilized in a query the SPARQL endpoint has to be capable of *inference and reasoning*. It has to have a reasoner which can infer logical consequences from a set of asserted facts and axioms. This way new triples or facts can be inferred about an entity which do not explicitly exist in the data provided. In other words, one could get all the information for the scholar by only knowing one of the IDs used in the

[5]http://en.wikipedia.org/wiki/Extract,_transform,_load

data sets, if they are connected with the owl:sameAs construct. Listing 4.4 shows the results of a query with and without reasoning.

```
PREFIX
hum:<http://www.unifr.ch/humanities.owl#>
 sci:<http://www.unifr.ch/science.owl#>

 select ?s ?p  {?s ?p hum:Professor484>}

 Result without reasoning:

hum:MasterThesis9688 hum:supervisedBy

 Result with reasoning:

hum:MasterThesis9688 hum:supervisedBy
 sci:MasterThesis9688 sci:supervisedBy
```

Listing 4.4: Query results without and with owl:sameAs reasoning.

Another possibility that reasoning in SPARQL provides is the construct owl:InverseOf which is also mentioned in section 2.8. With its help two object properties are defined as inverse or opposite. For example: if the following two triples exist {hum:MasterThesis hum:supervisedBy hum:Professor}and {hum:supervisedBy owl:inverseOf hum:supervises} this would mean that {hum:Professor hum:supervises hum:MasterThesis}. With the help of owl:inverseOf one can get more facts about a professor in one query, see Listing 4.5.

```
PREFIX hum:<http://www.unifr.ch/humanities.owl#>

 select ?s ?p  {hum:Professor484> ?p ?o}

 Result without reasoning:

 owl:type hum:Professor
hum:hasName "Smith"

 Result with reasoning:

 owl:type hum:Professor
hum:hasName "Smith"
hum:supervises hum:MasterThesis321
```

Listing 4.5: Query results for a professor with owl:inverseOf construct.

A number of comparisons using reasoning and multiple data sets are provided in Figure 4.13. Apache Cassandra is used to compare with AllegroGraph. In order to imitate reasoning on Cassandra, multiple queries are executed for one entity. First the original entity is requested, then is checked if it has a triple with predicate owl:sameAs and this way further entities are queried. In the figure the number of faculties represents the number of combined data sets. The *BowlognaBench data set* which is described above is used. The number of departments should indicate the number of owl:sameAs connections, with the possible scenario in mind that in a certain faculty many administrators may be inputting data and it could be badly integrated, so again as between faculties there could be different identifiers for the same professor. AllegroGraph is faster for simple query without reasoning but then with the increasing complexity of reasoning and with increasing the number of connections between different entities and the size of the data, it gets very slow compared to Cassandra. As was shown in the prior measurements, retrieval on Cassandra is generally very fast and it can scale well up to large data sets.

The next Figure 4.14. shows the times of retrieval when data sets are federated directly. Again Cassandra works very well in distributed environment. AllegroGraph shows in-

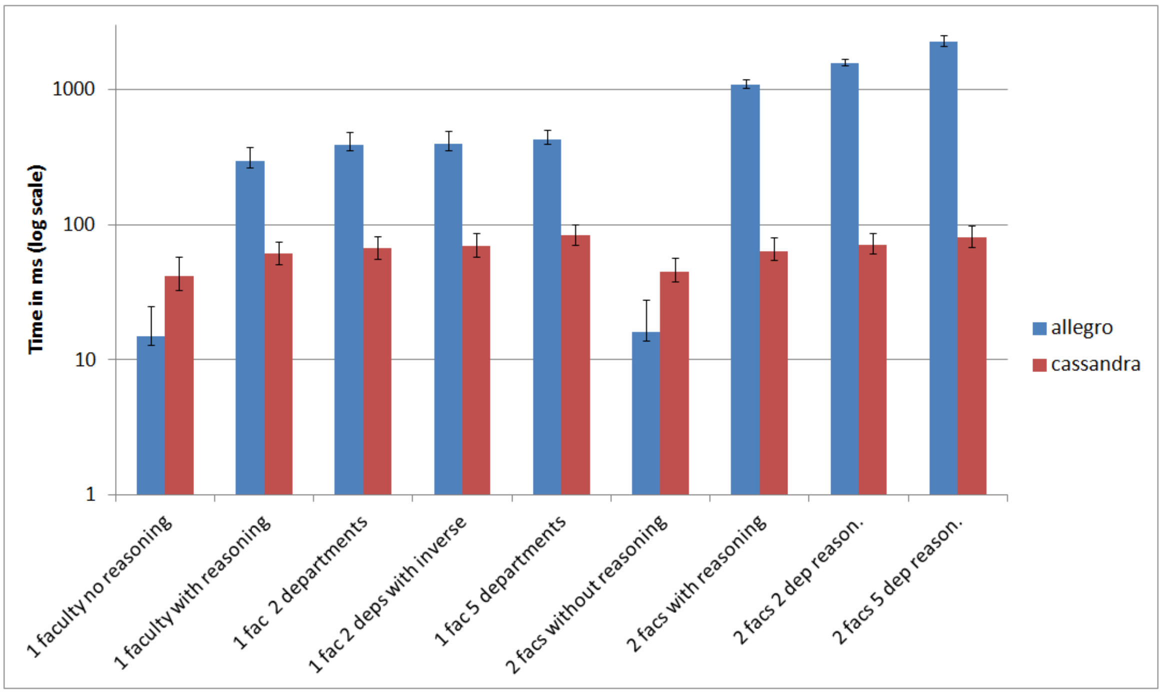


Figure 4.13.: Times for retrieval of a Professor entity with and without reasoning for different number of faculties – data sets and different number of departments – number of owl:sameAs connections.

crease in retrieval time when data sets are federated, but for 2-3 data sources this increase is lower than when more data sets are combined in one and connected with owl:sameAs construct, as shown above. So for integration of a small number of RDF data sets federation works relatively well. When there is federation and reasoning as shown in the figure, then the time deteriorates dramatically, so if this can be avoided it would be advised. Again Cassandra provides fast times, but it has to be kept in mind that the behaviour of Cassandra to work in a similar manner to that of a SPARQL endpoint has to be programmed "by hand" and thoughts on the way data is stored have to be dedicated. To decide which data solution is better, one has to bear in mind the particular usage scenario and the trade-offs which both solutions bear with themselves. Cassandra is a very fast scalable key-value store, but every more complicated querying feature has to be programmed, whereas AllegroGraph SPARQL endpoint provides relatively slower response times, but has the advantage of the powerful SPARQL querying functionalities.

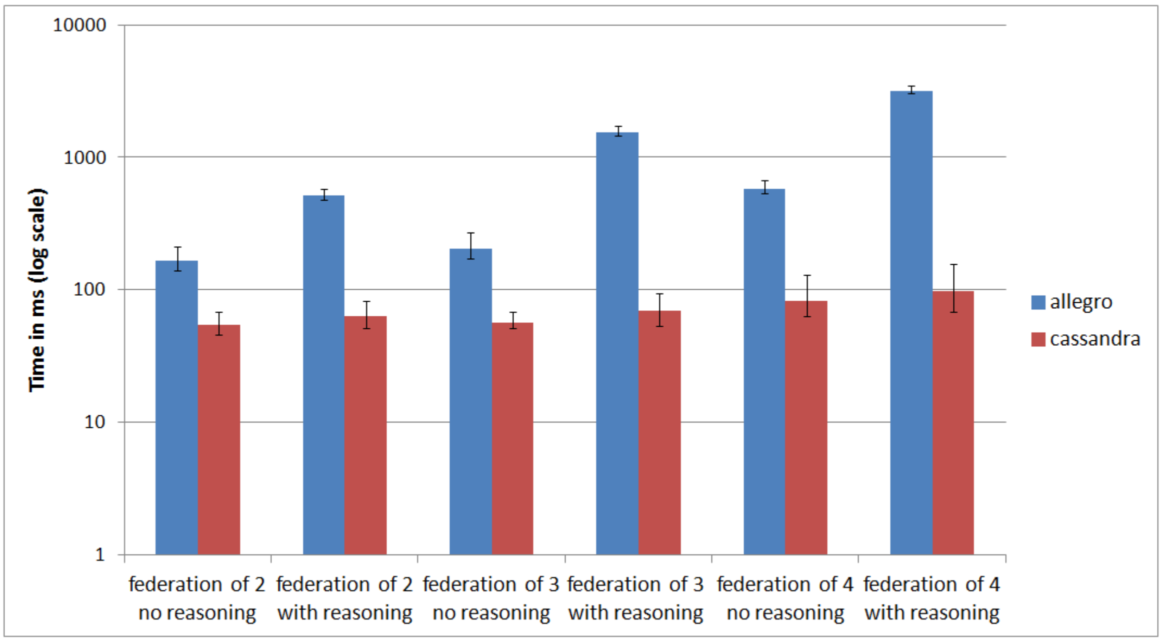


Figure 4.14.: Retrieval times for federated repositories of different number of faculties (data sets) with and without reasoning.

5 Conclusion

The Semantic Web and RDF have been around for more than ten years but lately they are gaining their biggest momentum and enjoy greater and greater popularity across multiple domains. Their adoption in fields such as chemistry, biology, neuroscience, business process analysis, library science, etc., and the successful developments in Linked Open Data have generated an enormous amount of data sets in quantity and in size. Traditional means for integration and reasoning struggle at such large scale. This thesis provides an overview of the implications related to the growth of the Web, the main points of the Semantic Web and the pursuit of integration of separate knowledge bases. It outlines the main features and characteristics of several systems that could facilitate the realization of the Semantic web and resource integration: Domain Name System, Digital Object Architecture, Entity Name System, Chord and Coral. Further an approach for handling RDF data as key – value pairs is presented in the context of data management solutions that are not specifically designed for RDF – Apache Cassandra, Open Chord and MySQL, using AllegroGraph RDF storage system as a base line. Using this approach the performance of the above mentioned systems is measured. Their main qualities and disadvantages are exposed, with the help of several types of tests based on: data loading, retrieval, updates, insertion, where different configurations for clusters or single machines are used for the benchmarking. A Java benchmarking suit is developed for the automation and facilitation of all the tests on the different systems.

The results of the benchmarks conducted for the four solutions emphasized their different purposes but showed that in certain use cases certain system could be considered most appropriate. Apache Cassandra could utilize simple commodity machines for creating a cluster that can handle enormous amounts of data with very reasonable response times; AllegroGraph would be best when more than just simple queries are needed, as it implements SPARQL; Chord is best when large amounts of data have to be handled by a cluster with constantly changing nodes; MySQL could provide fast performance and simple integration with not large datasets.

For the future, more thoughts can be invested in finding different scenarios close to the real world and elaborating on the queries that are executed on the data with the different solutions. Tuning and configuration is also possible with the solutions, and this requires certain amount of time and effort after one is familiar with the different storage solutions.

A
Common Acronyms

OWL Web Ontology Language

RDBMS Relational Database Management System

SPARQL SPARQL Protocol and RDF Query Language

RDFS RDF Schema

P2P peer-to-peer

RDF Resource Description Framework

DHT Distributed Hash Table

URI Uniform Resource Identifier

URL Uniform Resource Identifier

URI Uniform Resource Identifier

WWW World Wide Web

DOA Digital Object Architecture

ENS Entity Name System

DNS Domain Name System

W3C World Wide Web Consortium

B
License of the Documentation

References

[1] Keith Alexander and Michael Hausenblas. Describing linked datasets - on the design and usage of void, the vocabulary of interlinked datasets. In *In Linked Data on the Web Workshop (LDOW 09), in conjunction with 18th International World Wide Web Conference (WWW 09*, 2009.

[2] Allegrograph, benchmark results for lubm(8000). http://www.franz.com/agraph/allegrograph/agraph_bench_lubm.lhtml (accessed July 30, 2012).

[3] Cool URIs don't change. http://www.w3.org/Provider/Style/URI.html (accessed July 30, 2012).

[4] Triple Store Management. http://www.chrisfrymann.com/category/infrastructure/performance/ (accessed July 30, 2012).

[5] Architecture of the World Wide Web, Volume One, 2004. http://www.w3.org/TR/2004/REC-webarch-20041215/ (accessed July 6, 2012).

[6] Bringing big data to the enterprise. http://www-01.ibm.com/software/data/bigdata/ (accessed June 29, 2012).

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, September 2009.

[8] P. Bouquet, H. Stoermer, C. Niederee, and A. Maa. Entity name system: The backbone of an open and scalable web of data. In *Semantic Computing, 2008 IEEE International Conference on*, pages 554 –561, aug. 2008.

[9] Apache Cassandra Documentation. http://www.datastax.com/docs/1.0/ddl/column_family (accessed July 30, 2012).

[10] Philippe Cudre-Mauroux, Suchit Agarwal, and Karl Aberer. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing*, 11(5):36–44, September 2007.

[11] Dbpedia. http://dbpedia.org/About (accessed August 28, 2012).

[12] Gianluca Demartini, Iliya Enchev, Joël Gapany, and Philippe Cudré-Mauroux. Bowlognabench-benchmarking rdf analytics. In *First International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2011)*, pages 29–43, June 2011.

[13] Cool uris for the semantic web. http://www.w3.org/TR/cooluris/ (accessed August 22, 2012).

[14] Dereferencing http uris. http://www.w3.org/2001/tag/doc/httpRange-14/2007-05-31/HttpRange-14 (accessed August 22, 2012).

[15] AllegroGraph, persistent graph database for RDF. http://www.franz.com/agraph/allegrograph/ (accessed July 30, 2012).

[16] Digital Enterprise Research Institute, Galway, Ireland. http://www.deri.ie/ (accessed July 30, 2012).

[17] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accessed June 30, 2005).

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[19] Michael J. Freedman. Experiences with coralcdn: a five-year operational view. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[20] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space.* Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[21] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[22] Linked Data - Connect Distributed Data across the Web. http://linkeddata.org/ (accessed July 30, 2012).

[23] The Linking Open Data cloud diagram. http://richard.cyganiak.de/2007/10/lod/ (accessed August 22, 2012).

[24] Statistics on data sets. http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics (accessed August 22, 2012).

[25] Günter Ludwig and Harth Andreas. Cumulusrdf: Linked data management on nested key-value stores. In *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011)*, 2011.

[26] Metadata Architecture. http://www.w3.org/DesignIssues/Metadata.html (accessed July 13, 2012).

[27] P. Mockapetris. DOMAIN NAMES - CONCEPTS AND FACILITIES. RFC 1034 (Informational), November 1987.

[28] MySQL. http://www.mysql.com/ (accessed July 30, 2012).

[29] The Myth of Names and Addresses. http://www.w3.org/DesignIssues/NameMyth.html (accessed July 30, 2012).

[30] NxParser. http://code.google.com/p/nxparser/ (accessed July 30, 2012).

[31] Open Chord. http://open-chord.sourceforge.net/ (accessed July 12, 2012).

[32] National Information Standards Organization. *Understanding Metadata.* NISO Press, National Information Standards Organization, 4733 Bethesda Avenue, Suite 300, Bethesda, MD 20814 USA, 2004.

[33] Owl web ontology language guide. http://www.ksl.stanford.edu/software/jtp/doc/owl-reasoning.html (accessed August 28, 2012).

[34] Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/ (accessed August 28, 2012).

[35] Rdf/rdfs reasoning capabilities. http://www.w3.org/TR/owl-guide/ (accessed August 28, 2012).

[36] Sparql 1.1 federated query. http://www.w3.org/TR/sparql11-federated-query/ (accessed August 28, 2012).

[37] Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/ (accessed August 28, 2012).

[38] Sparql update. http://www.w3.org/Submission/SPARQL-Update/ (accessed August 28, 2012).

[39] State of the lod cloud. http://thedatahub.org/dataset/dbpedia (accessed August 28, 2012).

[40] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[41] S. Sun, L. Lannom, and B. Boesch. Handle System Overview. RFC 3650 (Informational), November 2003.

[42] Semantic Web Road map . http://www.w3.org/DesignIssues/Semantic.html (accessed July 6, 2012).

[43] Domain Name System. http://en.wikipedia.org/wiki/Domain_Name_System (accessed July 30, 2012).